1、Celery介绍和基本使用

```
Celery 是一个 基于python开发的分布式异步消息任务队列，通过它可以轻松的实现任务的异步处理， 如果你的业务场景中需要用到异步任务，就可以考虑使用celery。
```

```
Celery是一个功能完备即插即用的任务队列。它使得我们不需要考虑复杂的问题，使用非常简单。

celery适用异步处理问题，当发送邮件、或者文件上传，图像处理等等一些比较耗时的操作，我们可将其异步执行，这样用户不需要等待很久，提高用户体验。
```

```
celery的特点是：
    简单，易于使用和维护，有丰富的文档。
    高效，单个celery进程每分钟可以处理数百万个任务。 灵活，celery中几乎每个部分都可以自定义扩展。 c
    celery非常易于集成到一些web开发框架中.
```

```
Celery 安装

pip install -U Celery

# -U就是 --upgrade，意思是如果已安装就升级到最新版
```

2.任务队列

```
任务队列是一种跨线程、跨机器工作的一种机制.

任务队列中包含称作任务的工作单元。
有专门的工作进程持续不断的监视任务队列，并从中获得新的任务并处理.

celery通过消息进行通信，通常使用一个叫Broker(中间人)来协client(任务的发出者)
```

```
和worker(任务的处理者).

clients发出消息到队列中，broker将队列中的信息派发给worker来处理。

一个celery系统可以包含很多的worker和broker，可增强横向扩展性和高可用性能。
```

image

3.borker

```
Celery需要一种解决消息的发送和接受的方式，我们把这种用来存储消息的的中间装置叫做message broker,也可叫做消息中间人。
```

```
Celery的默认broker是RabbitMQ, 仅需配置一行就可以

broker_url = 'amqp://guest:guest@localhost:5672//'

rabbitMQ 没装的话请装一下[安装可自行百度]

本次我们并不会使用它。因为其适应系统兼容性部署比较麻烦。
```

```
我们采用Redis做broker也可以,需要安装redis组件

# 中文手册：https://www.celerycn.io/ru-men/zhong-jian-ren-brokers/shi-yong-redis

# 对于现阶段及普及实用性来说，redis更好些
pip install -U "celery[redis]"

# 配置
# 配置消息中间人地址：
app.conf.broker_url = 'redis://localhost:6379/0'

如果想获取每个任务的执行结果，还需要配置一下把任务结果存在哪
# 注解：保存任务执行返回结果保存到Redis
app.conf.result_backend = 'redis://localhost:6379/1'
```

4.创建应用

```
创建一个celery application 用来定义你的任务列表。

创建一个任务文件 tasks.py
```

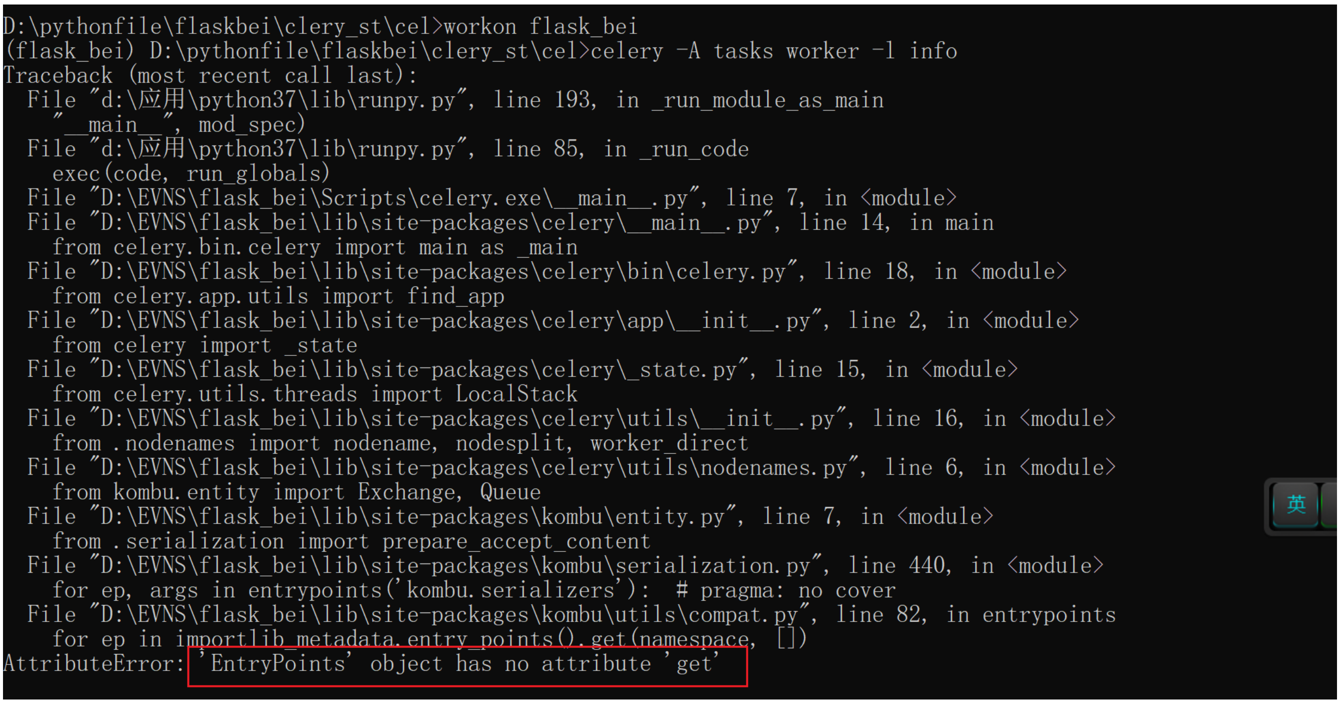
```
# -*- encoding: utf-8 -*-
from celery import Celery



# Celery第一个参数是给其设定一个名字， 第二参数我们设定一个中间人broker, 在这里我们使用Redis作为中间人。
app = Celery('test', broker='redis://127.0.0.1:6379/0')



# 任务函数，通过加上装饰器app.task, 将其注册到broker的队列中。
@app.task
def print_task():
    print("my celery 异步任务")
```

```
现在我们在创建工作者worker， 等待处理队列中的任务.

进入到 tasks.py目录下

    celery -A tasks worker -l info  # 不建议windows下使用

注意：如果是在windows下使用celery3.1.17以上版本可能会报错请使用下面命令开启worker：
    celery -A tasks worker -l info --pool=solo
celery对windows支持不太好。

# 池启动 pool参数：
https://mtank.gitee.io/2021/09/08/celery%E6%89%A7%E8%A1%8C%E6%B1%A0/
# solo pool运行于worker process（进程）内
```

```
不过目前celery5.0后版本适配依赖importlib-metadata5.0导致的问题
```

importlib-metadata-5.0.0，它删除了不推荐使用的端点
此时就属于兼容依赖问题

问题如下：

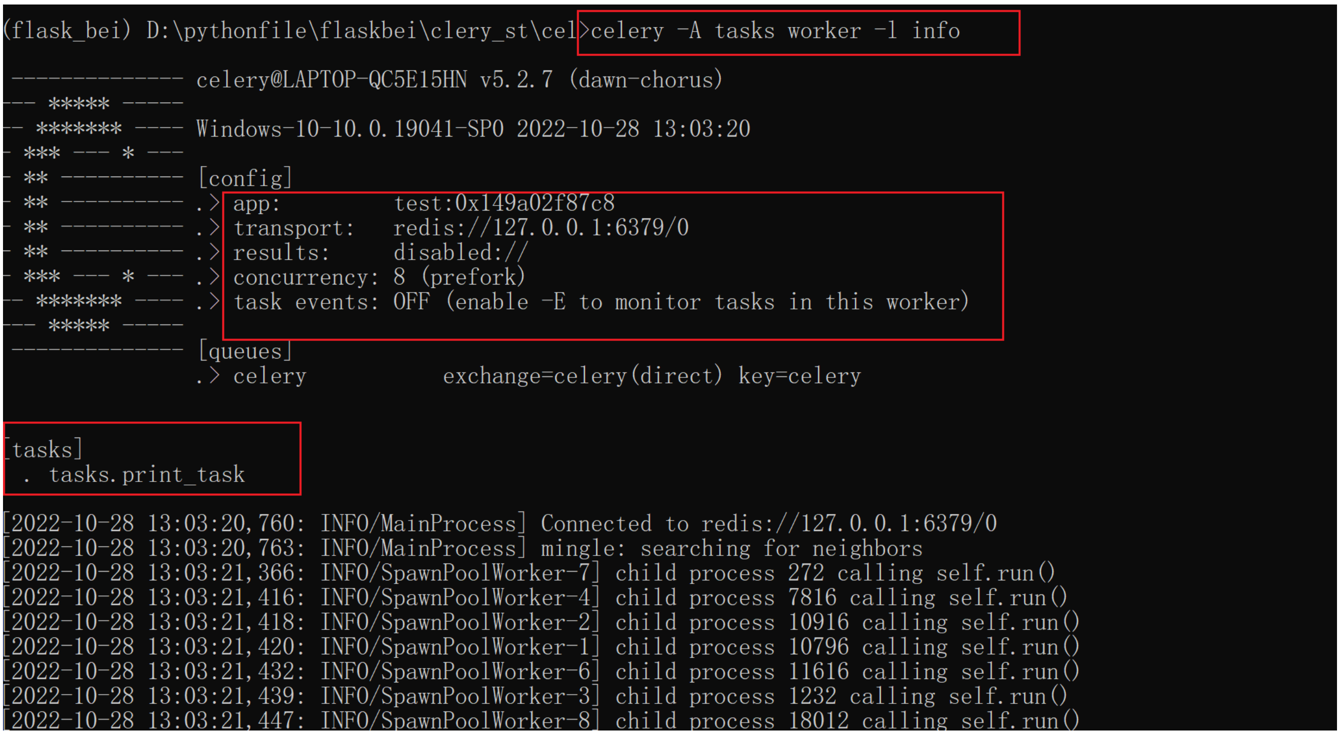


```
D:\pythonfile\flaskbei\clery_st\cel>workon flask_bei
(flask_bei) D:\pythonfile\flaskbei\clery_st\cel>celery -A tasks worker -l info
Traceback (most recent call last):
  File "d:\应用\python37\lib\runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "d:\应用\python37\lib\runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "D:\EVNS\flask_bei\Scripts\celery.exe\__main__.py", line 7, in <module>
  File "D:\EVNS\flask_bei\lib\site-packages\celery\__main__.py", line 14, in main
    from celery.bin.celery import main as _main
  File "D:\EVNS\flask_bei\lib\site-packages\celery\bin\celery.py", line 18, in <module>
    from celery.app.utils import find_app
  File "D:\EVNS\flask_bei\lib\site-packages\celery\app\__init__.py", line 2, in <module>
    from celery import _state
  File "D:\EVNS\flask_bei\lib\site-packages\celery\_state.py", line 15, in <module>
    from celery.utils.threads import LocalStack
  File "D:\EVNS\flask_bei\lib\site-packages\celery\utils\__init__.py", line 16, in <module>
    from .nodenames import nodename, nodesplit, worker_direct
  File "D:\EVNS\flask_bei\lib\site-packages\celery\utils\nodenames.py", line 6, in <module>
    from kombu.entity import Exchange, Queue
  File "D:\EVNS\flask_bei\lib\site-packages\kombu\entity.py", line 7, in <module>
    from .serialization import prepare_accept_content
  File "D:\EVNS\flask_bei\lib\site-packages\kombu\serialization.py", line 440, in <module>
    for ep, args in entrypoints('kombu.serializers'):  # pragma: no cover
  File "D:\EVNS\flask_bei\lib\site-packages\kombu\utils\compat.py", line 82, in entrypoints
    for ep in importlib_metadata.entry_points().get(namespace, [])
AttributeError: 'EntryPoints' object has no attribute 'get'
```

解决方案：降低importlib-metadata版本

pip install importlib-metadata==4.13.0

再次启动问题就解决了！！



```
(flask_bei) D:\pythonfile\flaskbei\clery_st\cel>celery -A tasks worker -l info

 -------------- celery@LAPTOP-QC5E15HN v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 13:03:20
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         test:0x149a02f87c8
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.print_task

[2022-10-28 13:03:20,760: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 13:03:20,763: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 13:03:21,366: INFO/SpawnPoolWorker-7] child process 272 calling self.run()
[2022-10-28 13:03:21,416: INFO/SpawnPoolWorker-4] child process 7816 calling self.run()
[2022-10-28 13:03:21,418: INFO/SpawnPoolWorker-2] child process 10916 calling self.run()
[2022-10-28 13:03:21,420: INFO/SpawnPoolWorker-1] child process 10796 calling self.run()
[2022-10-28 13:03:21,432: INFO/SpawnPoolWorker-6] child process 11616 calling self.run()
[2022-10-28 13:03:21,439: INFO/SpawnPoolWorker-3] child process 1232 calling self.run()
[2022-10-28 13:03:21,447: INFO/SpawnPoolWorker-8] child process 18012 calling self.run()
```

5.调用任务

任务加入到broker队列中。如何将任务函数加入到队列中，可使用delay()方法。

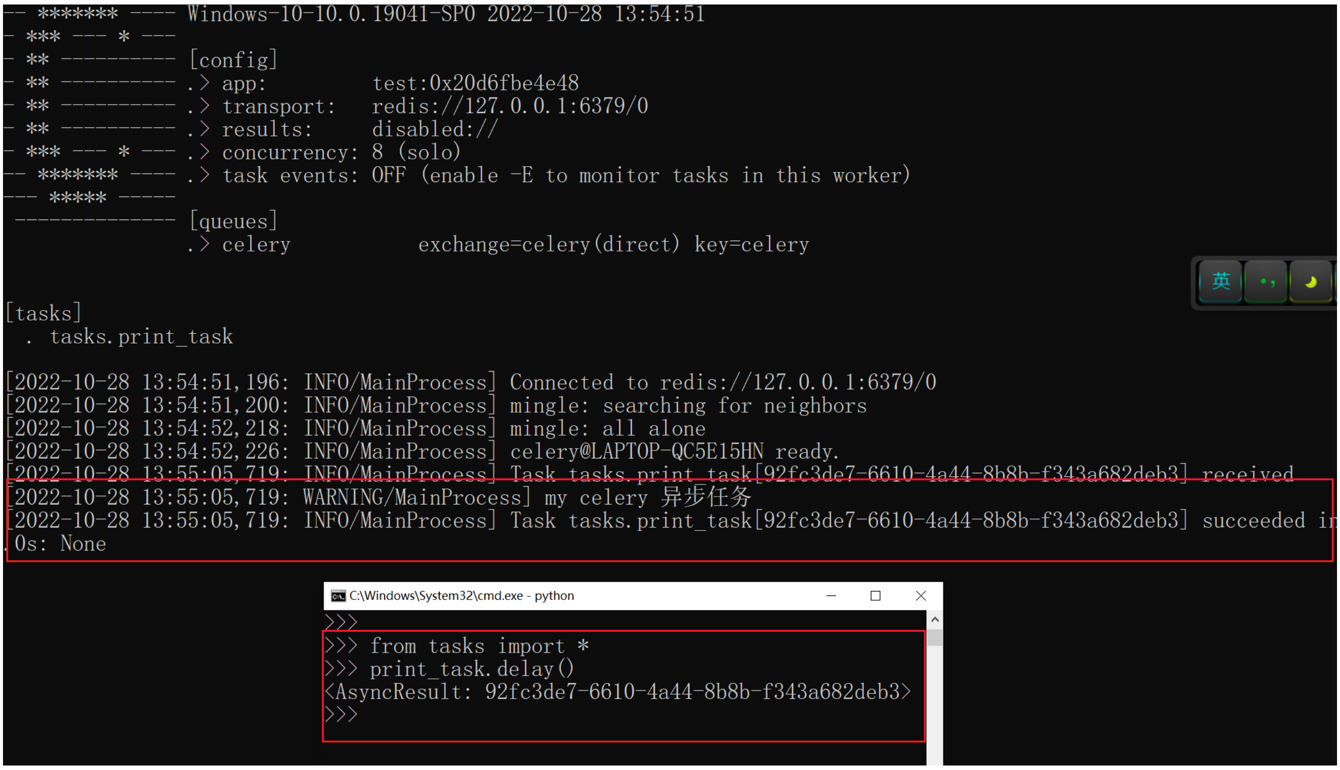
```
from tasks import *
```

```
print_task.delay()
```

或者：

```
print_task.apply_async()
```

注意测试时启动方式：celery -A tasks worker -l info --pool=solo



```
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 13:54:51
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         test:0x20d6fbe4e48
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.print_task

[2022-10-28 13:54:51,196: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 13:54:51,200: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 13:54:52,218: INFO/MainProcess] mingle: all alone
[2022-10-28 13:54:52,226: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
[2022-10-28 13:55:05,719: INFO/MainProcess] Task tasks.print_task[92fc3de7-6610-4a44-8b8b-f343a682deb3] received
[2022-10-28 13:55:05,719: WARNING/MainProcess] my celery 异步任务
[2022-10-28 13:55:05,719: INFO/MainProcess] Task tasks.print_task[92fc3de7-6610-4a44-8b8b-f343a682deb3] succeeded in
.0s: None
```

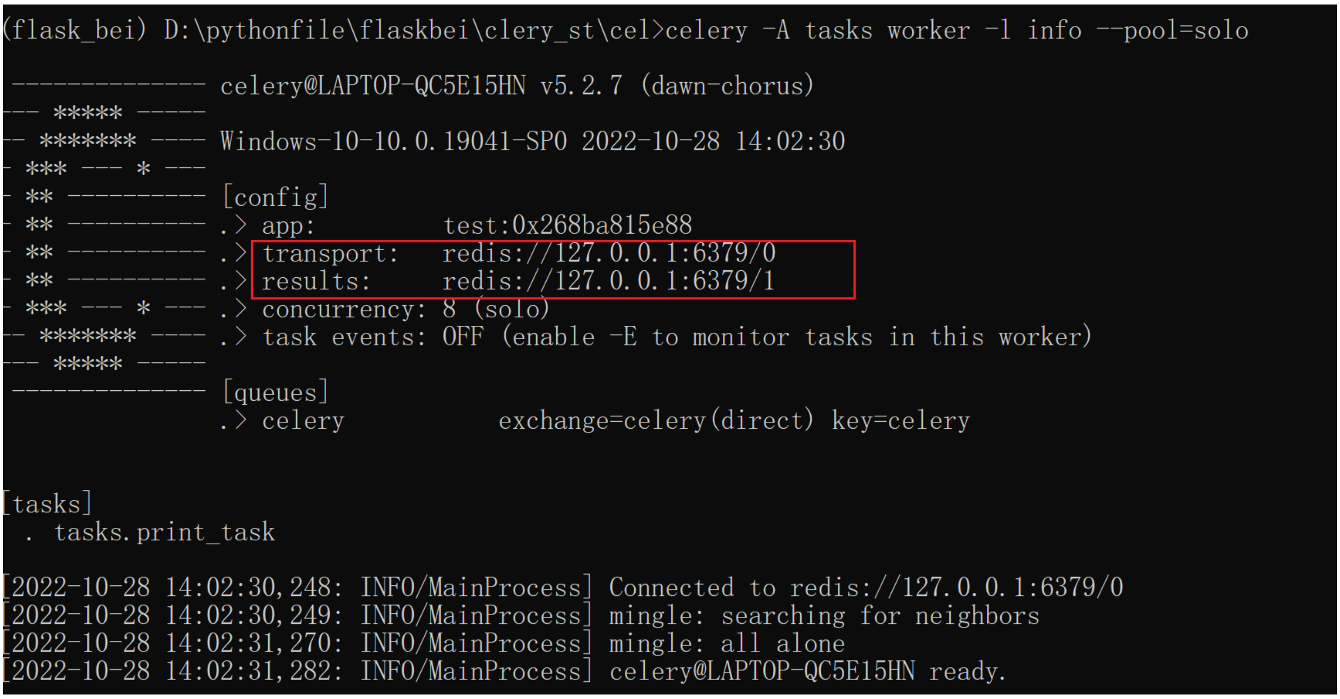
```
>>>
>>> from tasks import *
>>> print_task.delay()
<AsyncResult: 92fc3de7-6610-4a44-8b8b-f343a682deb3>
>>>
```

调用任务函数后，在worker的控制台，有一个任务被执行，返回一个AsyncResult对象，这个对象可以用来检查任务的状态或者获得任务的返回值。

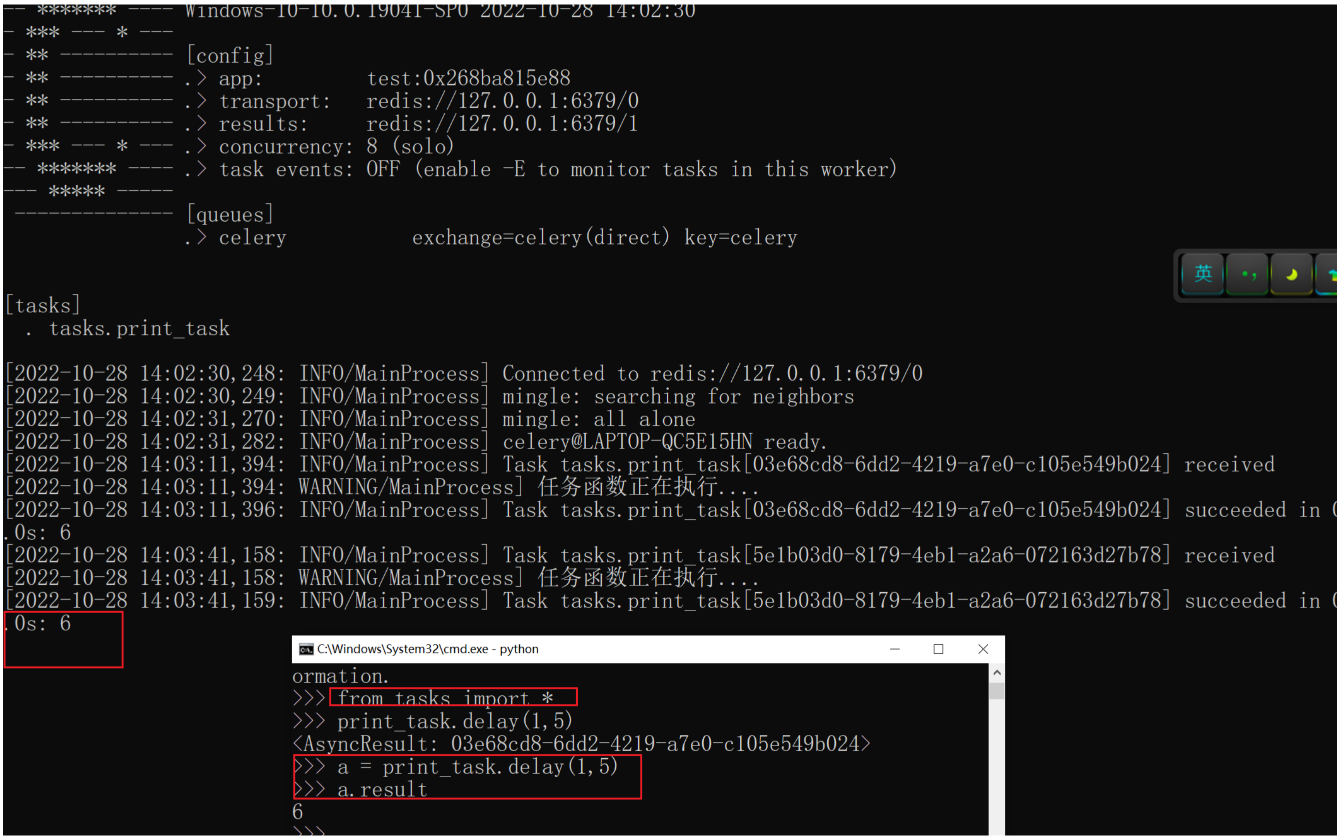
6.保存结果

如果我们想跟踪任务的状态，Celery需要将结果保存到某个地方，这里我们同样保存在redis中。

```
# -*- encoding: utf-8 -*-
from celery import Celery


# Celery第一个参数是给其设定一个名字， 第二参数我们设定一个中间人broker, 在这里我们使用Redis作为中间人。
# backend 指定保存结果,指定保存在redis中。
app = Celery('test', broker='redis://127.0.0.1:6379/0',
             backend='redis://127.0.0.1:6379/1'
             )

# 任务函数，通过加上装饰器app.task, 将其注册到broker的队列中。
# 任务函数有返回值，并且增加两个参数。
@app.task
def print_task(a, b):
    print("任务函数正在执行....")
    return a + b
```



```
(flask_bei) D:\pythonfile\flaskbei\clery_st\cel>celery -A tasks worker -l info --pool=solo

 -------------- celery@LAPTOP-QC5E15HN v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 14:02:30
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         test:0x268ba815e88
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.print_task

[2022-10-28 14:02:30,248: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 14:02:30,249: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 14:02:31,270: INFO/MainProcess] mingle: all alone
[2022-10-28 14:02:31,282: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
```



```
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 14:02:30
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         test:0x268ba815e88
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.print_task

[2022-10-28 14:02:30,248: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 14:02:30,249: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 14:02:31,270: INFO/MainProcess] mingle: all alone
[2022-10-28 14:02:31,282: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
[2022-10-28 14:03:11,394: INFO/MainProcess] Task tasks.print_task[03e68cd8-6dd2-4219-a7e0-c105e549b024] received
[2022-10-28 14:03:11,394: WARNING/MainProcess] 任务函数正在执行....
[2022-10-28 14:03:11,396: INFO/MainProcess] Task tasks.print_task[03e68cd8-6dd2-4219-a7e0-c105e549b024] succeeded in
.0s: 6
[2022-10-28 14:03:41,158: INFO/MainProcess] Task tasks.print_task[5e1b03d0-8179-4eb1-a2a6-072163d27b78] received
[2022-10-28 14:03:41,158: WARNING/MainProcess] 任务函数正在执行....
[2022-10-28 14:03:41,159: INFO/MainProcess] Task tasks.print_task[5e1b03d0-8179-4eb1-a2a6-072163d27b78] succeeded in
.0s: 6
```

```
ormation.
>>> from tasks import *
>>> print_task.delay(1,5)
<AsyncResult: 03e68cd8-6dd2-4219-a7e0-c105e549b024>
>>> a = print_task.delay(1,5)
>>> a.result
6
>>>
```

AsyncResult对象属性方法

```
https://docs.celeryq.dev/en/latest/reference/celery.result.html#module-celery.result
```

```
r = task.apply_async()
r.ready()     # 查看任务状态，返回布尔值， 任务执行完成，返回 True，否则返回 False.
r.wait()      # 会阻塞等待任务完成，返回任务执行结果，很少使用；
r.get(timeout=1)       # 获取任务执行结果，可以设置等待时间，如果超时但任务未完成返回None；
r.result      # 任务执行结果，未完成返回None；
r.state       # PENDING, START, SUCCESS，任务当前的状态
r.status      # PENDING, START, SUCCESS，任务当前的状态
r.successful  # 任务成功返回true
r.traceback  # 如果任务抛出了一个异常，可以获取原始的回溯信息
```

方法	说明

方法	说明
result	查看任务函数返回的结果
state/status	PENDING 任务正在等待执行。 STARTED 任务已经开始。 RETRY 任务将被重试，可能是因为失败。FAILURE 该任务引发了一个例外，或者超过了重试限制。该result属性包含任务引发的异常。 SUCCESS 执行成功
failed	执行成功返回 flask，执行失败返回 True
forget	删除这个任务的结果
get	等待任务准备就绪，然后返回结果，等待任务中的任务可能会导致死锁。参数：timeout等待多久超时
info	任务执行完成后，将包含返回值。如果任务引发异常，则这将是异常实例。
ready	True如果任务已执行则返回，如果任务仍在运行，挂起或正在等待重试，则False返回。

7.celery配置

```
Celery 配置简单，Celery有很多配置选项可以由开发人员配置，但是默认的配置都可以
满足大部分应用场景了。配置信息可以直接在app中设置，或者通过专有的配置模块来配
置。
```

```
from celery import Celery

app = Celery('test')
# 增加celery配置
app.conf.update(
    result_backend='redis://192.168.20.71:6379/2',
    broker_url='redis://192.168.20.71:6379/1',
)
```

```
通过配置文件配置：
    对于比较大的项目，我们建议配置信息作为一个单独的模块。

    我们可以通过调用app的函数来告诉Celery使用我们的配置模块。

    我们将Celery配置放在celeryconfig.py的模块中，这个模块名自定义。但需要确保
能在tasks.py中导入。
```

```
# 下面我们在tasks.py模块 同级目录下创建配置模块celeryconfig.py:

broker_url = 'redis://192.168.20.71:6379/0'
result_backend = 'redis://192.168.20.71:6379/1'
```

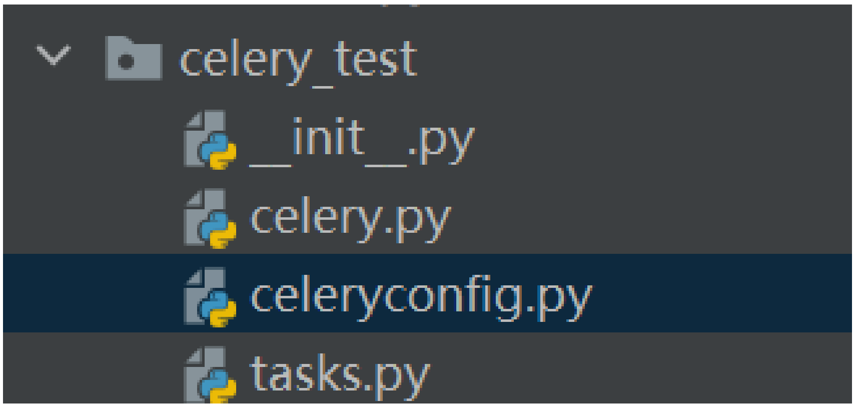
```
注意：celery 4.0以上的版本引入了小写配置，但是还会兼容大写配置。如果使用旧版
本，会识别不了小写配置，

https://docs.celeryq.dev/en/master/userguide/configuration.html#database-
backend-settings
```

```
# 修改tasks.py模块:
# -*- encoding: utf-8 -*-
from celery import Celery
```

```
import celeryconfig

app = Celery('test')

# 加载celery配置文件中的配置
app.config_from_object(celeryconfig)



# 任务函数，通过加上装饰器app.task，将其注册到broker的队列中。
# 任务函数有返回值，并且增加两个参数。
@app.task
def print_task(a, b):
    print("任务函数正在执行....")
    return a + b
```

8.在项目中使用celery

创建一个名为 celery_test目录作为celery文件目录。



celery_main.py 内容：

```
# -*- encoding: utf-8 -*-
from celery import Celery
from . import celeryconfig

# 创建celery应用
app = Celery('u_test')
# 加载celery配置文件中的配置
app.config_from_object(celeryconfig)
# 自动搜索任务，指定目录名
app.autodiscover_tasks(['celery_test'])
```

celeryconfig.py 内容：

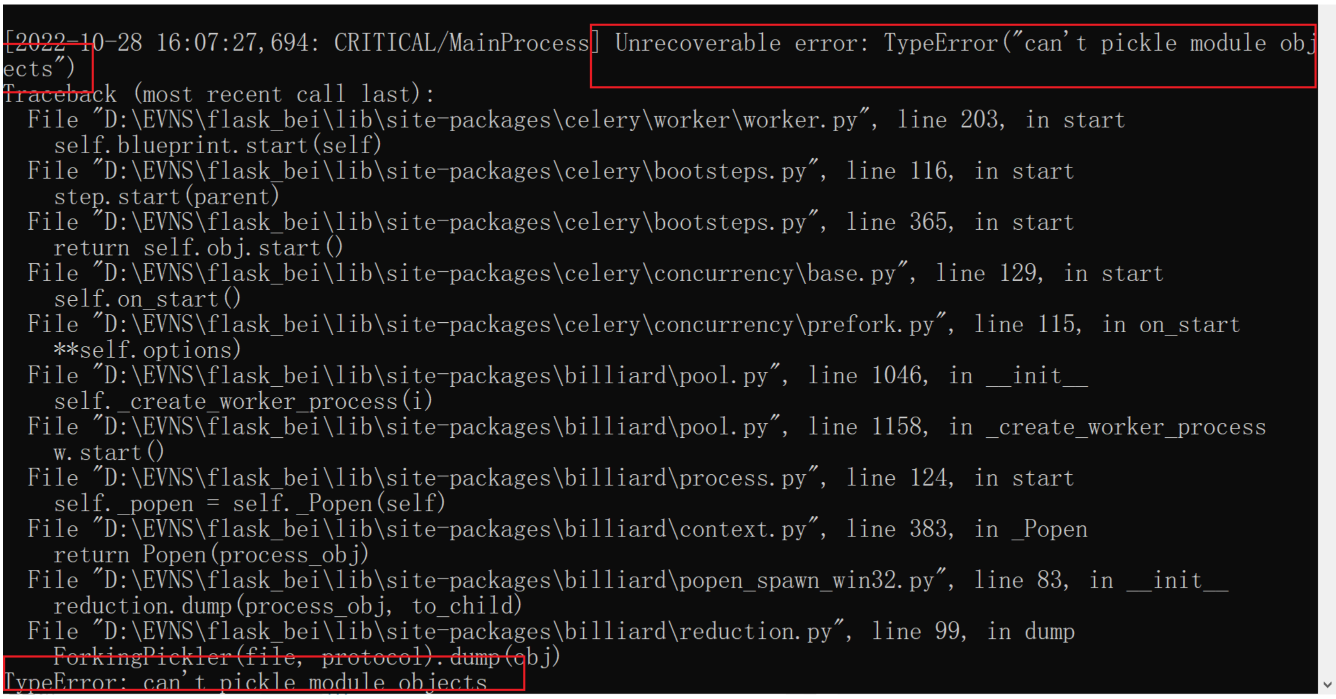
```
# -*- encoding: utf-8 -*-
broker_url = 'redis://127.0.0.1:6379/0'
result_backend = 'redis://127.0.0.1:6379/1'
```

tasks.py 内容：

```
# -*- encoding: utf-8 -*-
from .celery import app


@app.task
def print_task(a, b):
    print("任务函数正在执行....")
    return a + b


@app.task
def print_task1(a, b):
    print("任务函数正在执行....")
    return a + b


@app.task
def print_task2(a, b):
    print("任务函数正在执行....")
    return a + b


@app.task
def print_task3(a, b):
    print("任务函数正在执行....")
    return a + b
```

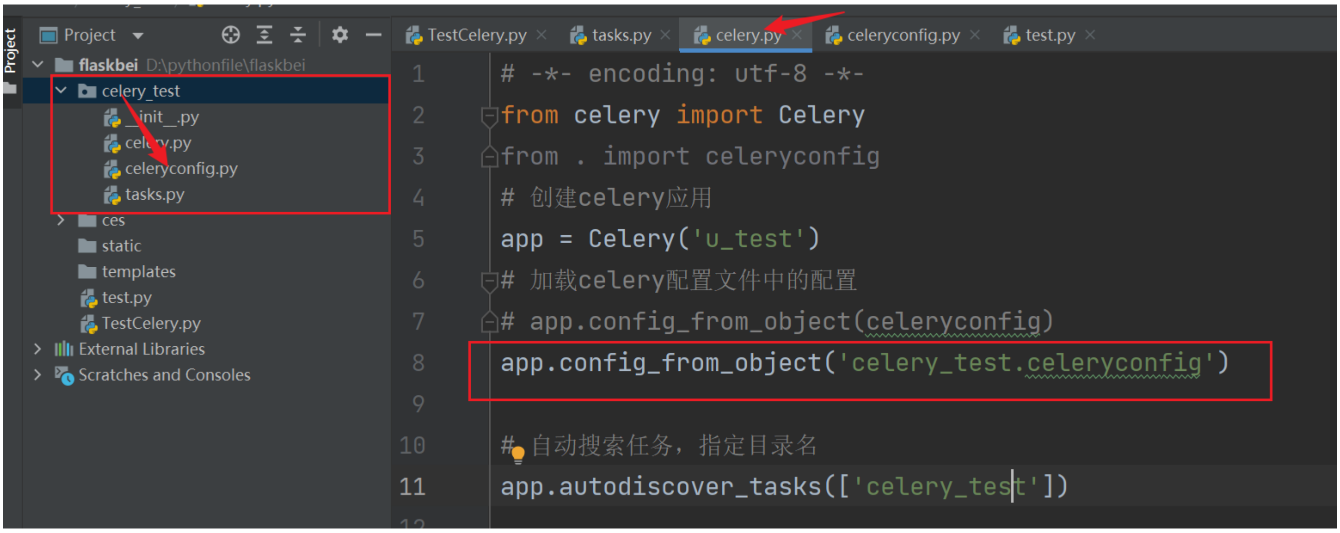
启动worker并制定队列名称为 'myqueue':

```
celery -A celery_test worker -l info -Qmyqueue

# 注意 -Q后面就是队列名称 ---禁止中间存在特殊符号及空格
# windows
celery -A celery_test worker -l info -Qmyqueue --pool=solo
```



```
[2022-10-28 16:07:27,694: CRITICAL/MainProcess] Unrecoverable error: TypeError("can't pickle module obj
ects")
Traceback (most recent call last):
  File "D:\EVNS\flask_bei\lib\site-packages\celery\worker\worker.py", line 203, in start
    self.blueprint.start(self)
  File "D:\EVNS\flask_bei\lib\site-packages\celery\bootsteps.py", line 116, in start
    step.start(parent)
  File "D:\EVNS\flask_bei\lib\site-packages\celery\bootsteps.py", line 365, in start
    return self.obj.start()
  File "D:\EVNS\flask_bei\lib\site-packages\celery\concurrency\base.py", line 129, in start
    self.on_start()
  File "D:\EVNS\flask_bei\lib\site-packages\celery\concurrency\prefork.py", line 115, in on_start
    **self.options)
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\pool.py", line 1046, in __init__
    self._create_worker_process(i)
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\pool.py", line 1158, in _create_worker_process
    w.start()
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\process.py", line 124, in start
    self._popen = self._Popen(self)
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\context.py", line 383, in _Popen
    return Popen(process_obj)
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\popen_spawn_win32.py", line 83, in __init__
    reduction.dump(process_obj, to_child)
  File "D:\EVNS\flask_bei\lib\site-packages\billiard\reduction.py", line 99, in dump
    ForkingPickler(file, protocol).dump(obj)
TypeError: can't pickle module objects
```

这是因为配置信息引入问题导致的
调整如下：



```
# -*- encoding: utf-8 -*-
from celery import Celery
from . import celeryconfig
# 创建celery应用
app = Celery('u_test')
# 加载celery配置文件中的配置
# app.config_from_object(celeryconfig)
app.config_from_object('celery_test.celeryconfig')

# 自动搜索任务，指定目录名
app.autodiscover_tasks(['celery_test'])
```

9.调用任务

```
调用任务之前一直使用delay()方法，还有apply_async()方法也可以调用任务,

apply_async()方法可以接收参数、执行选项，实际上delay方法也是调用
apply_async()方法，但是不能配置执行选项。
```

```
print_task.apply_async((2, 2), queue='myqueue', countdown=10)
```

```
任务my_task将会被发送到myqueue队列中，并且在发送10秒之后执行。
```

```
如果想要使用delay()方法，并且需要配置执行选项，
可以使用signature函数进行封装，将参数以及选项封装到任务函数中。

In [31]: a = tasks.print_task.signature((3,2),countdown=10)
In [32]: a.delay()
```

10.任务组

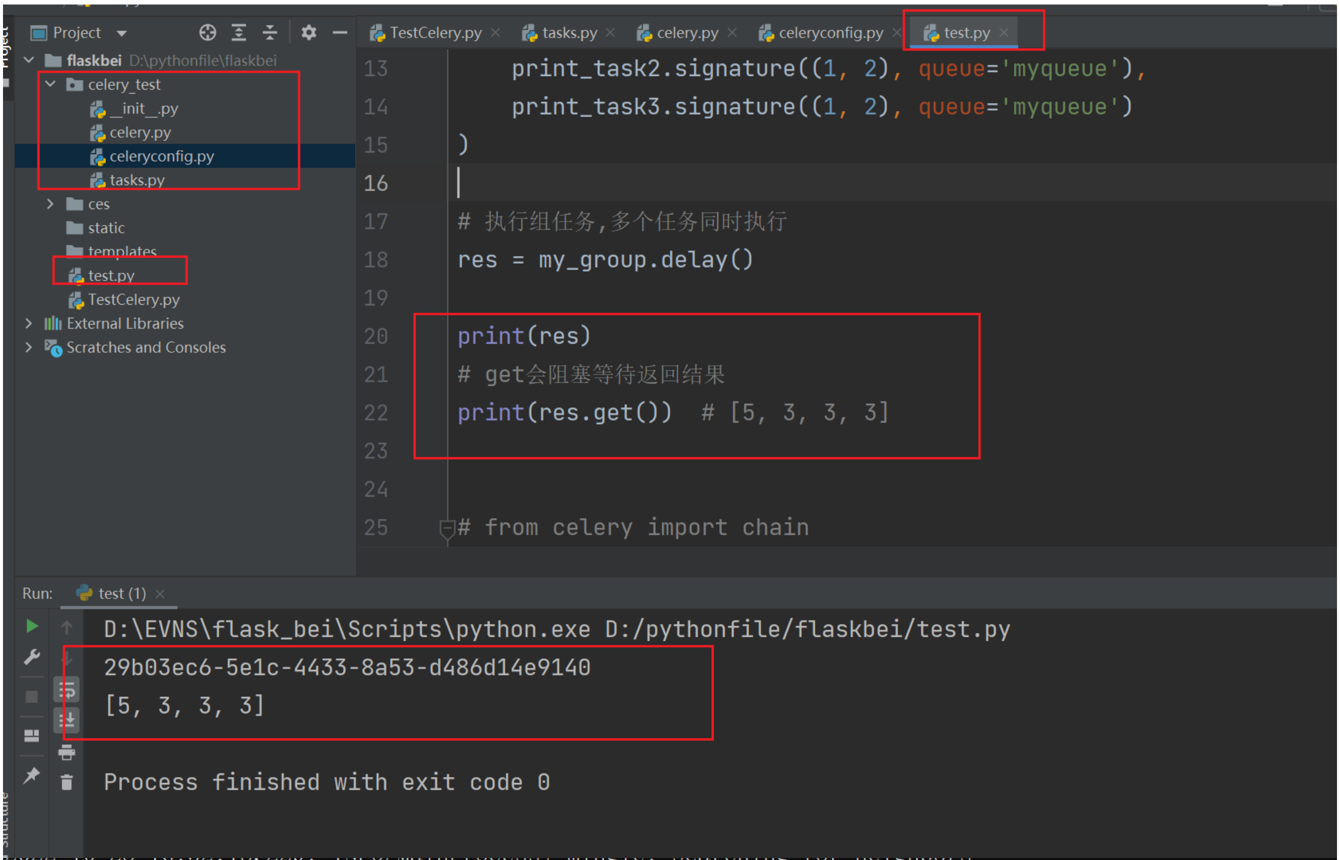
```
group: 一组任务并行（同时）执行，返回一组返回值，并可以按顺序检索返回值。

chain: 任务一个一个执行，一个执行完将执行return结果传递给下一个任务函数.
```

```
celery -A celery_test worker -l info -Qmyqueue --pool=solo
```

group使用：

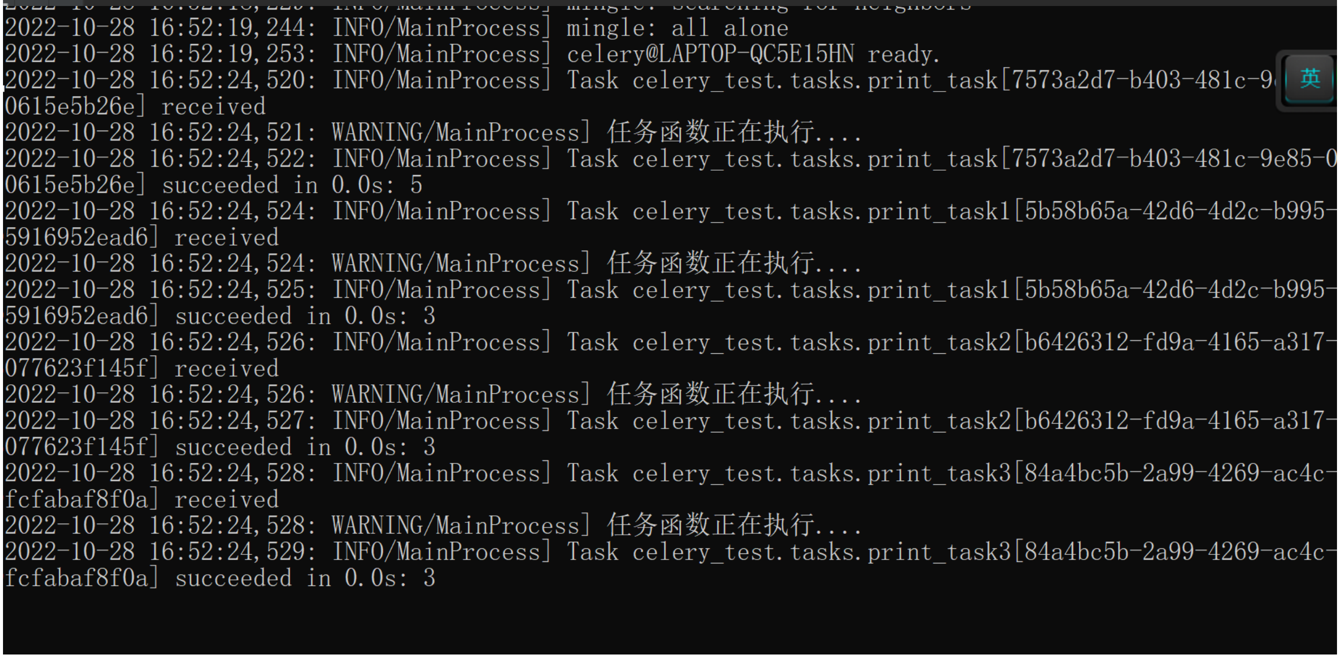
```
在celery_test同级目录下，创建一个test.py文件
```

```
# -*- encoding: utf-8 -*-
from celery_test.tasks import print_task
from celery_test.tasks import print_task1
from celery_test.tasks import print_task2
from celery_test.tasks import print_task3
from celery import group

# 将多个signature放入同一组中
my_group = group(
    print_task.signature((2, 3), queue='myqueue'),
    print_task1.signature((1, 2), queue='myqueue'),
    print_task2.signature((1, 2), queue='myqueue'),
    print_task3.signature((1, 2), queue='myqueue')
)

# 执行组任务,多个任务同时执行
res = my_group.delay()

# get会阻塞等待返回结果
print(res.get())  # [5, 3, 3, 3]
```



```
    print_task2.signature((1, 2), queue='myqueue'),
    print_task3.signature((1, 2), queue='myqueue')
)

# 执行组任务,多个任务同时执行
res = my_group.delay()

print(res)
# get会阻塞等待返回结果
print(res.get())  # [5, 3, 3, 3]


# from celery import chain
```

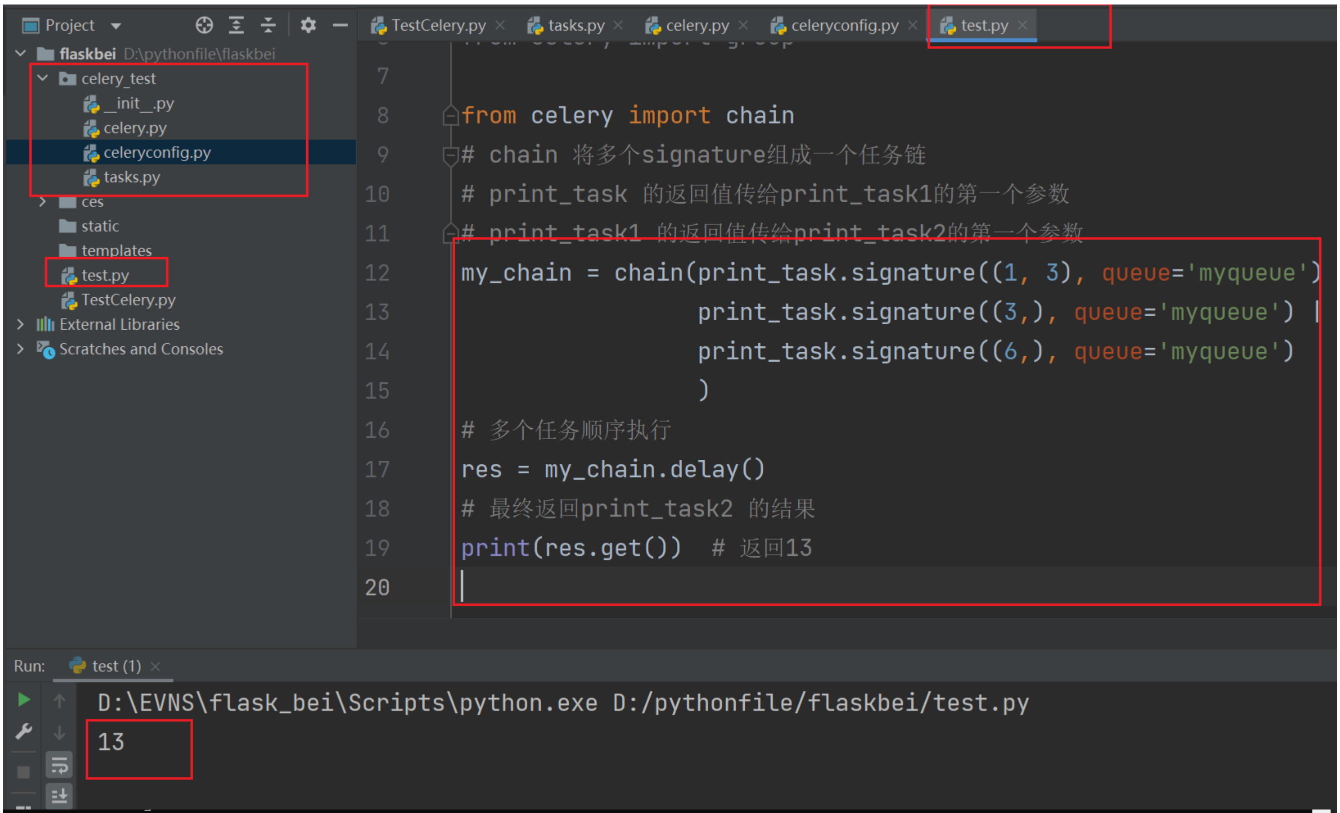
```
D:\EVNS\flask_bei\Scripts\python.exe D:/pythonfile/flaskbei/test.py
29b03ec6-5e1c-4433-8a53-d486d14e9140
[5, 3, 3, 3]

Process finished with exit code 0
```



```
2022-10-28 16:52:19,244: INFO/MainProcess] mingle: all alone
2022-10-28 16:52:19,253: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
2022-10-28 16:52:24,520: INFO/MainProcess] Task celery_test.tasks.print_task[7573a2d7-b403-481c-9
0615e5b26e] received
2022-10-28 16:52:24,521: WARNING/MainProcess] 任务函数正在执行....
2022-10-28 16:52:24,522: INFO/MainProcess] Task celery_test.tasks.print_task[7573a2d7-b403-481c-9e85-0
0615e5b26e] succeeded in 0.0s: 5
2022-10-28 16:52:24,524: INFO/MainProcess] Task celery_test.tasks.print_task1[5b58b65a-42d6-4d2c-b995-
5916952ead6] received
2022-10-28 16:52:24,524: WARNING/MainProcess] 任务函数正在执行....
2022-10-28 16:52:24,525: INFO/MainProcess] Task celery_test.tasks.print_task1[5b58b65a-42d6-4d2c-b995-
5916952ead6] succeeded in 0.0s: 3
2022-10-28 16:52:24,526: INFO/MainProcess] Task celery_test.tasks.print_task2[b6426312-fd9a-4165-a317-
077623f145f] received
2022-10-28 16:52:24,526: WARNING/MainProcess] 任务函数正在执行....
2022-10-28 16:52:24,527: INFO/MainProcess] Task celery_test.tasks.print_task2[b6426312-fd9a-4165-a317-
077623f145f] succeeded in 0.0s: 3
2022-10-28 16:52:24,528: INFO/MainProcess] Task celery_test.tasks.print_task3[84a4bc5b-2a99-4269-ac4c-
fcfabaf8f0a] received
2022-10-28 16:52:24,528: WARNING/MainProcess] 任务函数正在执行....
2022-10-28 16:52:24,529: INFO/MainProcess] Task celery_test.tasks.print_task3[84a4bc5b-2a99-4269-ac4c-
fcfabaf8f0a] succeeded in 0.0s: 3
```

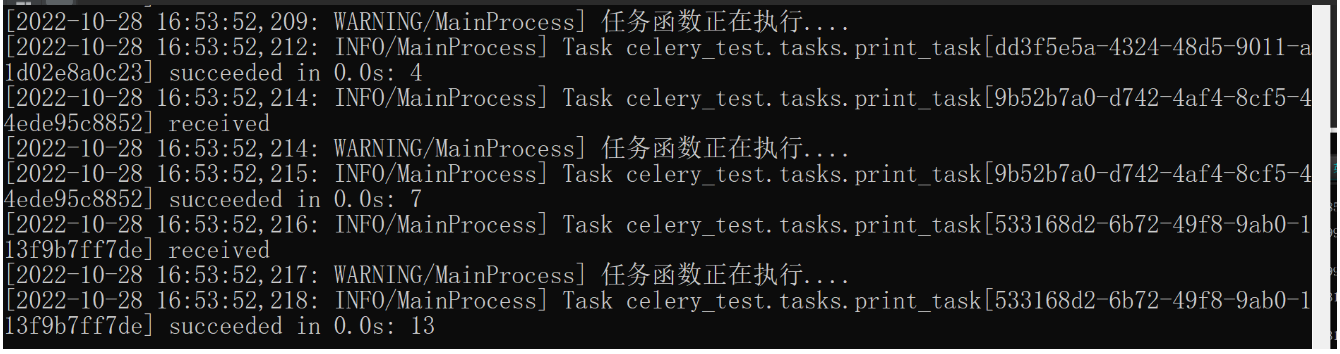
chain 使用：

```
# -*- encoding: utf-8 -*-
from celery_test.tasks import print_task
from celery_test.tasks import print_task1
from celery_test.tasks import print_task2
from celery_test.tasks import print_task3
from celery import chain


# chain 将多个signature组成一个任务链
# print_task 的返回值传给print_task1的第一个参数
# print_task1 的返回值传给print_task2的第一个参数
my_chain = chain(print_task.signature((1, 3), queue='myqueue') |
                 print_task.signature((3,), queue='myqueue') |
                 print_task.signature((6,), queue='myqueue')
                 )
# 多个任务顺序执行
res = my_chain.delay()
# 最终返回print_task2 的结果
print(res.get())  # 返回13
```



```
from celery import group

from celery import chain
# chain 将多个signature组成一个任务链
# print_task 的返回值传给print_task1的第一个参数
# print_task1 的返回值传给print_task2的第一个参数
my_chain = chain(print_task.signature((1, 3), queue='myqueue')
                 print_task.signature((3,), queue='myqueue') |
                 print_task.signature((6,), queue='myqueue')
                 )
# 多个任务顺序执行
res = my_chain.delay()
# 最终返回print_task2 的结果
print(res.get())  # 返回13
```

```
D:\EVNS\flask_bei\Scripts\python.exe D:/pythonfile/flaskbei/test.py
13
```



```
[2022-10-28 16:53:52,209: WARNING/MainProcess] 任务函数正在执行....
[2022-10-28 16:53:52,212: INFO/MainProcess] Task celery_test.tasks.print_task[dd3f5e5a-4324-48d5-9011-a
1d02e8a0c23] succeeded in 0.0s: 4
[2022-10-28 16:53:52,214: INFO/MainProcess] Task celery_test.tasks.print_task[9b52b7a0-d742-4af4-8cf5-4
4ede95c8852] received
[2022-10-28 16:53:52,214: WARNING/MainProcess] 任务函数正在执行....
[2022-10-28 16:53:52,215: INFO/MainProcess] Task celery_test.tasks.print_task[9b52b7a0-d742-4af4-8cf5-4
4ede95c8852] succeeded in 0.0s: 7
[2022-10-28 16:53:52,216: INFO/MainProcess] Task celery_test.tasks.print_task[533168d2-6b72-49f8-9ab0-1
13f9b7ff7de] received
[2022-10-28 16:53:52,217: WARNING/MainProcess] 任务函数正在执行....
[2022-10-28 16:53:52,218: INFO/MainProcess] Task celery_test.tasks.print_task[533168d2-6b72-49f8-9ab0-1
13f9b7ff7de] succeeded in 0.0s: 13
```

11.任务路由

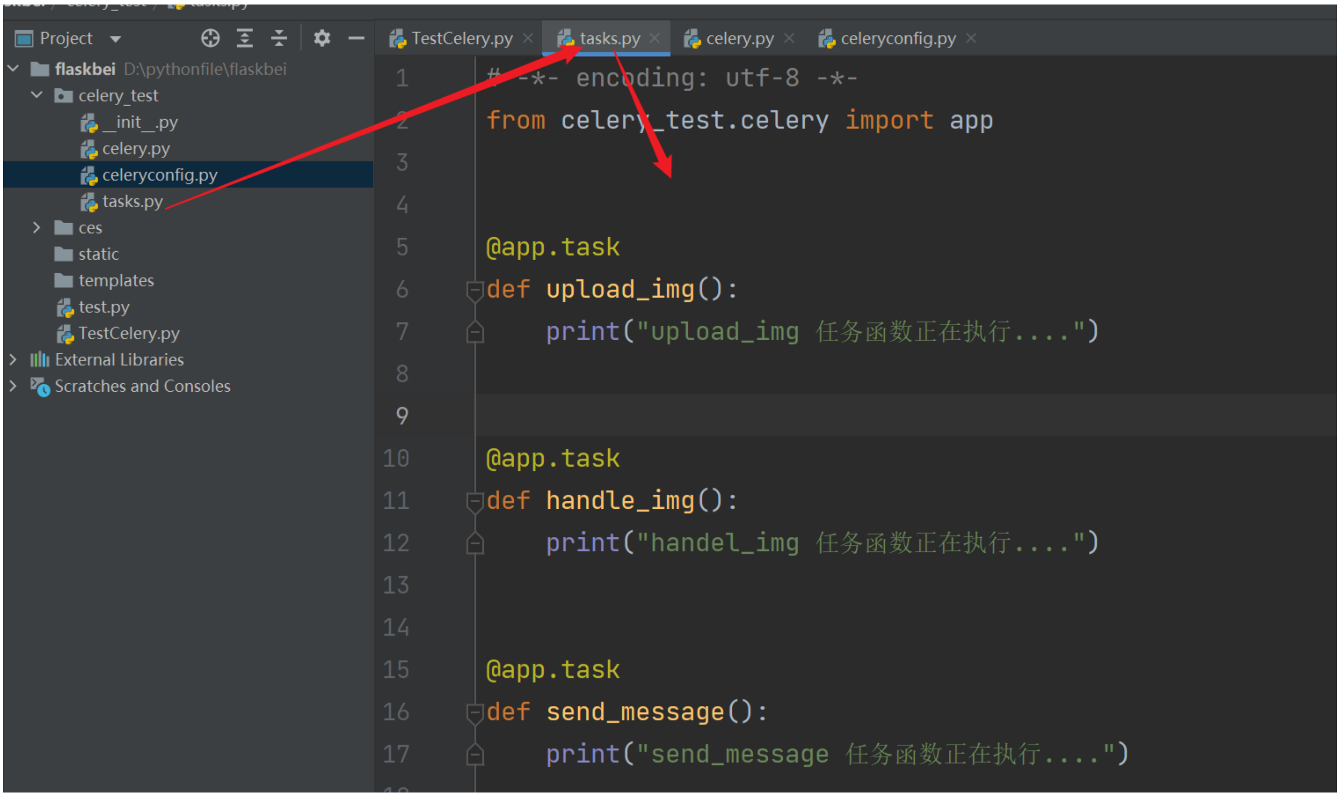
在生产中可以有多个worker，分别处理不同的任务。比如一个worker处理图片上传，一个worker专门处理短信发送。

我们创建两个队列，一个专门用于图片上传任务队列和图像处理，一个用来短信发送任务队列。

celery支持队列路由设置的，也可以在调用任务的时候指定队列。

首先在tasks.py中增加任务函数：

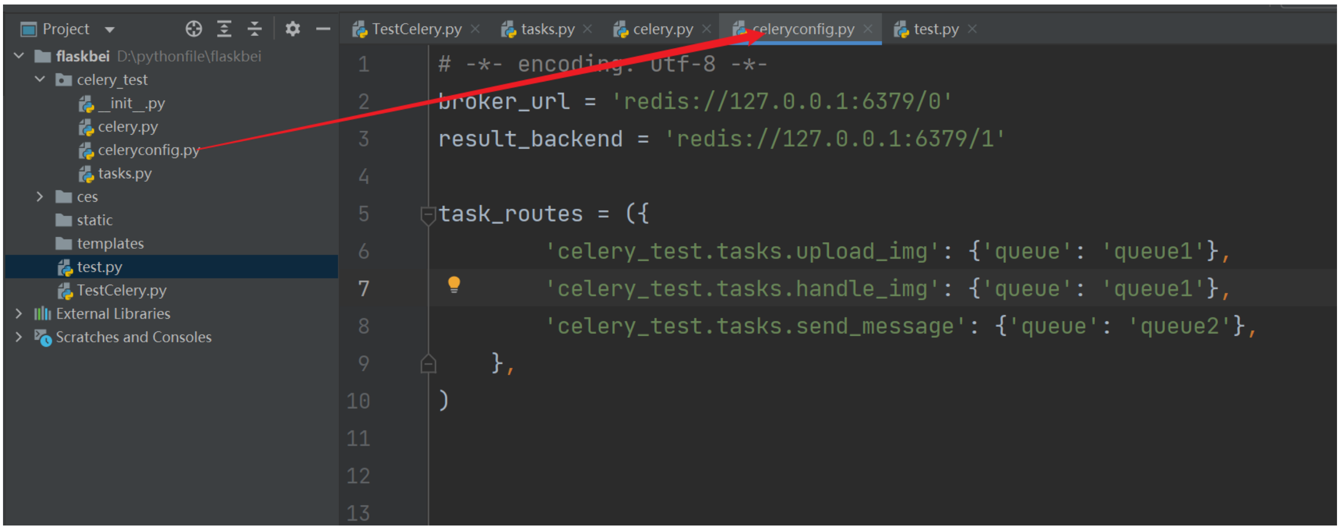
```
# -*- encoding: utf-8 -*-
from celery_test.celery import app


@app.task
def upload_img():
    print("upload_img 任务函数正在执行....")


@app.task
def handle_img():
    print("handel_img 任务函数正在执行....")


@app.task
def send_message():
    print("send_message 任务函数正在执行....")
```



```
# -*- encoding: utf-8 -*-
from celery_test.celery import app


@app.task
def upload_img():
    print("upload_img 任务函数正在执行....")


@app.task
def handle_img():
    print("handel_img 任务函数正在执行....")


@app.task
def send_message():
    print("send_message 任务函数正在执行....")
```

celeryconfig.py模块中配置 CELERY_ROUTES 选项：

```
# -*- encoding: utf-8 -*-
broker_url = 'redis://127.0.0.1:6379/0'
result_backend = 'redis://127.0.0.1:6379/1'

task_routes = ({
        'celery_test.tasks.upload_img': {'queue': 'queue1'},
        'celery_test.tasks.handle_img': {'queue': 'queue1'},
        'celery_test.tasks.send_message': {'queue': 'queue2'},
    },
)
```



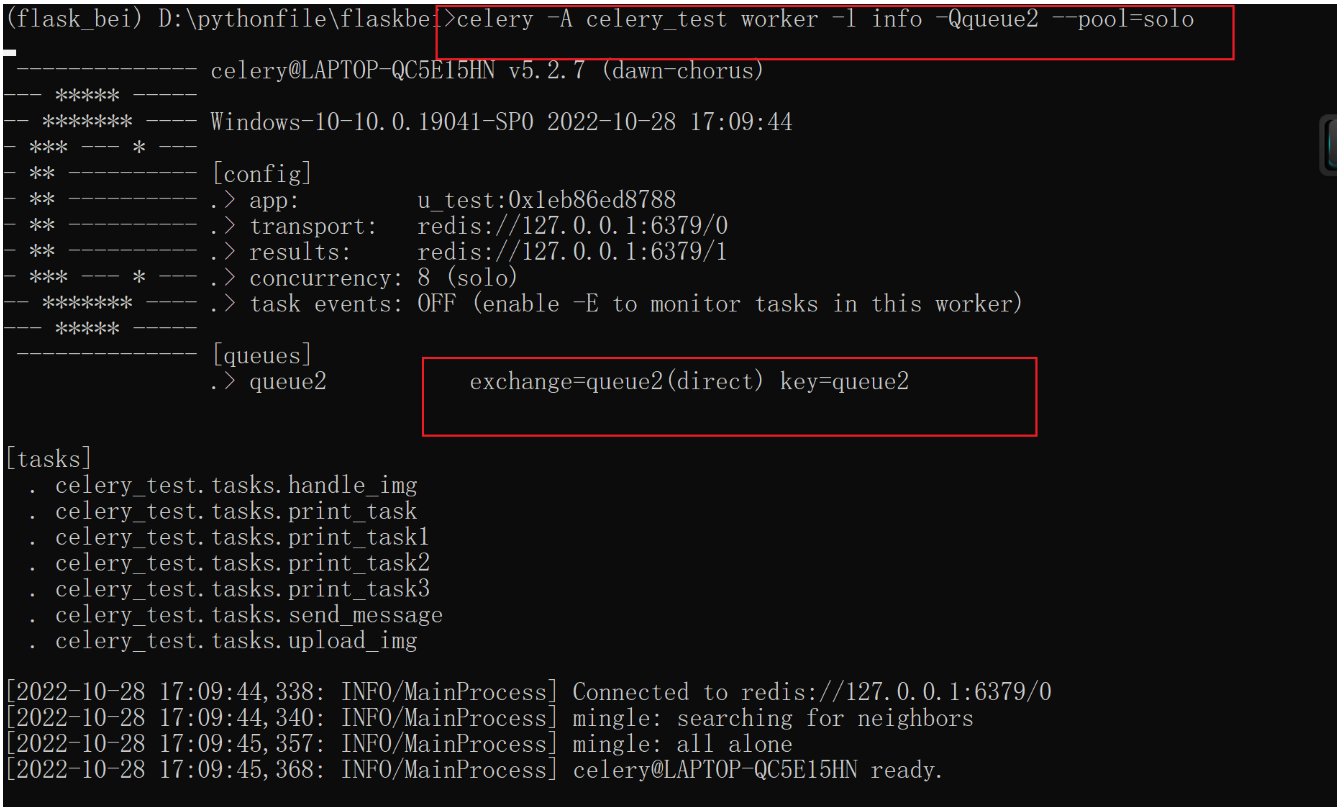
```
# -*- encoding: utf-8 -*-
broker_url = 'redis://127.0.0.1:6379/0'
result_backend = 'redis://127.0.0.1:6379/1'

task_routes = ({
        'celery_test.tasks.upload_img': {'queue': 'queue1'},
        'celery_test.tasks.handle_img': {'queue': 'queue1'},
        'celery_test.tasks.send_message': {'queue': 'queue2'},
    },
)
```

在test.py模块中将调用任务：

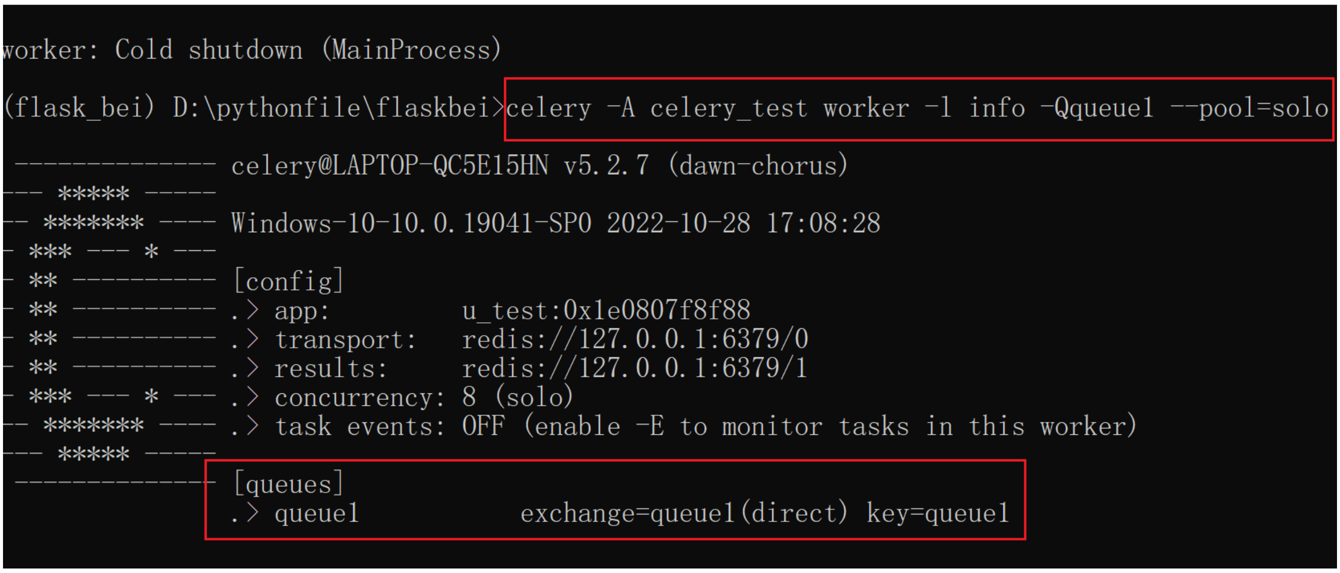
```
from celery_test.tasks import *
# 将任务加入队列
a = upload_img.delay()
print(a)

b = handle_img.delay()
print(b)

c = send_message.delay()
print(c)
```

```
打开两个终端开启两个worker，两个队列
celery -A celery_test worker -l info -Qqueue1 --pool=solo
celery -A celery_test worker -l info -Qqueue2 --pool=solo
```

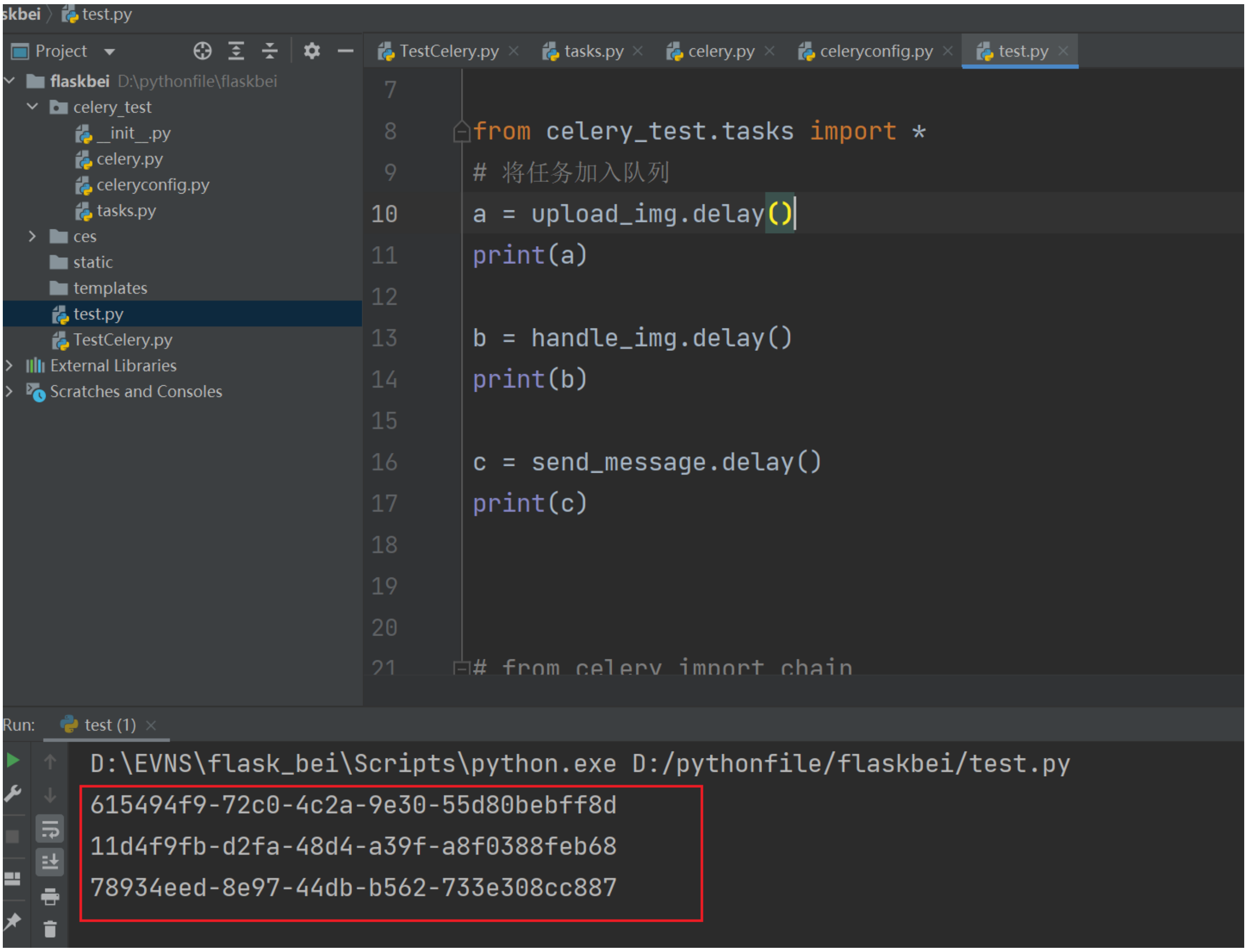
```
也执行一个worker处理两个队列
celery -A celery_test worker -l info -Qqueue1,queue2 --pool=solo
```



```
(flask_bei) D:\pythonfile\flaskbei>celery -A celery_test worker -l info -Qqueue2 --pool=solo

 -------------- celery@LAPTOP-QC5E15HN v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 17:09:44
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         u_test:0x1eb86ed8788
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> queue2           exchange=queue2(direct) key=queue2


[tasks]
  . celery_test.tasks.handle_img
  . celery_test.tasks.print_task
  . celery_test.tasks.print_task1
  . celery_test.tasks.print_task2
  . celery_test.tasks.print_task3
  . celery_test.tasks.send_message
  . celery_test.tasks.upload_img

[2022-10-28 17:09:44,338: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 17:09:44,340: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 17:09:45,357: INFO/MainProcess] mingle: all alone
[2022-10-28 17:09:45,368: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
```

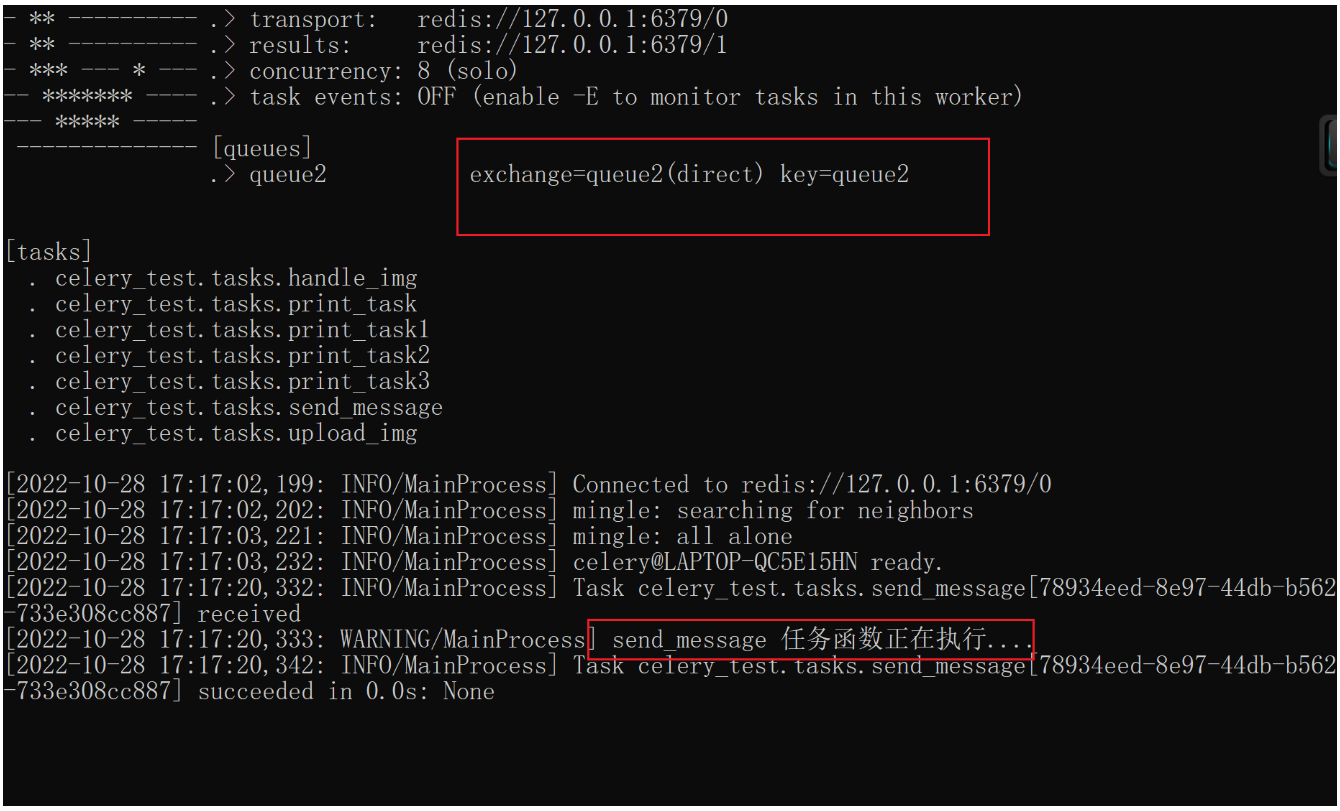


```
worker: Cold shutdown (MainProcess)

(flask_bei) D:\pythonfile\flaskbei>celery -A celery_test worker -l info -Qqueue1 --pool=solo

 -------------- celery@LAPTOP-QC5E15HN v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 17:08:28
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         u_test:0x1e0807f8f88
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> queue1           exchange=queue1(direct) key=queue1
```

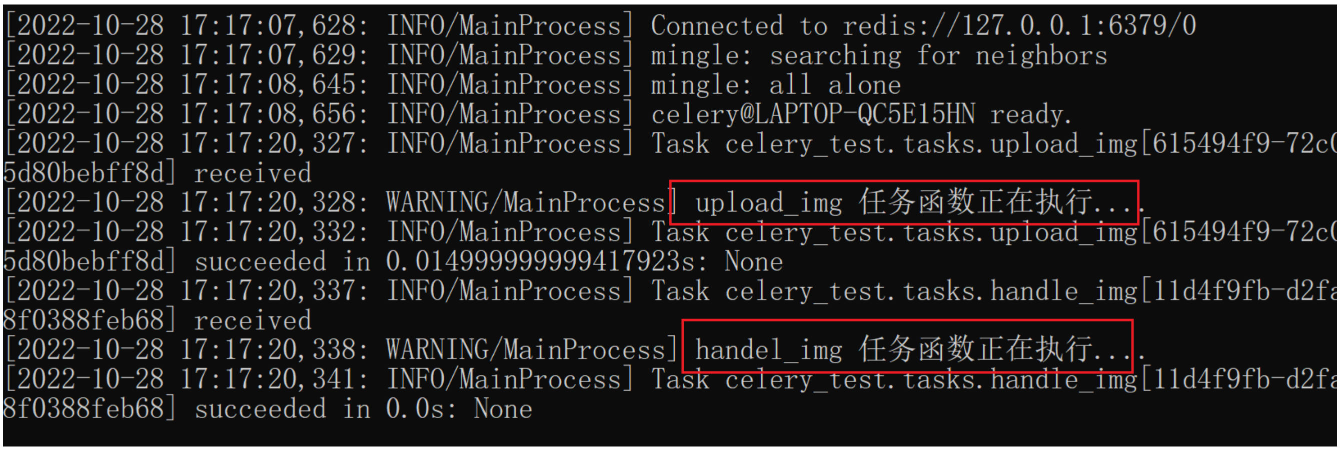


```
from celery_test.tasks import *
# 将任务加入队列
a = upload_img.delay()
print(a)

b = handle_img.delay()
print(b)

c = send_message.delay()
print(c)



# from celery import chain
```

```
D:\EVNS\flask_bei\Scripts\python.exe D:/pythonfile/flaskbei/test.py
615494f9-72c0-4c2a-9e30-55d80bebff8d
11d4f9fb-d2fa-48d4-a39f-a8f0388feb68
78934eed-8e97-44db-b562-733e308cc887
```

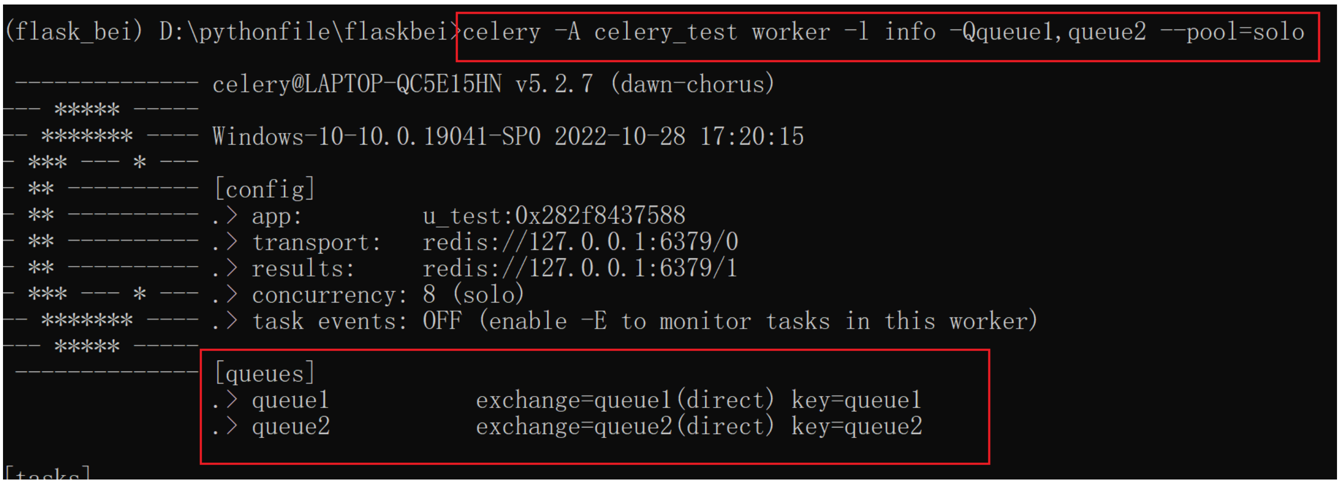


```
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> queue2           exchange=queue2(direct) key=queue2


[tasks]
  . celery_test.tasks.handle_img
  . celery_test.tasks.print_task
  . celery_test.tasks.print_task1
  . celery_test.tasks.print_task2
  . celery_test.tasks.print_task3
  . celery_test.tasks.send_message
  . celery_test.tasks.upload_img

[2022-10-28 17:17:02,199: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 17:17:02,202: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 17:17:03,221: INFO/MainProcess] mingle: all alone
[2022-10-28 17:17:03,232: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
[2022-10-28 17:17:20,332: INFO/MainProcess] Task celery_test.tasks.send_message[78934eed-8e97-44db-b562
-733e308cc887] received
[2022-10-28 17:17:20,333: WARNING/MainProcess] send_message 任务函数正在执行....
[2022-10-28 17:17:20,342: INFO/MainProcess] Task celery_test.tasks.send_message[78934eed-8e97-44db-b562
-733e308cc887] succeeded in 0.0s: None
```



```
[2022-10-28 17:17:07,628: INFO/MainProcess] Connected to redis://127.0.0.1:6379/0
[2022-10-28 17:17:07,629: INFO/MainProcess] mingle: searching for neighbors
[2022-10-28 17:17:08,645: INFO/MainProcess] mingle: all alone
[2022-10-28 17:17:08,656: INFO/MainProcess] celery@LAPTOP-QC5E15HN ready.
[2022-10-28 17:17:20,327: INFO/MainProcess] Task celery_test.tasks.upload_img[615494f9-72c
5d80bebff8d] received
[2022-10-28 17:17:20,328: WARNING/MainProcess] upload_img 任务函数正在执行....
[2022-10-28 17:17:20,332: INFO/MainProcess] Task celery_test.tasks.upload_img[615494f9-72c
5d80bebff8d] succeeded in 0.014999999999417923s: None
[2022-10-28 17:17:20,337: INFO/MainProcess] Task celery_test.tasks.handle_img[11d4f9fb-d2f
8f0388feb68] received
[2022-10-28 17:17:20,338: WARNING/MainProcess] handel_img 任务函数正在执行....
[2022-10-28 17:17:20,341: INFO/MainProcess] Task celery_test.tasks.handle_img[11d4f9fb-d2f
8f0388feb68] succeeded in 0.0s: None
```

一个worker处理两个队列



```
(flask_bei) D:\pythonfile\flaskbei>celery -A celery_test worker -l info -Qqueue1,queue2 --pool=solo

 -------------- celery@LAPTOP-QC5E15HN v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Windows-10-10.0.19041-SP0 2022-10-28 17:20:15
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         u_test:0x282f8437588
- ** ---------- .> transport:   redis://127.0.0.1:6379/0
- ** ---------- .> results:     redis://127.0.0.1:6379/1
- *** --- * --- .> concurrency: 8 (solo)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> queue1           exchange=queue1(direct) key=queue1
                .> queue2           exchange=queue2(direct) key=queue2


[tasks]
```