问题1：celery启动worker后放入任务，报错

WARNING/MainProcess] Received and deleted unknown message. Wrong destination?!?

参考 <https://github.com/celery/celery/issues/3675>

1. pip uninstall librabbitmq
2. 将py文件中celery配置时的broker由ampq改为pyampq

如：app = Celery('tasks', backend='amqp', broker='pyamqp://guest@localhost//')

问题2：启动celery的网页版监控页面flower，观察不到信息

解决：先在命令行启动woker后，再celery flower观察

问题3：启动多个worker

如果这样做你需要用包含hostname参数的节点名来使每个worke有r唯一的名字。

如：celery -A proj worker --loglevel=INFO --concurrency=10 -n worker1.%h

参考 <http://blog.csdn.net/happyanger6/article/details/51793490>

<http://docs.jinkan.org/docs/celery/userguide/workers.html>

问题4：使用RabbitMQ

参考<http://docs.jinkan.org/docs/celery/getting-started/brokers/rabbitmq.html>

问题5：终端运行worker

celery multi start –w1 –A proj –l info

将会把log文件等放在启动终端的那个文件夹下

Restart, stop同理

指定存放log的文件夹: celery multi start w1 -A proj -l info --pidfile=/var/run/celery/%n.pid

--logfile=/var/log/celery/%n.pid

高级功能，如celery multi start 10 -A proj -l info -Q:1-3 images,video -Q:4,5 data -Q default -L:4,5 debug 批量运行并设置worker，参考multi的API用法

问题6：子任务

Res = add.subtask((2, 2), countdown=10)

Res.get()

Keyword arguments can also be added later

问题7：数据类型

Groups 组，并发执行

如：r = celery.group(add.s(i,i) for i in range(10))

Result = r().get()

Chains 链，链式执行

rr = celery.chain(add.s(4,76)|add.s(4,76)|add.s(4,76)))

Chords 相当于自动()的组

问题8：设置路由

1)设置该任务的路由队列

app.conf.update(

CELERY\_ROUTES = {

'proj.tasks.add': {'queue': 'hipri'},

},

)

或

>>> from proj.tasks import add

>>> add.apply\_async((2, 2), queue='hipri')

2)设置worker处理的路由队列

celery -A proj worker -Q hipri,celery

(the default queue is named celery for historical reasons)

问题9:远程控制

在终端显示存活的worker : celery -A proj inspect active

在终端显示当前所有worker正在处理的事件： celery -A proj events –dump

# Application

>>> from celery import Celery

>>> app = Celery()

>>> app

<Celery \_\_main\_\_:0x100469fd0>

## Configuration

1)app.conf.CELERY\_ENABLE\_UTC = True

2) config\_from\_object

1)通过celeryconfig.py

2) 通过cerleryconfig库

3) 通过自定义一个class，如

class Config:

CELERY\_ENABLE\_UTC = True

CELERY\_TIMEZONE = 'Europe/London'

app.config\_from\_object(Config)

3) config\_from\_envvar

### 检查设置

app.conf.table(with\_defaults=False, censored=True)

# Tasks

Task应当是幂等的。Celery通过让worker提前承认task来保证其只被一个worker获取。如果task确实是幂等的，可以把ask\_late设为True

通过装饰器在Application实例中注册，同时也可以设置参数

@app.task(serializer='json')

def create\_user(username, password):

User.objects.create(username=username, password=password)

### Name

名字必须唯一，最佳实践是将其按路径命名，如

@task(name='proj.tasks.add')

def add(x, y):

return x + y

### Context

@app.task(bind=True)

def dump\_context(self, x, y):

print('Executing task id {0.id}, args: {0.args!r} kwargs: {0.kwargs!r}'.format(

self.request))

bind是绑定app，从而传入的第一个self是app的指针

由self.requests可获取实例化后该任务的具体信息，如id等

### Log

worker会自动为你设置日志，你也可以手动配置日志。

你可以使用一个名为“celery.task”的特定logger，你可以从这个logger继承，这样你就能够自动在log里记录任务名称和taks id.

from celery.utils.log import get\_task\_logger

logger = get\_task\_logger(\_\_name\_\_)

@app.task

def add(x, y):

logger.info('Adding {0} + {1}'.format(x, y))

return x + y

(想想怎么保存日志到文件)

可以在函数里print() 这些print的内容会被定向到对应worker的终端窗口显示

### Retrying

@app.task(bind=True)

def send\_twitter\_status(self, oauth, tweet):

try:

twitter = Twitter(oauth)

twitter.update\_status(tweet)

except (Twitter.FailWhaleError, Twitter.LoginError) as exc:

raise self.retry(exc=exc)

异常被抛出的情况

1. 若没有raise 则重试达到了最大次数
2. 若有，则是raise抛出的

默认重试次数为3

可以设置重试的间隔时间：

@app.task(bind=True, default\_retry\_delay=30 \* 60) # retry in 30 minutes.

def add(self, x, y):

try:

…

except Exception as exc:

raise self.retry(exc=exc, countdown=60) # override the default and

# retry in 1 minute

### Creating pickleable exceptions

对于args需要继承Exception的\_\_Init\_\_

对于kargs需要拆开

以保证能正常被序列化

### There are a number of exceptions that can be used to signal this function to change how it treats the return of the task.

##### Ignore

The task may raise Ignore to force the worker to ignore the task

##### Retry

### 自定义任务类

from celery import Task

class DatabaseTask(Task):

abstract = True

\_db = None

@property

def db(self):

if self.\_db is None:

self.\_db = Database.connect()

return self.\_db

@app.task(base=DatabaseTask)

def process\_rows():

for row in process\_rows.db.table.all():

…

在worker派生的N个进程中，每个进程里有且只有一个Task实例作为进程内的全局变量，不同进程之间的Task实例不同（已验证）

### [How it works](http://docs.jinkan.org/docs/celery/userguide/tasks.html#id13)

When tasks are sent, no actual function code is sent with it, just the name of the task to execute. When the worker then receives the message it can look up the name in its task registry to find the execution code.

This means that your workers should always be updated with the same software as the client. This is a drawback, but the alternative is a technical challenge that has yet to be solved.

当代码更新时，worker需要被重启更新。因为之前的worker收到的只是tasks中方法的名字，而不是其代码，worker会根据名字寻找自己进程的对应代码并运行，这还是以前的代码。