Django celery 研究说明文档

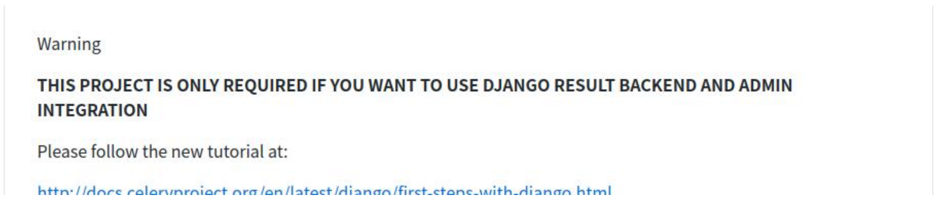
选型

Celery(https://docs.celeryproject.org/en/stable/django/first-steps-with-django.html#using-celery-with-django
)

django-celery（https://github.com/celery/django-celery）

之前的老版本的时候 celery 不支持 django 所以有了第三方库 django-celery, 其中

django-celery 最重要的一个特点或者说这个包的最重要的一个意义是：


Warning

THIS PROJECT IS ONLY REQUIRED IF YOU WANT TO USE DJANGO RESULT BACKEND AND ADMIN INTEGRATION

Please follow the new tutorial at:

http://docs.celeryproject.org/en/latest/django/first-steps-with-django.html

也就是把 celery 的相关结果和运行状态保存在 django 的 orm 中，也就是 mysql 或者 psql

等中。但是现在 celery 官方已经有对应的第三方库 django-celery-results 保存结果以及

django-celery-beat 来实现定时器任务的相关管理。

接下来会分别用官方 celery 模块和 django-celery 实现对应的功能 ,并且对比他们的却别和

联系。

Celery

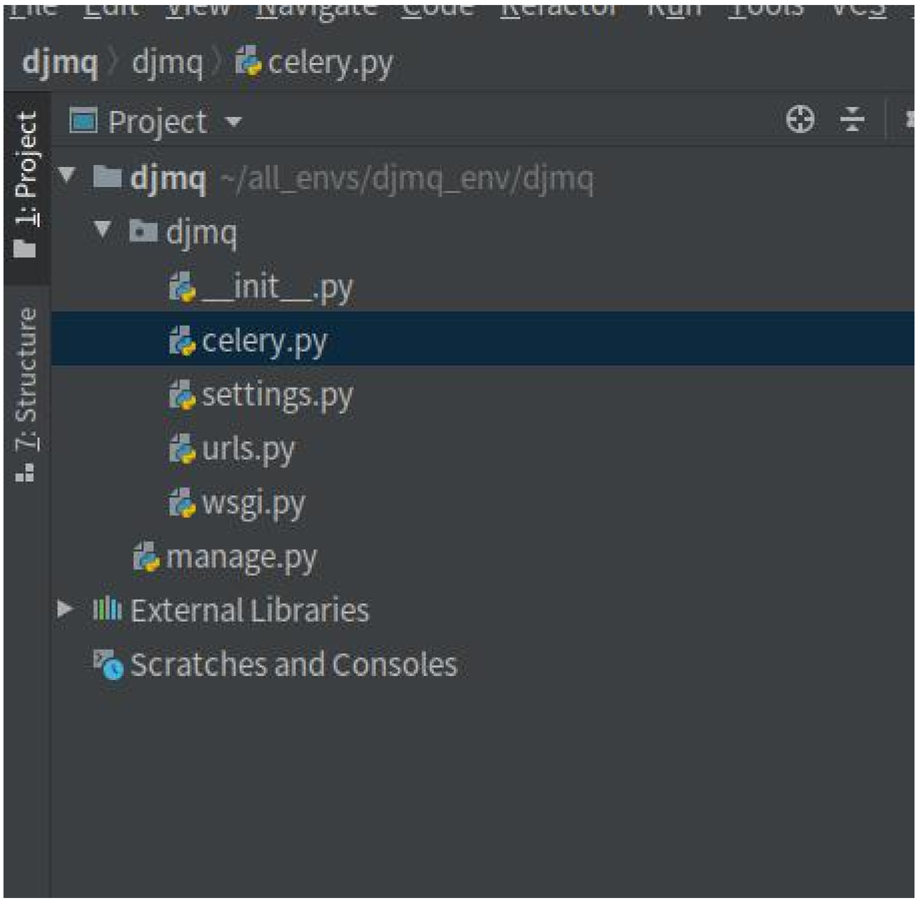
安装

```
ip install -i https://pypi.douban.com/simple django==2.0.0
pip install -i https://pypi.douban.com/simple redis
```

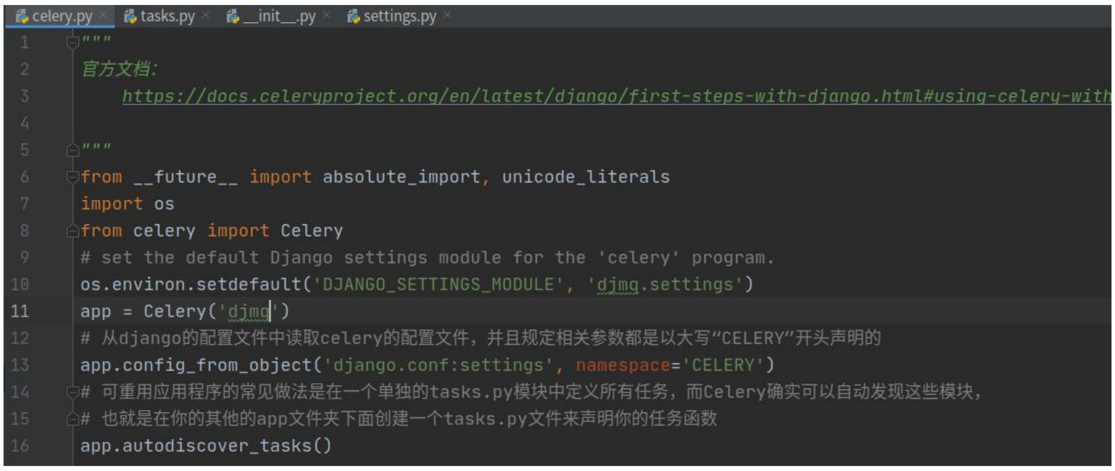
```
django-admin.py startproject djmq # 创建 django 项目
```

配置

在项目目录下创建一个 celery.py 并添加相关配置



```
djmq 〉 djmq 〉 celery.py
Project
▼ djmq ~/all_envs/djmq_env/djmq
    ▼ djmq
        __init__.py
        celery.py
        settings.py
        urls.py
        wsgi.py
      manage.py
▶ External Libraries
  Scratches and Consoles
```

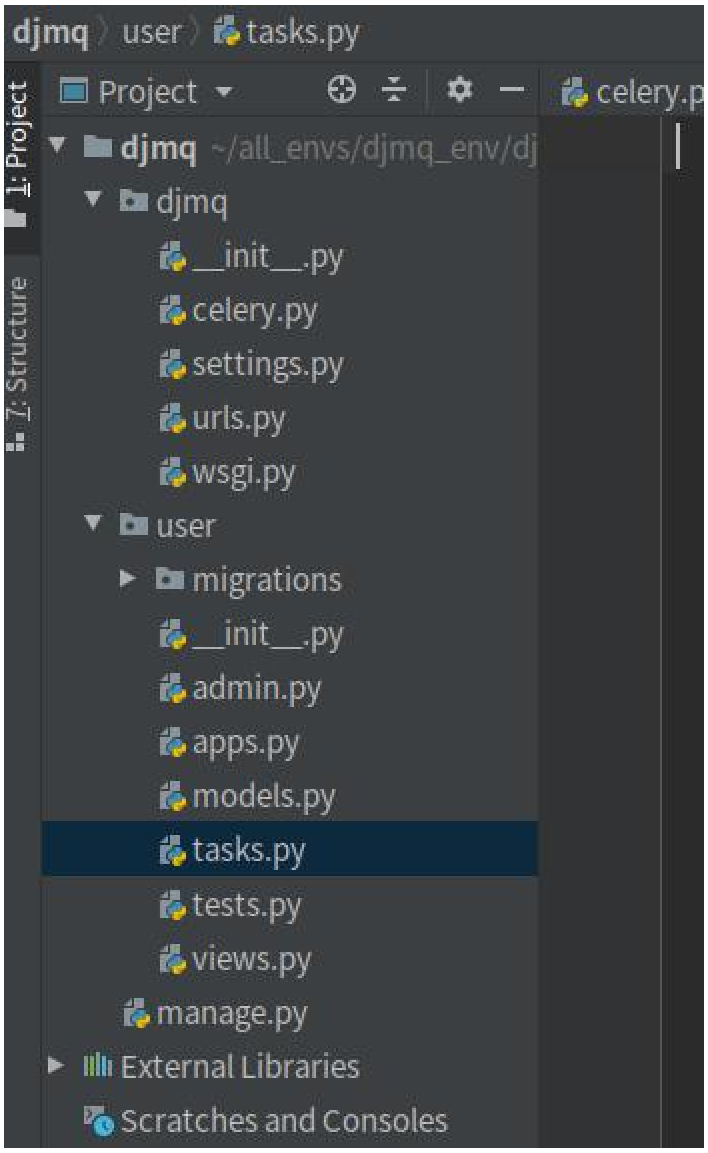


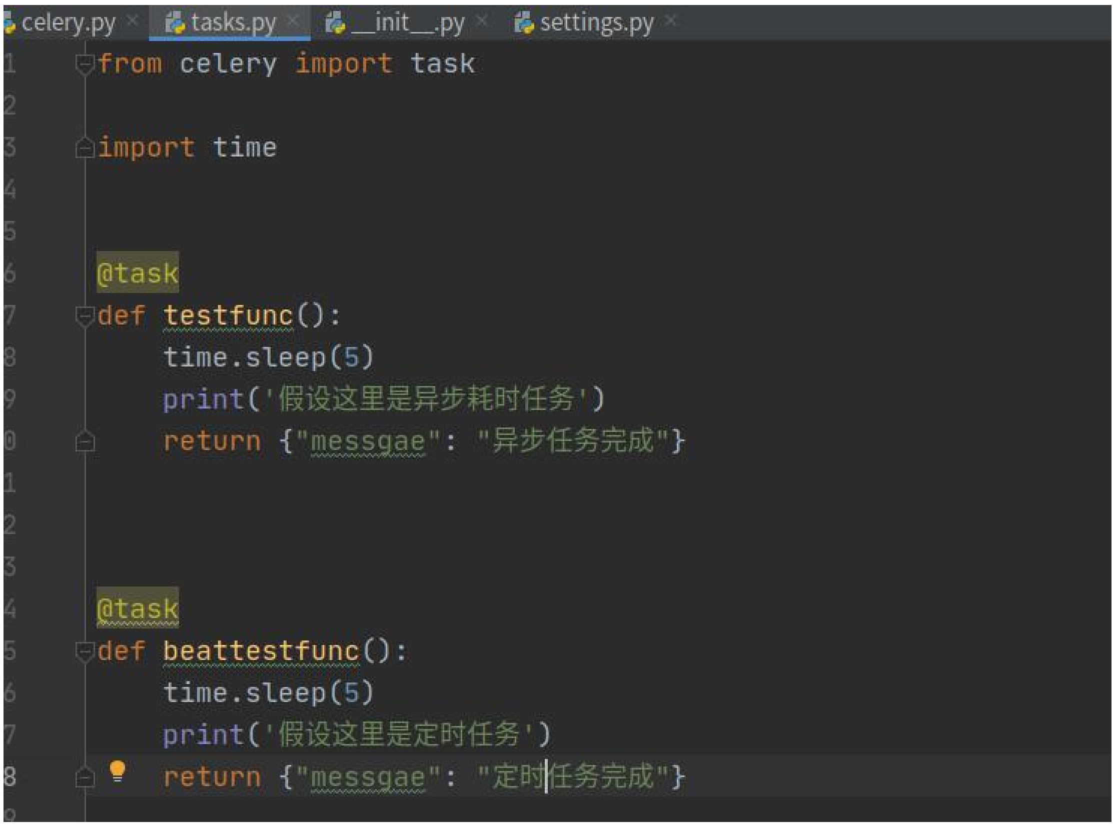
```python
"""
官方文档:
    https://docs.celeryproject.org/en/latest/django/first-steps-with-django.html#using-celery-with

"""
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery
# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'djmq.settings')
app = Celery('djmq')
# 从django的配置文件中读取celery的配置文件，并且规定相关参数都是以大写“CELERY”开头声明的
app.config_from_object('django.conf:settings', namespace='CELERY')
# 可重用应用程序的常见做法是在一个单独的tasks.py模块中定义所有任务，而Celery确实可以自动发现这些模块，
# 也就是在你的其他的app文件夹下面创建一个tasks.py文件来声明你的任务函数
app.autodiscover_tasks()
```

使用

配置完成后 celery 会在创建的 app 应用下主动发现 tasks.py 默认 tasks.py 就是保存 celery

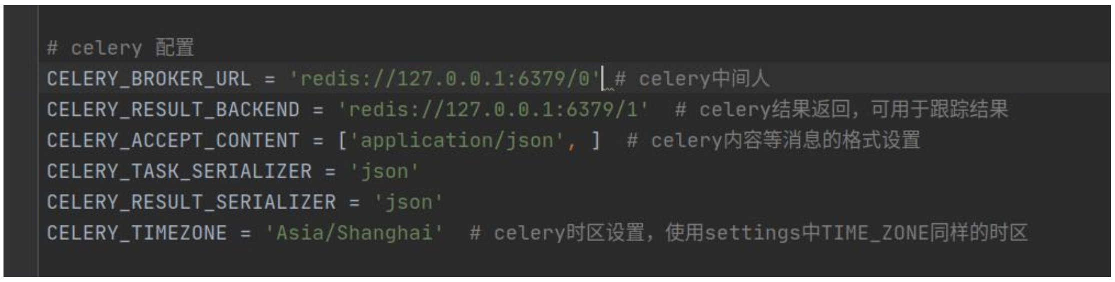
任务的文件，所以你可以在 tasks.py 下定义你的异步耗时任务。




celery.py tasks.py __init__.py settings.py

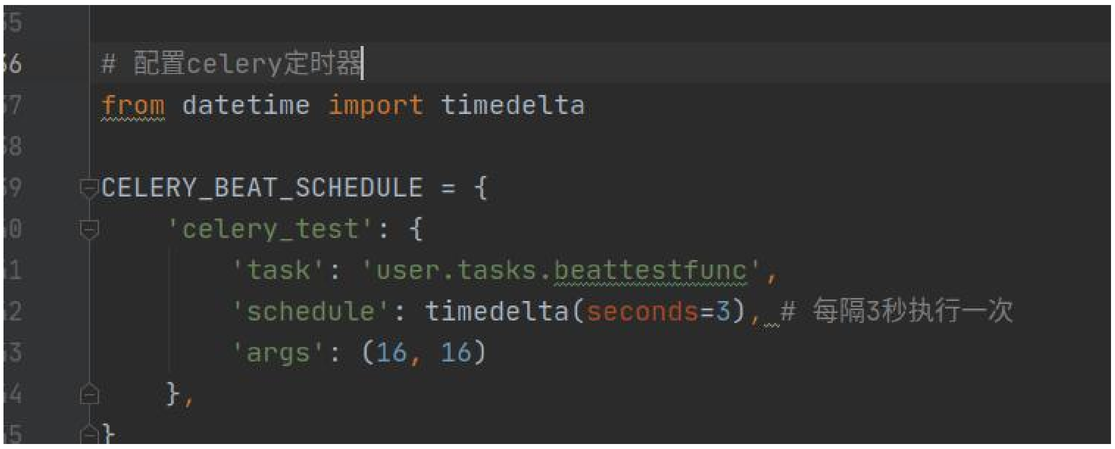
```python
from celery import task

import time


@task
def testfunc():
    time.sleep(5)
    print('假设这里是异步耗时任务')
    return {"messgae": "异步任务完成"}



@task
def beattestfunc():
    time.sleep(5)
    print('假设这里是定时任务')
    return {"messgae": "定时任务完成"}
```

配置 celery 选项



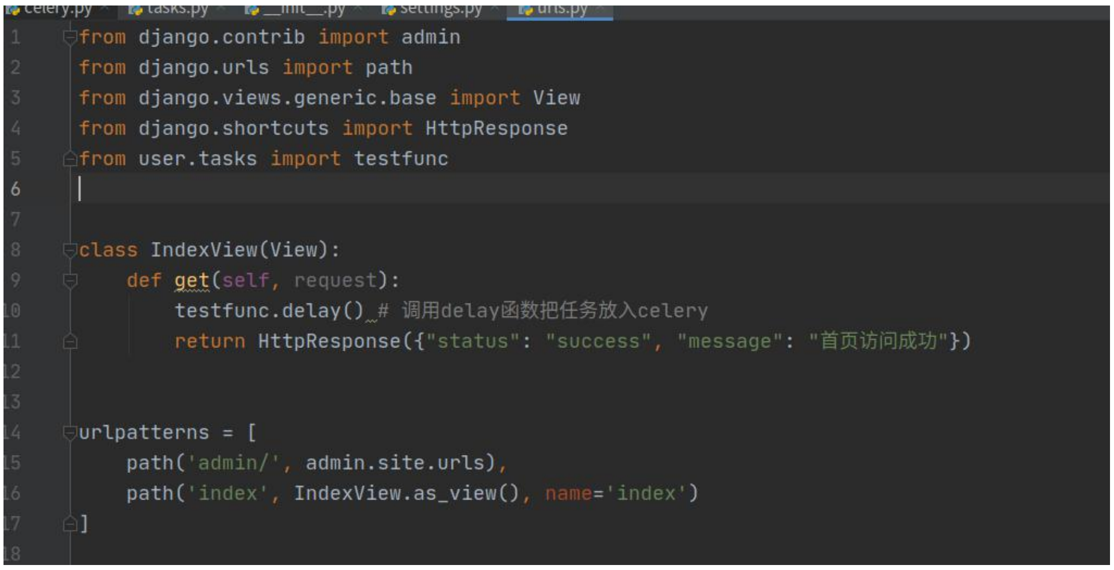
```python
# celery 配置
CELERY_BROKER_URL = 'redis://127.0.0.1:6379/0' # celery中间人
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/1'  # celery结果返回，可用于跟踪结果
CELERY_ACCEPT_CONTENT = ['application/json', ]  # celery内容等消息的格式设置
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'Asia/Shanghai'  # celery时区设置，使用settings中TIME_ZONE同样的时区
```

声明定时任务其实就是 linux 的 contrab 每三秒执行一次



```python

# 配置celery定时器
from datetime import timedelta

CELERY_BEAT_SCHEDULE = {
    'celery_test': {
        'task': 'user.tasks.beattestfunc',
        'schedule': timedelta(seconds=3), # 每隔3秒执行一次
        'args': (16, 16)
    },
}
```

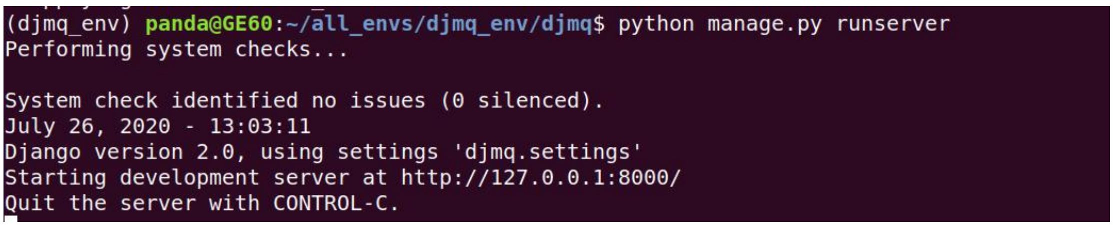
编写简单路由模拟触发异步任务



```python
from django.contrib import admin
from django.urls import path
from django.views.generic.base import View
from django.shortcuts import HttpResponse
from user.tasks import testfunc


class IndexView(View):
    def get(self, request):
        testfunc.delay() # 调用delay函数把任务放入celery
        return HttpResponse({"status": "success", "message": "首页访问成功"})


urlpatterns = [
    path('admin/', admin.site.urls),
    path('index', IndexView.as_view(), name='index')
]
```

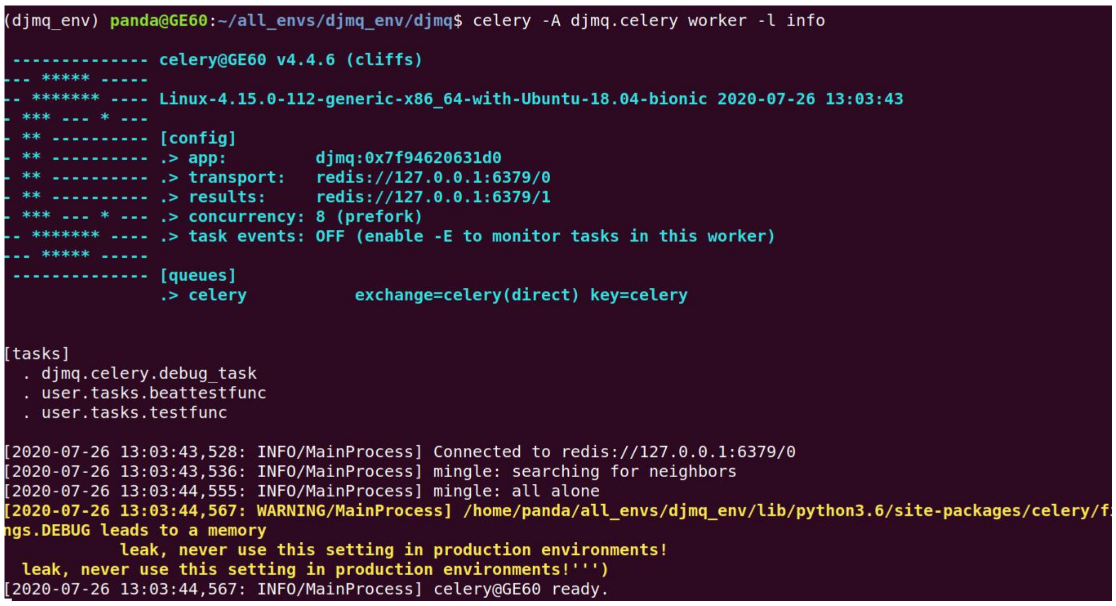
正常使用

启动 django



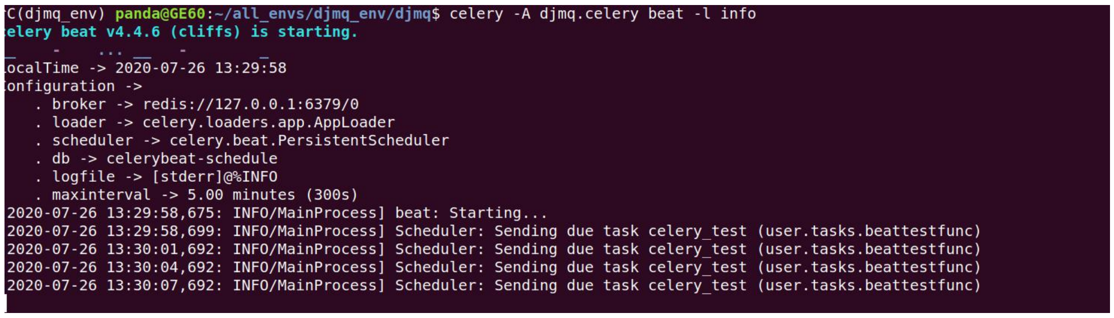
```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
July 26, 2020 - 13:03:11
Django version 2.0, using settings 'djmq.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

启动 celery 的 woker 生产者



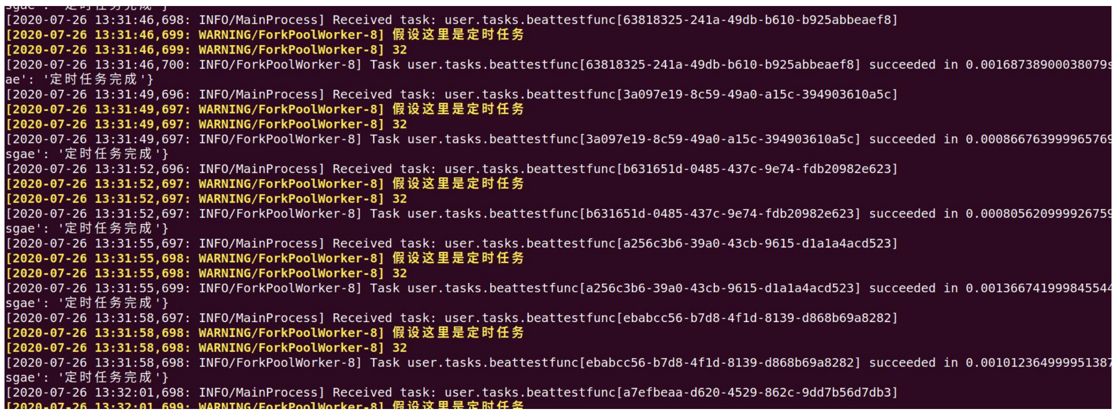
```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery worker -l info

 -------------- celery@GE60 v4.4.6 (cliffs)
--- ***** -----
-- ******* ---- Linux-4.15.0-112-generic-x86_64-with-Ubuntu-18.04-bionic 2020-07-26 13:03:43
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         djmq:0x7f94620631d0
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . djmq.celery.debug_task
  . user.tasks.beattestfunc
  . user.tasks.testfunc

[2020-07-26 13:03:43,528: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2020-07-26 13:03:43,536: INFO/MainProcess] mingle: searching for neighbors
[2020-07-26 13:03:44,555: INFO/MainProcess] mingle: all alone
[2020-07-26 13:03:44,567: WARNING/MainProcess] /home/panda/all_envs/djmq_env/lib/python3.6/site-packages/celery/f
ngs.DEBUG leads to a memory
            leak, never use this setting in production environments!
  leak, never use this setting in production environments!''')
[2020-07-26 13:03:44,567: INFO/MainProcess] celery@GE60 ready.
```

启动 celery 的 beat 定时任务



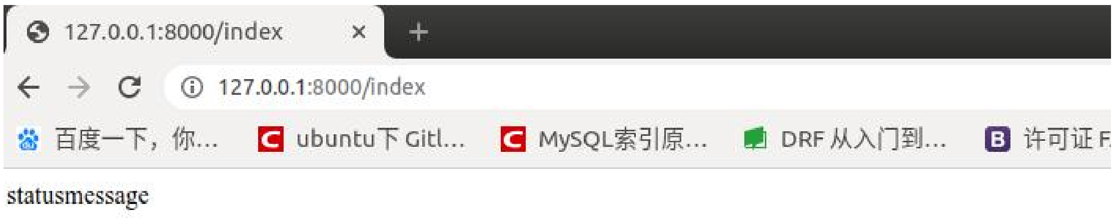
```
C(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery beat -l info
elery beat v4.4.6 (cliffs) is starting.
_    -    ... __   -        _
ocalTime -> 2020-07-26 13:29:58
onfiguration ->
    . broker -> redis://127.0.0.1:6379/0
    . loader -> celery.loaders.app.AppLoader
    . scheduler -> celery.beat.PersistentScheduler
    . db -> celerybeat-schedule
    . logfile -> [stderr]@%INFO
    . maxinterval -> 5.00 minutes (300s)
2020-07-26 13:29:58,675: INFO/MainProcess] beat: Starting...
2020-07-26 13:29:58,699: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
2020-07-26 13:30:01,692: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
2020-07-26 13:30:04,692: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
2020-07-26 13:30:07,692: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
```

定时任务已经在往队列中计加入了，如此同时消费者已经在消费了



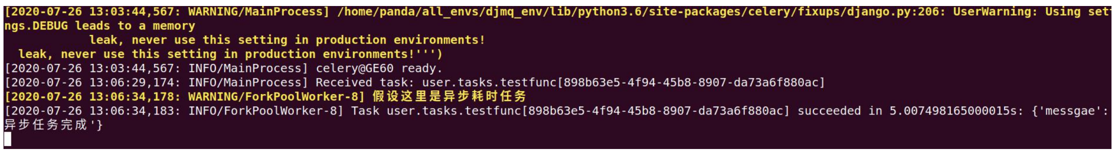
```
[2020-07-26 13:31:46,698: INFO/MainProcess] Received task: user.tasks.beattestfunc[63818325-241a-49db-b610-b925abbeaef8]
[2020-07-26 13:31:46,699: WARNING/ForkPoolWorker-8] 假设这里是定时任务
[2020-07-26 13:31:46,699: WARNING/ForkPoolWorker-8] 32
[2020-07-26 13:31:46,700: INFO/ForkPoolWorker-8] Task user.tasks.beattestfunc[63818325-241a-49db-b610-b925abbeaef8] succeeded in 0.00168738900038079s
ae': '定时任务完成'}
[2020-07-26 13:31:49,696: INFO/MainProcess] Received task: user.tasks.beattestfunc[3a097e19-8c59-49a0-a15c-394903610a5c]
[2020-07-26 13:31:49,697: WARNING/ForkPoolWorker-8] 假设这里是定时任务
[2020-07-26 13:31:49,697: WARNING/ForkPoolWorker-8] 32
[2020-07-26 13:31:49,697: INFO/ForkPoolWorker-8] Task user.tasks.beattestfunc[3a097e19-8c59-49a0-a15c-394903610a5c] succeeded in 0.00086676399996576
sgae': '定时任务完成'}
[2020-07-26 13:31:52,696: INFO/MainProcess] Received task: user.tasks.beattestfunc[b631651d-0485-437c-9e74-fdb20982e623]
[2020-07-26 13:31:52,697: WARNING/ForkPoolWorker-8] 假设这里是定时任务
[2020-07-26 13:31:52,697: WARNING/ForkPoolWorker-8] 32
[2020-07-26 13:31:52,697: INFO/ForkPoolWorker-8] Task user.tasks.beattestfunc[b631651d-0485-437c-9e74-fdb20982e623] succeeded in 0.00080562099992675
sgae': '定时任务完成'}
[2020-07-26 13:31:55,697: INFO/MainProcess] Received task: user.tasks.beattestfunc[a256c3b6-39a0-43cb-9615-d1a1a4acd523]
[2020-07-26 13:31:55,698: WARNING/ForkPoolWorker-8] 假设这里是定时任务
[2020-07-26 13:31:55,698: WARNING/ForkPoolWorker-8] 32
[2020-07-26 13:31:55,699: INFO/ForkPoolWorker-8] Task user.tasks.beattestfunc[a256c3b6-39a0-43cb-9615-d1a1a4acd523] succeeded in 0.00136674199984554
sgae': '定时任务完成'}
[2020-07-26 13:31:58,697: INFO/MainProcess] Received task: user.tasks.beattestfunc[ebabcc56-b7d8-4f1d-8139-d868b69a8282]
[2020-07-26 13:31:58,698: WARNING/ForkPoolWorker-8] 假设这里是定时任务
[2020-07-26 13:31:58,698: WARNING/ForkPoolWorker-8] 32
[2020-07-26 13:31:58,698: INFO/ForkPoolWorker-8] Task user.tasks.beattestfunc[ebabcc56-b7d8-4f1d-8139-d868b69a8282] succeeded in 0.00101236499995138
sgae': '定时任务完成'}
[2020-07-26 13:32:01,698: INFO/MainProcess] Received task: user.tasks.beattestfunc[a7efbeaa-d620-4529-862c-9dd7b56d7db3]
[2020-07-26 13:32:01,699: WARNING/ForkPoolWorker-8] 假设这里是定时任务
```

访问路由触发异步任务



statusmessage

任务执行成功

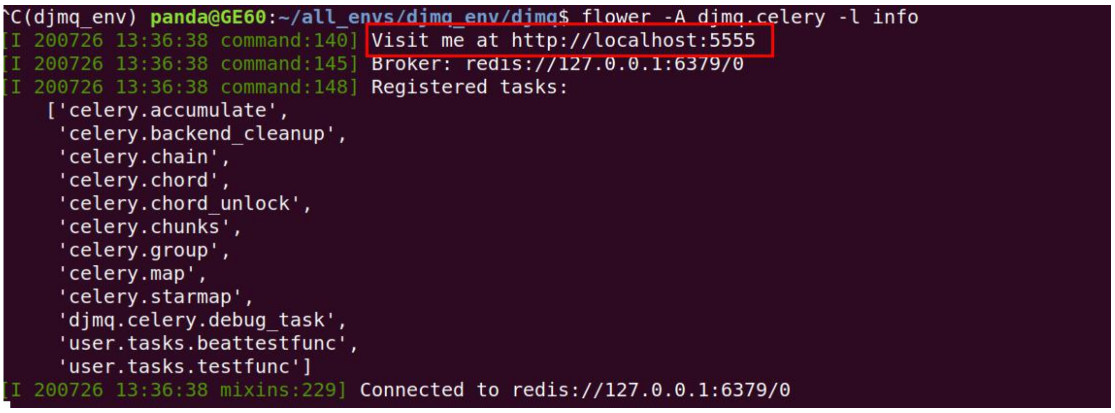


```
[2020-07-26 13:03:44,567: WARNING/MainProcess] /home/panda/all_envs/djmq_env/lib/python3.6/site-packages/celery/fixups/django.py:206: UserWarning: Using set
ngs.DEBUG leads to a memory
            leak, never use this setting in production environments!
  leak, never use this setting in production environments!''')
[2020-07-26 13:03:44,567: INFO/MainProcess] celery@GE60 ready.
[2020-07-26 13:06:29,174: INFO/MainProcess] Received task: user.tasks.testfunc[898b63e5-4f94-45b8-8907-da73a6f880ac]
[2020-07-26 13:06:34,178: WARNING/ForkPoolWorker-8] 假设这里是异步耗时任务
[2020-07-26 13:06:34,183: INFO/ForkPoolWorker-8] Task user.tasks.testfunc[898b63e5-4f94-45b8-8907-da73a6f880ac] succeeded in 5.007498165000015s: {'messgae':
异步任务完成'}
```

监控 flower

https://flower-docs-cn.readthedocs.io/zh/latest/

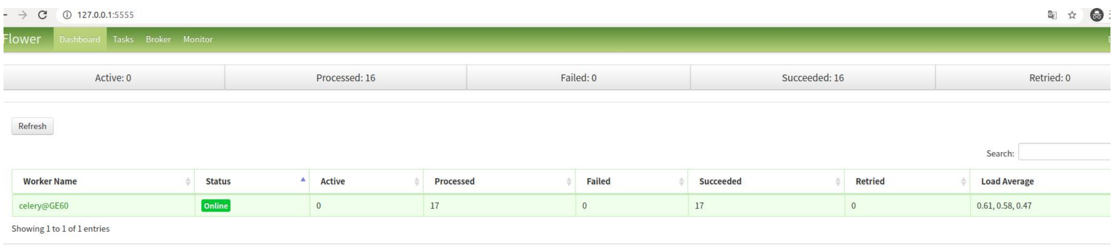
Pip install flower
flower -A djmq.celery -l info



```
^C(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ flower -A djmq.celery -l info
[I 200726 13:36:38 command:140] Visit me at http://localhost:5555
[I 200726 13:36:38 command:145] Broker: redis://127.0.0.1:6379/0
[I 200726 13:36:38 command:148] Registered tasks:
    ['celery.accumulate',
     'celery.backend_cleanup',
     'celery.chain',
     'celery.chord',
     'celery.chord_unlock',
     'celery.chunks',
     'celery.group',
     'celery.map',
     'celery.starmap',
     'djmq.celery.debug_task',
     'user.tasks.beattestfunc',
     'user.tasks.testfunc']
[I 200726 13:36:38 mixins:229] Connected to redis://127.0.0.1:6379/0
```

访问 https://127.0.0.1:5555 可视化界面

首页



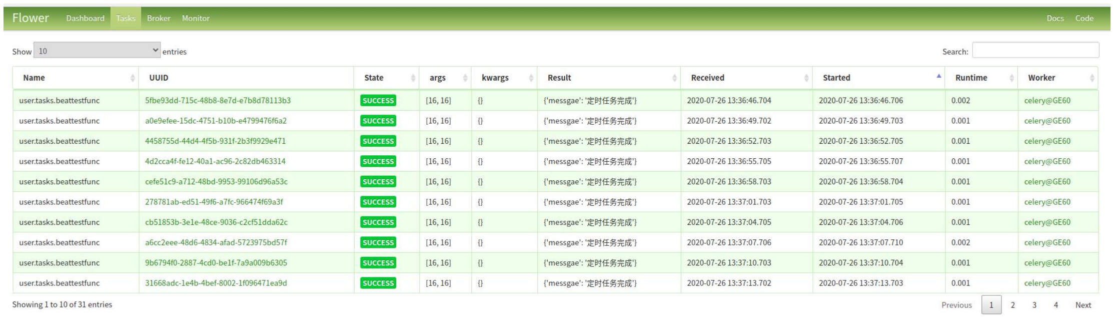
Active: 0	Processed: 16	Failed: 0	Succeeded: 16	Retried: 0

Search:

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@GE60	Online	0	17	0	17	0	0.61, 0.58, 0.47

Showing 1 to 1 of 1 entries

所有执行过的任务(成功或者失败的 ,包括返回的结果参数)



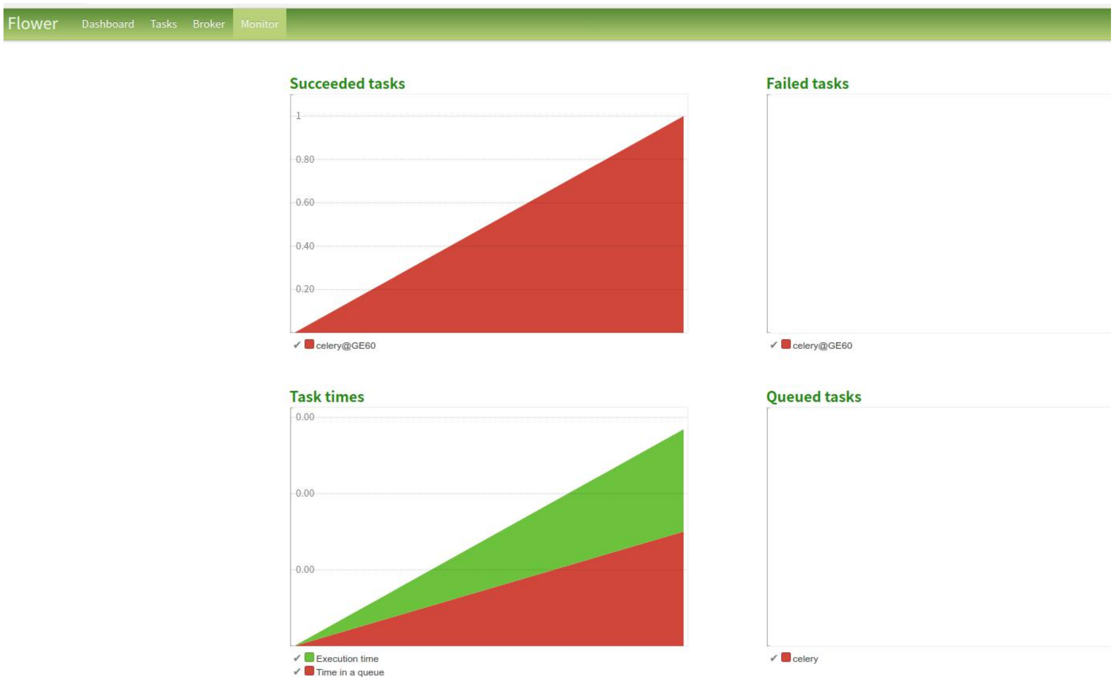
Show 10 entries

Search:

Name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
user.tasks.beattestfunc	5fbe93dd-715c-48b8-8e7d-e7b8d78113b3	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:36:46.704	2020-07-26 13:36:46.706	0.002	celery@GE60
user.tasks.beattestfunc	a0e9efee-15dc-4751-b10b-e4799476f6a2	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:36:49.702	2020-07-26 13:36:49.703	0.001	celery@GE60
user.tasks.beattestfunc	4458755d-44d4-4f5b-931f-2b3f9929e471	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:36:52.703	2020-07-26 13:36:52.705	0.001	celery@GE60
user.tasks.beattestfunc	4d2cca4f-fe12-40a1-ac96-2c82db463314	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:36:55.705	2020-07-26 13:36:55.707	0.001	celery@GE60
user.tasks.beattestfunc	cefe51c9-a712-48bd-9953-99106d96a53c	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:36:58.703	2020-07-26 13:36:58.704	0.001	celery@GE60
user.tasks.beattestfunc	278781ab-ed51-49f6-a7fc-966474f69a3f	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:37:01.703	2020-07-26 13:37:01.705	0.001	celery@GE60
user.tasks.beattestfunc	cb51853b-3e1e-48ce-9036-c2cf51dda62c	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:37:04.705	2020-07-26 13:37:04.706	0.001	celery@GE60
user.tasks.beattestfunc	a6cc2eee-48d6-4834-afad-5723975bd57f	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:37:07.706	2020-07-26 13:37:07.710	0.002	celery@GE60
user.tasks.beattestfunc	9b6794f0-2887-4cd0-be1f-7a9a009b6305	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:37:10.703	2020-07-26 13:37:10.704	0.001	celery@GE60
user.tasks.beattestfunc	31668adc-1e4b-4bef-8002-1f096471ea9d	SUCCESS	[16, 16]	{}	{'messgae': '定时任务完成'}	2020-07-26 13:37:13.702	2020-07-26 13:37:13.703	0.001	celery@GE60

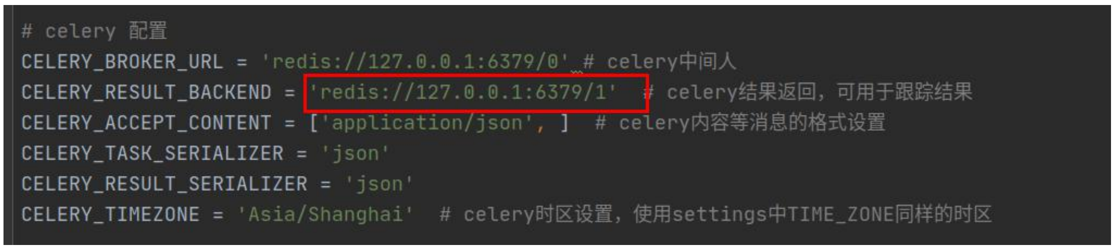
Showing 1 to 10 of 31 entries

饼图分析

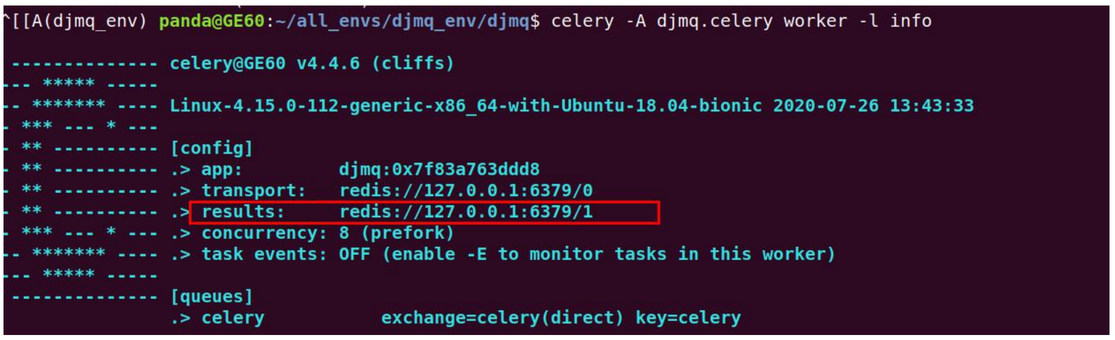


至此 celery 异步任务和定时任务以及监控状态相关功能基本介绍完毕，但是根据配置这些

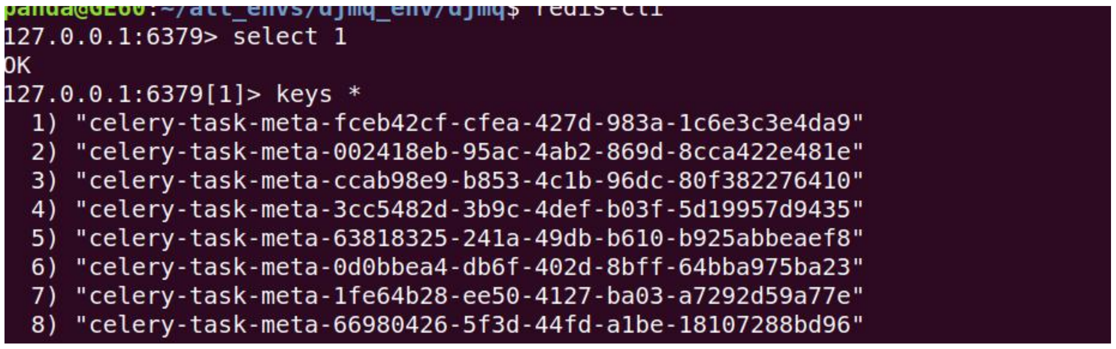
执行结果都是保存在 redis 中的



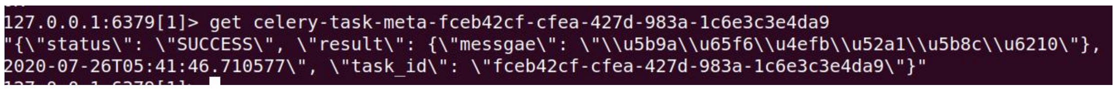
```
# celery 配置
CELERY_BROKER_URL = 'redis://127.0.0.1:6379/0' # celery中间人
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/1'  # celery结果返回，可用于跟踪结果
CELERY_ACCEPT_CONTENT = ['application/json', ]  # celery内容等消息的格式设置
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'Asia/Shanghai'  # celery时区设置，使用settings中TIME_ZONE同样的时区
```



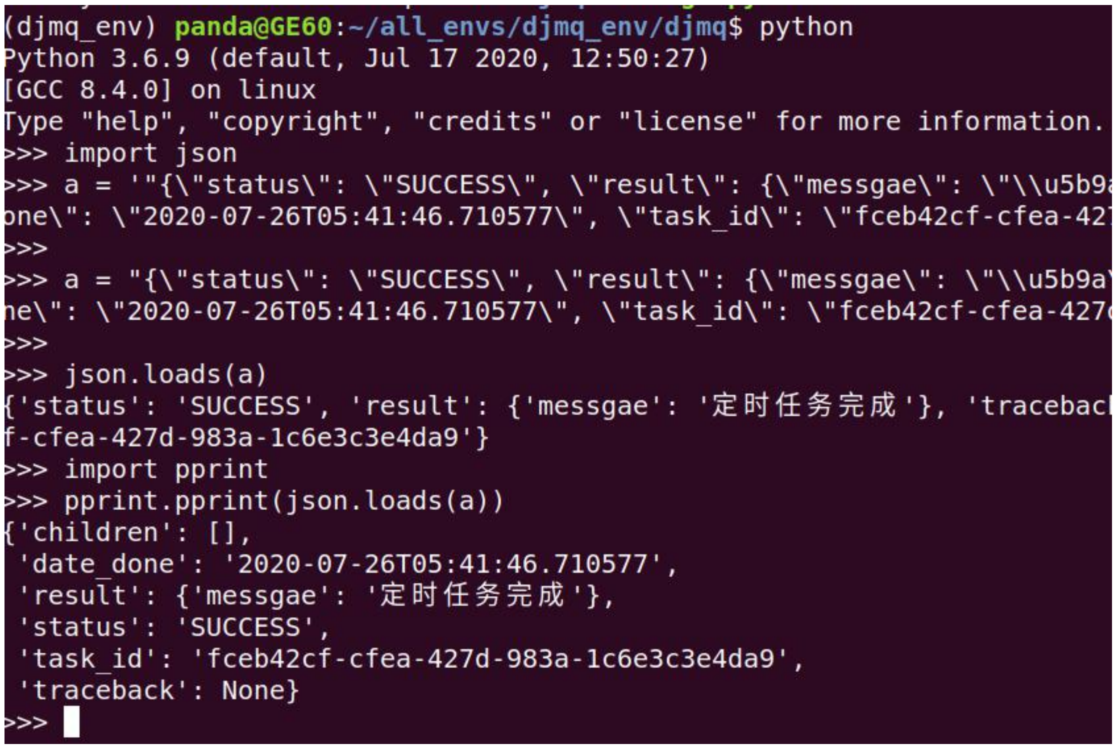
```
^[[A(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery worker -l info

 -------------- celery@GE60 v4.4.6 (cliffs)
--- ***** -----
-- ******* ---- Linux-4.15.0-112-generic-x86_64-with-Ubuntu-18.04-bionic 2020-07-26 13:43:33
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         djmq:0x7f83a763ddd8
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery
```



```
panda@GE60:~/all_envs/djmq_env/djmq$ redis-cli
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> keys *
  1) "celery-task-meta-fceb42cf-cfea-427d-983a-1c6e3c3e4da9"
  2) "celery-task-meta-002418eb-95ac-4ab2-869d-8cca422e481e"
  3) "celery-task-meta-ccab98e9-b853-4c1b-96dc-80f382276410"
  4) "celery-task-meta-3cc5482d-3b9c-4def-b03f-5d19957d9435"
  5) "celery-task-meta-63818325-241a-49db-b610-b925abbeaef8"
  6) "celery-task-meta-0d0bbea4-db6f-402d-8bff-64bba975ba23"
  7) "celery-task-meta-1fe64b28-ee50-4127-ba03-a7292d59a77e"
  8) "celery-task-meta-66980426-5f3d-44fd-a1be-18107288bd96"
```

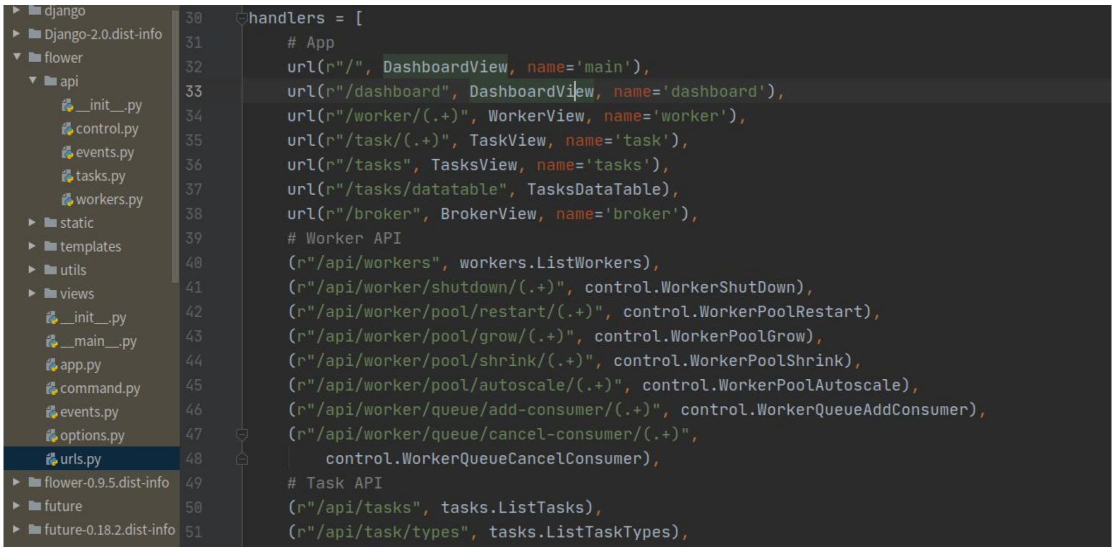


```
127.0.0.1:6379[1]> get celery-task-meta-fceb42cf-cfea-427d-983a-1c6e3c3e4da9
"{\"status\": \"SUCCESS\", \"result\": {\"messgae\": \"\\u5b9a\\u65f6\\u4efb\\u52a1\\u5b8c\\u6210\"},
2020-07-26T05:41:46.710577\", \"task_id\": \"fceb42cf-cfea-427d-983a-1c6e3c3e4da9\"}"
```

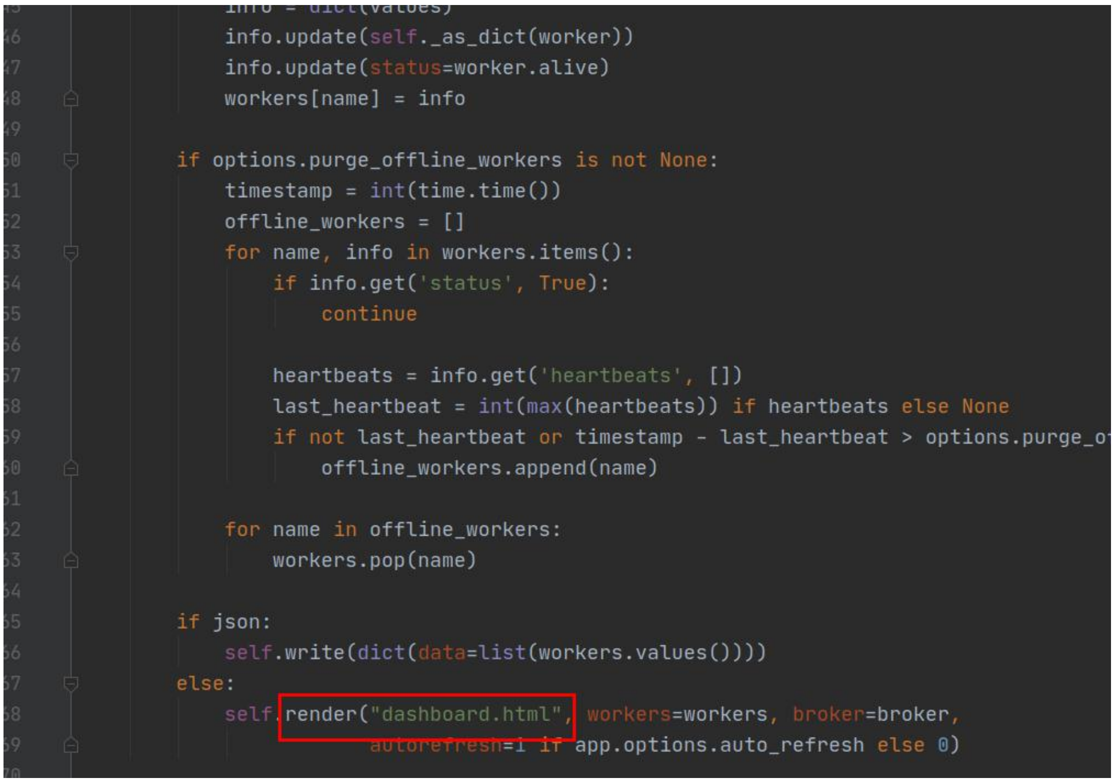


```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ python
Python 3.6.9 (default, Jul 17 2020, 12:50:27)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import json
>>> a = '"{\"status\": \"SUCCESS\", \"result\": {\"messgae\": \"\\u5b9a
one\": \"2020-07-26T05:41:46.710577\", \"task_id\": \"fceb42cf-cfea-42
>>>
>>> a = "{\"status\": \"SUCCESS\", \"result\": {\"messgae\": \"\\u5b9a
ne\": \"2020-07-26T05:41:46.710577\", \"task_id\": \"fceb42cf-cfea-427
>>>
>>> json.loads(a)
{'status': 'SUCCESS', 'result': {'messgae': '定时任务完成'}, 'tracebac
f-cfea-427d-983a-1c6e3c3e4da9'}
>>> import pprint
>>> pprint.pprint(json.loads(a))
{'children': [],
 'date_done': '2020-07-26T05:41:46.710577',
 'result': {'messgae': '定时任务完成'},
 'status': 'SUCCESS',
 'task_id': 'fceb42cf-cfea-427d-983a-1c6e3c3e4da9',
 'traceback': None}
>>>
```

数据库只有这么简单的数据，至于可视化界面上的其他参数则是 flower 动态生成的，这一点可以在 flower 的源码中看到：



```
handlers = [
    # App
    url(r"/", DashboardView, name='main'),
    url(r"/dashboard", DashboardView, name='dashboard'),
    url(r"/worker/(.+)", WorkerView, name='worker'),
    url(r"/task/(.+)", TaskView, name='task'),
    url(r"/tasks", TasksView, name='tasks'),
    url(r"/tasks/datatable", TasksDataTable),
    url(r"/broker", BrokerView, name='broker'),
    # Worker API
    (r"/api/workers", workers.ListWorkers),
    (r"/api/worker/shutdown/(.+)", control.WorkerShutDown),
    (r"/api/worker/pool/restart/(.+)", control.WorkerPoolRestart),
    (r"/api/worker/pool/grow/(.+)", control.WorkerPoolGrow),
    (r"/api/worker/pool/shrink/(.+)", control.WorkerPoolShrink),
    (r"/api/worker/pool/autoscale/(.+)", control.WorkerPoolAutoscale),
    (r"/api/worker/queue/add-consumer/(.+)", control.WorkerQueueAddConsumer),
    (r"/api/worker/queue/cancel-consumer/(.+)",
        control.WorkerQueueCancelConsumer),
    # Task API
    (r"/api/tasks", tasks.ListTasks),
    (r"/api/task/types", tasks.ListTaskTypes),
```

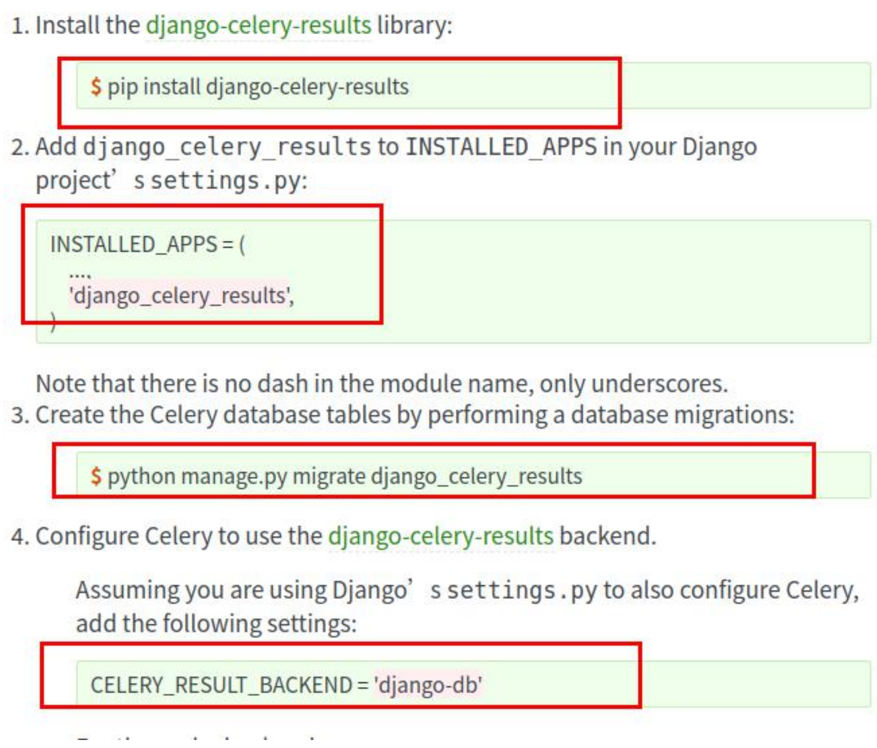


```
            info = dict(values)
            info.update(self._as_dict(worker))
            info.update(status=worker.alive)
            workers[name] = info

        if options.purge_offline_workers is not None:
            timestamp = int(time.time())
            offline_workers = []
            for name, info in workers.items():
                if info.get('status', True):
                    continue

                heartbeats = info.get('heartbeats', [])
                last_heartbeat = int(max(heartbeats)) if heartbeats else None
                if not last_heartbeat or timestamp - last_heartbeat > options.purge_o
                    offline_workers.append(name)

            for name in offline_workers:
                workers.pop(name)

        if json:
            self.write(dict(data=list(workers.values())))
        else:
            self.render("dashboard.html", workers=workers, broker=broker,
                        autorefresh=1 if app.options.auto_refresh else 0)
```

Django-celery-results

看名字就知道把 celery 的结果保存在另外的地方。文档就是 celery 的页面下面拓展部分，

https://docs.celeryproject.org/en/stable/django/first-steps-with-django.html#using-celery-with-django


1. Install the django-celery-results library:

```
$ pip install django-celery-results
```

2. Add `django_celery_results` to `INSTALLED_APPS` in your Django project’s `settings.py`:

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

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

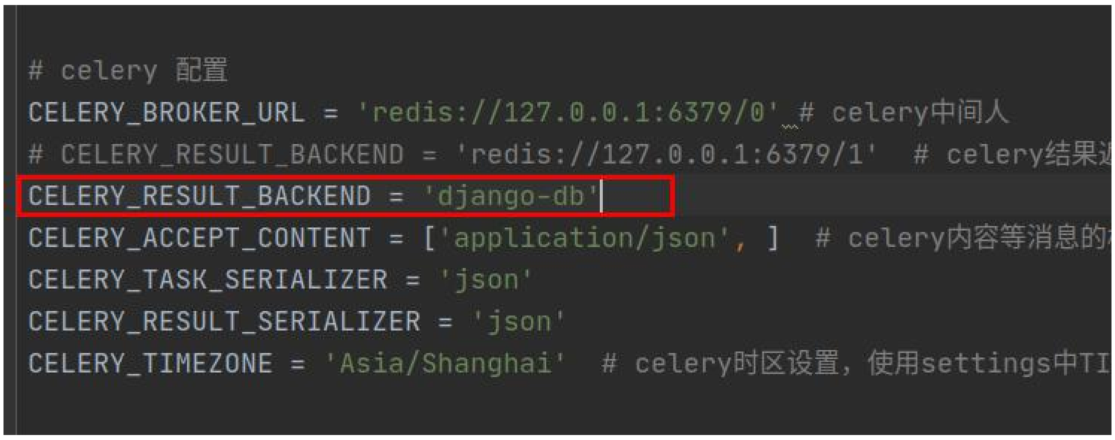
3. Create the Celery database tables by performing a database migrations:

```
$ python manage.py migrate django_celery_results
```

4. Configure Celery to use the django-celery-results backend.

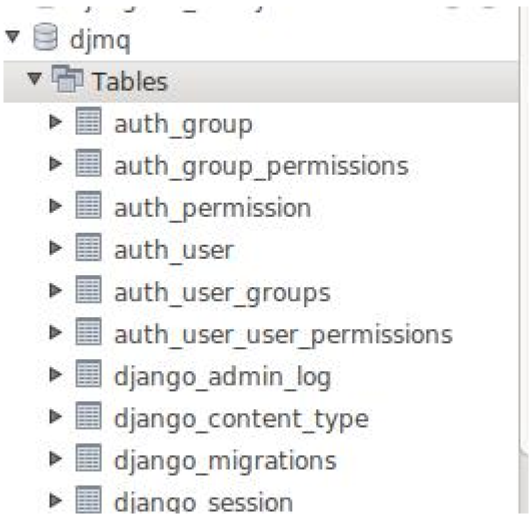
   Assuming you are using Django’s `settings.py` to also configure Celery, add the following settings:

```
CELERY_RESULT_BACKEND = 'django-db'
```



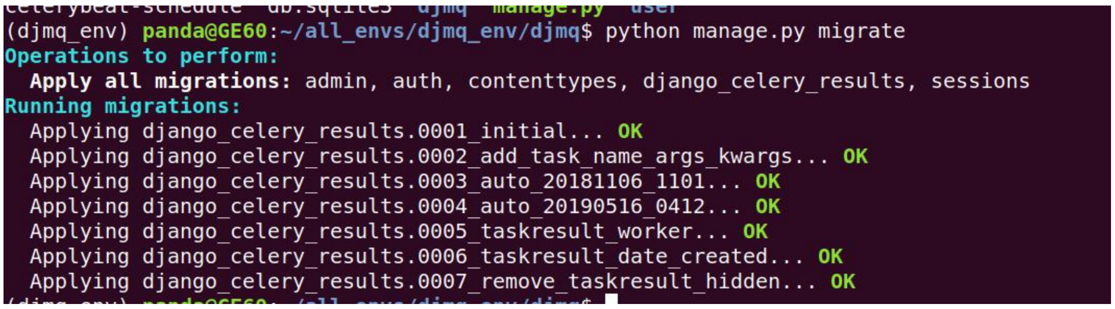
```
# celery 配置
CELERY_BROKER_URL = 'redis://127.0.0.1:6379/0' # celery中间人
# CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/1'  # celery结果
CELERY_RESULT_BACKEND = 'django-db'
CELERY_ACCEPT_CONTENT = ['application/json', ]  # celery内容等消息的
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'Asia/Shanghai'  # celery时区设置，使用settings中TI
```

在生成数据库表我们先看看之前的表的结构

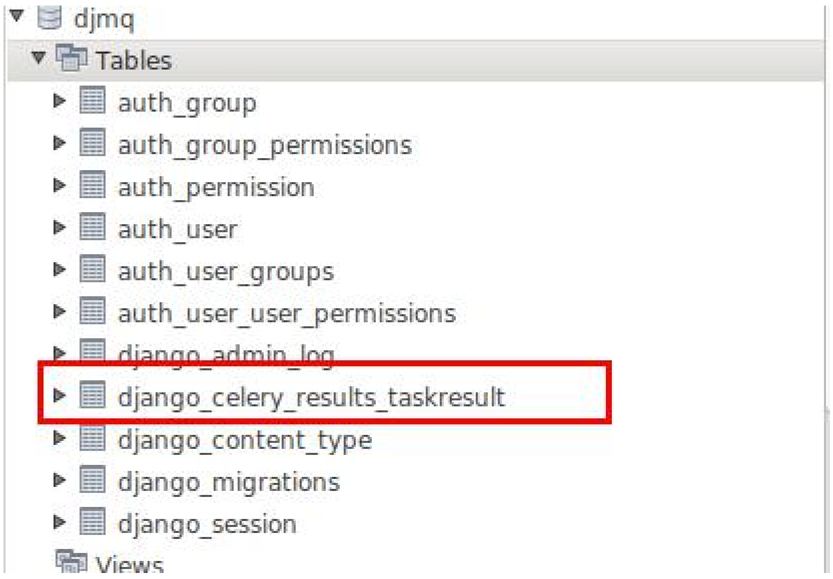


```
▼ djmq
  ▼ Tables
    ▶ auth_group
    ▶ auth_group_permissions
    ▶ auth_permission
    ▶ auth_user
    ▶ auth_user_groups
    ▶ auth_user_user_permissions
    ▶ django_admin_log
    ▶ django_content_type
    ▶ django_migrations
    ▶ django_session
```

安装 django-celery-results 迁移数据库后

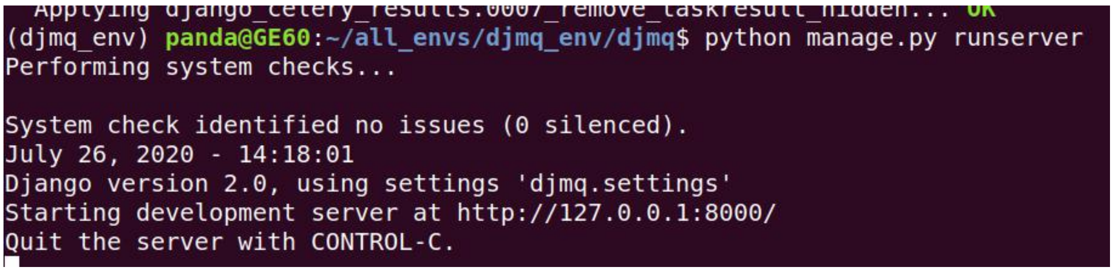


```
celerybeat-schedule  db.sqlite3  djmq  manage.py  user
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, django_celery_results, sessions
Running migrations:
  Applying django_celery_results.0001_initial... OK
  Applying django_celery_results.0002_add_task_name_args_kwargs... OK
  Applying django_celery_results.0003_auto_20181106_1101... OK
  Applying django_celery_results.0004_auto_20190516_0412... OK
  Applying django_celery_results.0005_taskresult_worker... OK
  Applying django_celery_results.0006_taskresult_date_created... OK
  Applying django_celery_results.0007_remove_taskresult_hidden... OK
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$
```

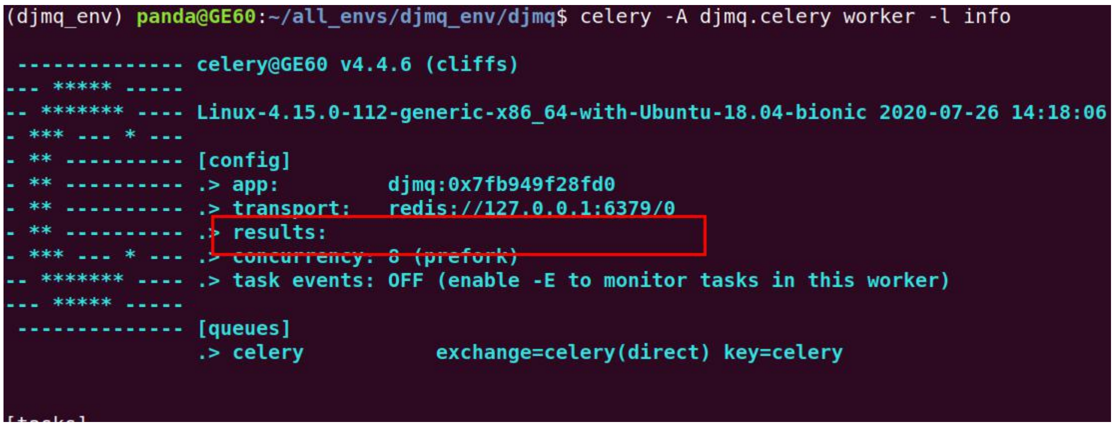


```
djmq
  Tables
    auth_group
    auth_group_permissions
    auth_permission
    auth_user
    auth_user_groups
    auth_user_user_permissions
    django_admin_log
    django_celery_results_taskresult
    django_content_type
    django_migrations
    django_session
  Views
```

重新启动项目

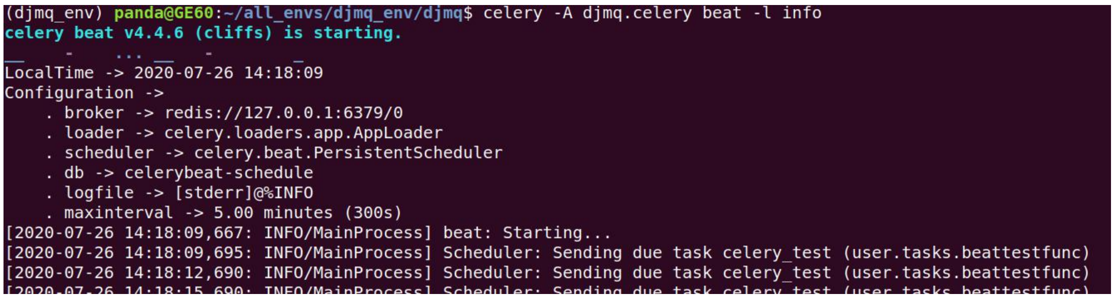


```
  Applying django_celery_results.0007_remove_taskresult_hidden... OK
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
July 26, 2020 - 14:18:01
Django version 2.0, using settings 'djmq.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

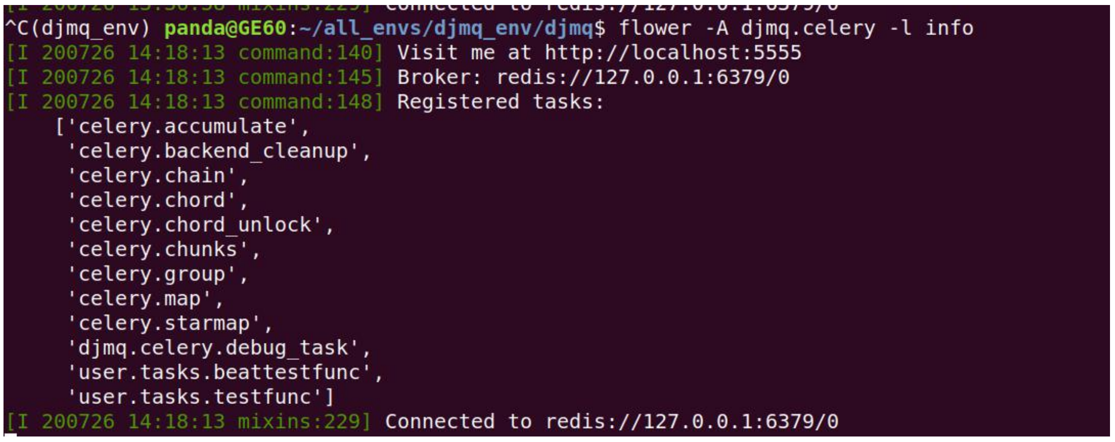


```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery worker -l info

 -------------- celery@GE60 v4.4.6 (cliffs)
--- ***** -----
-- ******* ---- Linux-4.15.0-112-generic-x86_64-with-Ubuntu-18.04-bionic 2020-07-26 14:18:06
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         djmq:0x7fb949f28fd0
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
```

这里不是 redis /1 库了

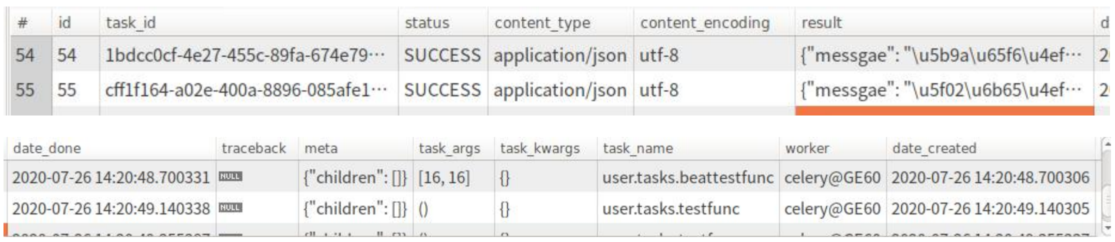


```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery beat -l info
celery beat v4.4.6 (cliffs) is starting.
__    -    ... __   -        _
LocalTime -> 2020-07-26 14:18:09
Configuration ->
    . broker -> redis://127.0.0.1:6379/0
    . loader -> celery.loaders.app.AppLoader
    . scheduler -> celery.beat.PersistentScheduler
    . db -> celerybeat-schedule
    . logfile -> [stderr]@%INFO
    . maxinterval -> 5.00 minutes (300s)
[2020-07-26 14:18:09,667: INFO/MainProcess] beat: Starting...
[2020-07-26 14:18:09,695: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
[2020-07-26 14:18:12,690: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
[2020-07-26 14:18:15,690: INFO/MainProcess] Scheduler: Sending due task celery_test (user.tasks.beattestfunc)
```



```
^C(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ flower -A djmq.celery -l info
[I 200726 14:18:13 command:140] Visit me at http://localhost:5555
[I 200726 14:18:13 command:145] Broker: redis://127.0.0.1:6379/0
[I 200726 14:18:13 command:148] Registered tasks:
    ['celery.accumulate',
     'celery.backend_cleanup',
     'celery.chain',
     'celery.chord',
     'celery.chord_unlock',
     'celery.chunks',
     'celery.group',
     'celery.map',
     'celery.starmap',
     'djmq.celery.debug_task',
     'user.tasks.beattestfunc',
     'user.tasks.testfunc']
[I 200726 14:18:13 mixins:229] Connected to redis://127.0.0.1:6379/0
```

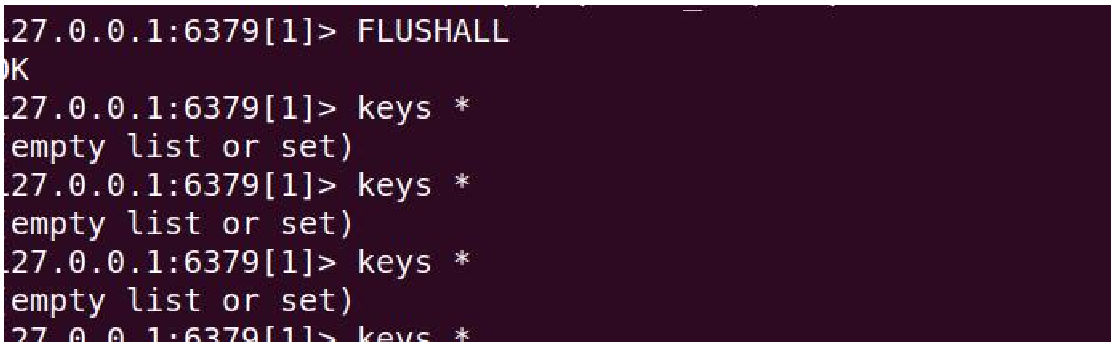
异步任务的结果和执行状态现在保存在 db 中了，如果需要可以做 api 接口拱调用



#	id	task_id	status	content_type	content_encoding	result	d
54	54	1bdcc0cf-4e27-455c-89fa-674e79⋯	SUCCESS	application/json	utf-8	{"messgae": "\u5b9a\u65f6\u4ef⋯	2
55	55	cff1f164-a02e-400a-8896-085afe1⋯	SUCCESS	application/json	utf-8	{"messgae": "\u5f02\u6b65\u4ef⋯	2

date_done	traceback	meta	task_args	task_kwargs	task_name	worker	date_created
2020-07-26 14:20:48.700331	NULL	{"children": []}	[16, 16]	{}	user.tasks.beattestfunc	celery@GE60	2020-07-26 14:20:48.700306
2020-07-26 14:20:49.140338	NULL	{"children": []}	()	{}	user.tasks.testfunc	celery@GE60	2020-07-26 14:20:49.140305

Redis /1 库中没有数据了



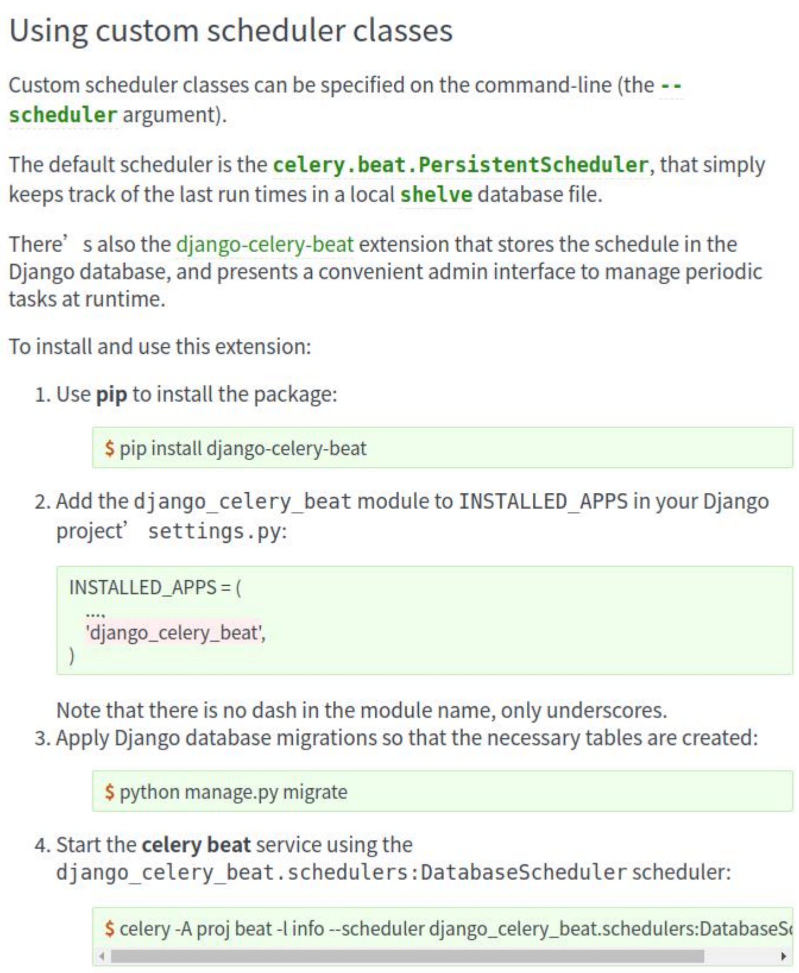
```
.27.0.0.1:6379[1]> FLUSHALL
)K
.27.0.0.1:6379[1]> keys *
(empty list or set)
.27.0.0.1:6379[1]> keys *
(empty list or set)
.27.0.0.1:6379[1]> keys *
(empty list or set)
.27.0.0.1:6379[1]> keys *
```

Django-celery-beat

Celery-beat 文档：

https://docs.celeryproject.org/en/stable/userguide/periodic-tasks.html#beat-custom-schedulers

Django-celery-beat 在下面自定义的地方



Using custom scheduler classes

Custom scheduler classes can be specified on the command-line (the `--scheduler` argument).

The default scheduler is the `celery.beat.PersistentScheduler`, that simply keeps track of the last run times in a local `shelve` database file.

There’ s also the django-celery-beat extension that stores the schedule in the Django database, and presents a convenient admin interface to manage periodic tasks at runtime.

To install and use this extension:

1. Use **pip** to install the package:

   ```
   $ pip install django-celery-beat
   ```

2. Add the `django_celery_beat` module to `INSTALLED_APPS` in your Django project’ `settings.py`:

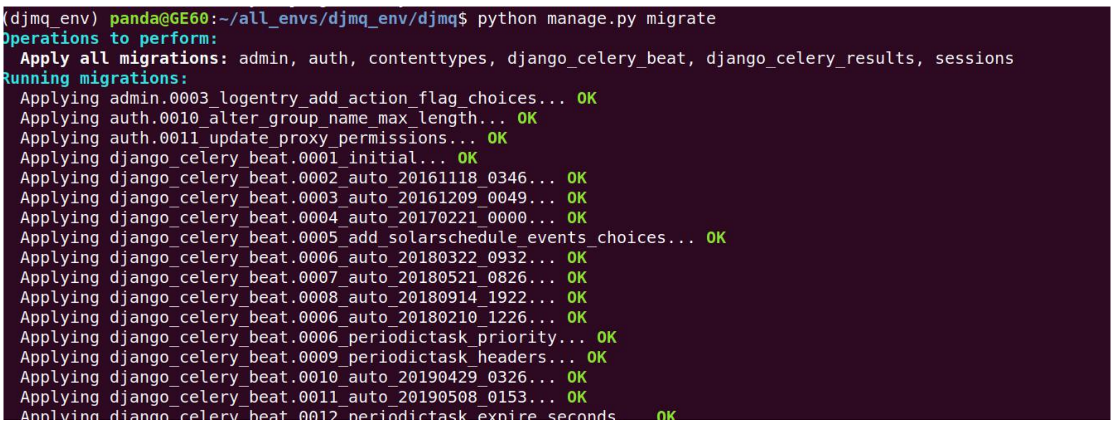
   ```
   INSTALLED_APPS = (
       ....,
       'django_celery_beat',
   )
   ```

   Note that there is no dash in the module name, only underscores.
3. Apply Django database migrations so that the necessary tables are created:

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

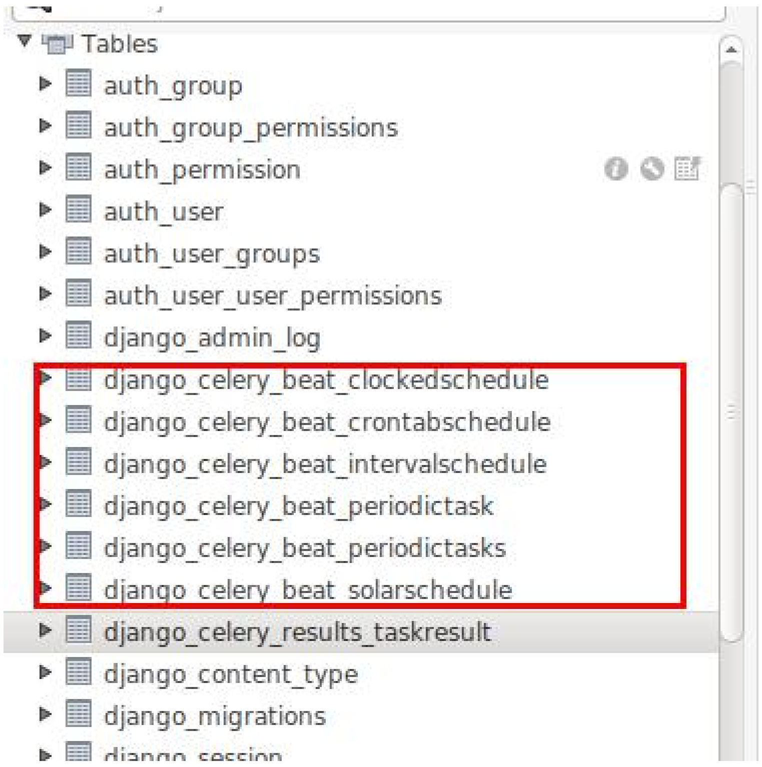
4. Start the **celery beat** service using the `django_celery_beat.schedulers:DatabaseScheduler` scheduler:

   ```
   $ celery -A proj beat -l info --scheduler django_celery_beat.schedulers:DatabaseSc
   ```



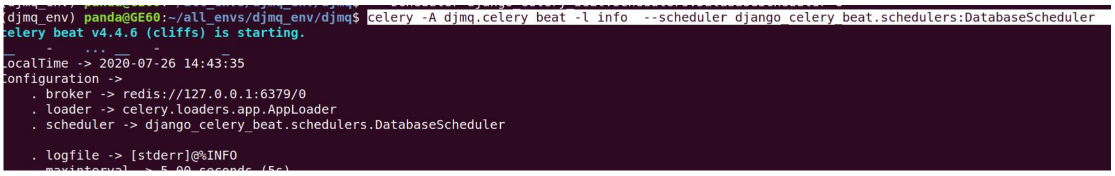
```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, django_celery_beat, django_celery_results, sessions
Running migrations:
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying django_celery_beat.0001_initial... OK
  Applying django_celery_beat.0002_auto_20161118_0346... OK
  Applying django_celery_beat.0003_auto_20161209_0049... OK
  Applying django_celery_beat.0004_auto_20170221_0000... OK
  Applying django_celery_beat.0005_add_solarschedule_events_choices... OK
  Applying django_celery_beat.0006_auto_20180322_0932... OK
  Applying django_celery_beat.0007_auto_20180521_0826... OK
  Applying django_celery_beat.0008_auto_20180914_1922... OK
  Applying django_celery_beat.0006_auto_20180210_1226... OK
  Applying django_celery_beat.0006_periodictask_priority... OK
  Applying django_celery_beat.0009_periodictask_headers... OK
  Applying django_celery_beat.0010_auto_20190429_0326... OK
  Applying django_celery_beat.0011_auto_20190508_0153... OK
  Applying django_celery_beat.0012_periodictask_expire_seconds... OK
```

有必要新增这么多表么。。。。。。



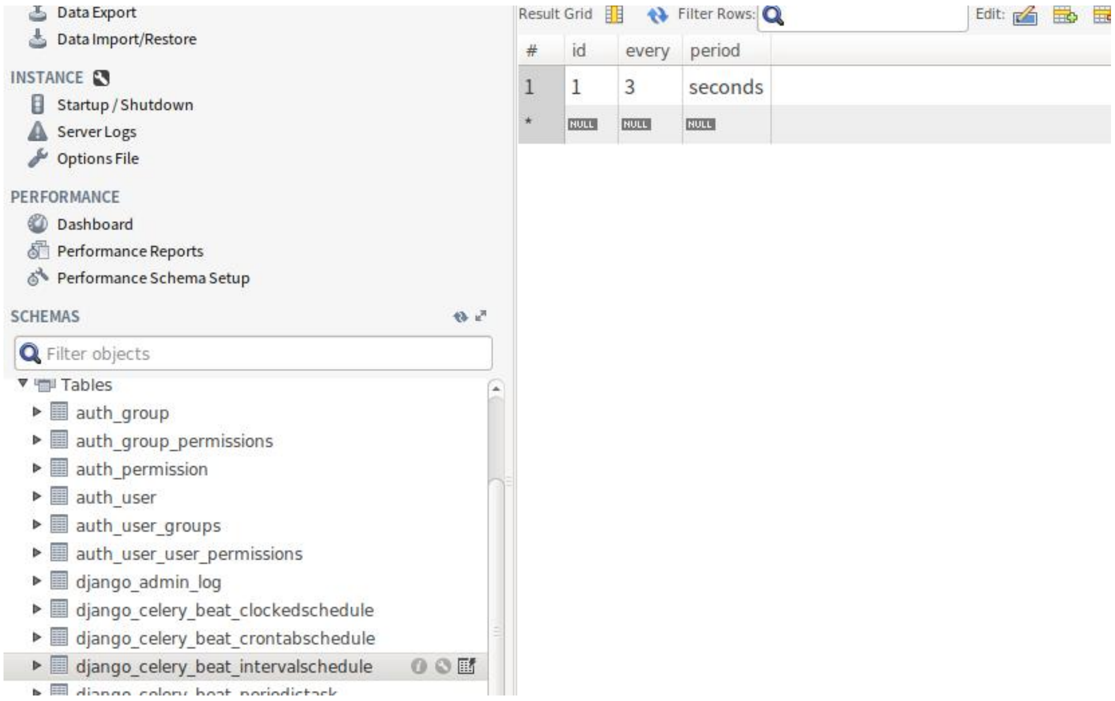
```
Tables
  auth_group
  auth_group_permissions
  auth_permission
  auth_user
  auth_user_groups
  auth_user_user_permissions
  django_admin_log
  django_celery_beat_clockedschedule
  django_celery_beat_crontabschedule
  django_celery_beat_intervalschedule
  django_celery_beat_periodictask
  django_celery_beat_periodictasks
  django_celery_beat_solarschedule
  django_celery_results_taskresult
  django_content_type
  django_migrations
  django_session
```

Ceery beat 启动的命令需要和之前不一样，需要指定 beat 的 backend 为 django db

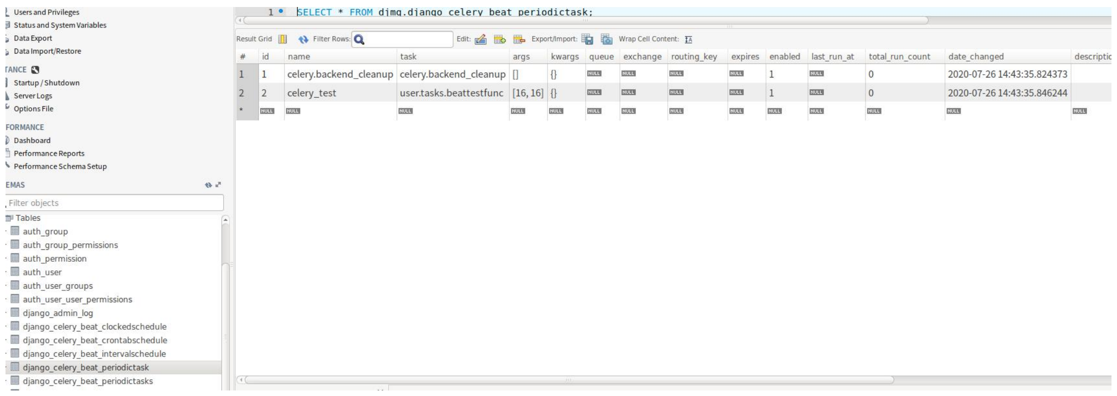


```
(djmq_env) panda@GE60:~/all_envs/djmq_env/djmq$ celery -A djmq.celery beat -l info  --scheduler django_celery_beat.schedulers:DatabaseScheduler
celery beat v4.4.6 (cliffs) is starting.
__    -    ... __   -        _
LocalTime -> 2020-07-26 14:43:35
Configuration ->
    . broker -> redis://127.0.0.1:6379/0
    . loader -> celery.loaders.app.AppLoader
    . scheduler -> django_celery_beat.schedulers.DatabaseScheduler

    . logfile -> [stderr]@%INFO
    . maxinterval -> 5.00 seconds (5s)
```

多了很多表有些还没用到，所以也体现不出来。。。



#	id	every	period
1	1	3	seconds
*	NULL	NULL	NULL



```
SELECT * FROM djmq.django_celery_beat_periodictask;
```

#	id	name	task	args	kwargs	queue	exchange	routing_key	expires	enabled	last_run_at	total_run_count	date_changed	descriptic
1	1	celery.backend_cleanup	celery.backend_cleanup	[]	{}	NULL	NULL	NULL	NULL	1	NULL	0	2020-07-26 14:43:35.824373	
2	2	celery_test	user.tasks.beattestfunc	[16, 16]	{}	NULL	NULL	NULL	NULL	1	NULL	0	2020-07-26 14:43:35.846244	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

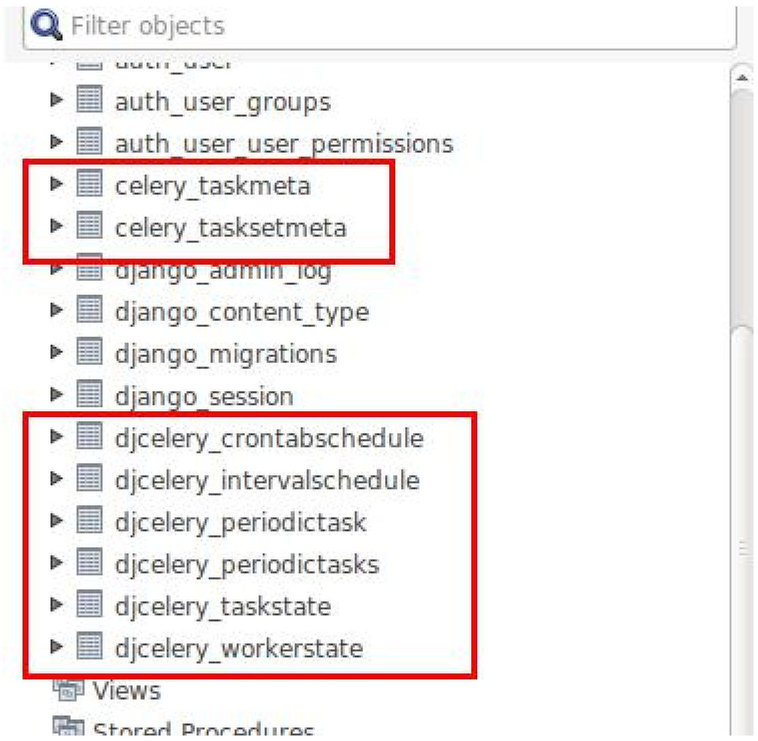
为了实现 worker 的结果 db 化和 beat 的结果 db 化一共新增了 7 个表。。

Django-celery

Django-celety 就是把 django-celery-results 和 django-celery-beat 集成了

配置很简单，迁移数据库后我们看看有什么新增的表

新增 8 个表。。。



Filter objects

auth_user_groups
auth_user_user_permissions
celery_taskmeta
celery_tasksetmeta
django_admin_log
django_content_type
django_migrations
django_session
djcelery_crontabschedule
djcelery_intervalschedule
djcelery_periodictask
djcelery_periodictasks
djcelery_taskstate
djcelery_workerstate
Views
Stored Procedures

表的大致结构是一样的。。。。看个人需求把，我还是比较相信官方提供的工具，虽然说安装的包是多了点。。。。