# Celery

### 1 定义

Celery 是一个简单、灵活且可靠的，处理大量消息的分布式系统

它是一个专注于实时处理的任务队列，同时也支持任务调度

中文官网：http://docs.jinkan.org/docs/celery/

在线安装 sudo pip3 install -U Celery

离线安装

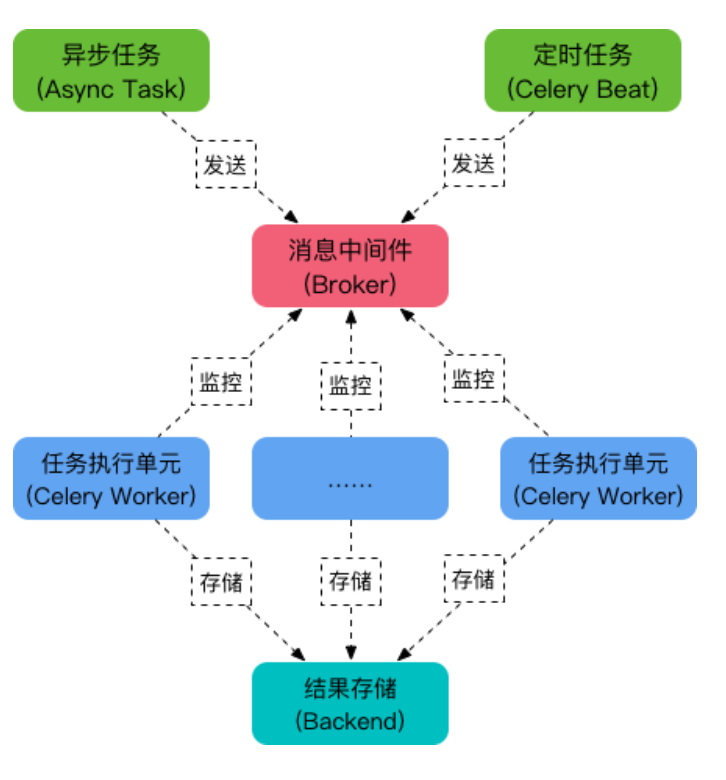
tar xvfz celery-0.0.0.tar.gz  
cd celery-0.0.0  
python3 setup.py build  
python3 setup.py install

名词解释：

broker - 消息传输的中间件，生产者一旦有消息发送，将发至broker；【RQ，redis】

backend - 用于存储消息/任务结果，如果需要跟踪和查询任务状态，则需添加要配置相关

worker - 工作者 - 消费/执行broker中消息/任务的进程

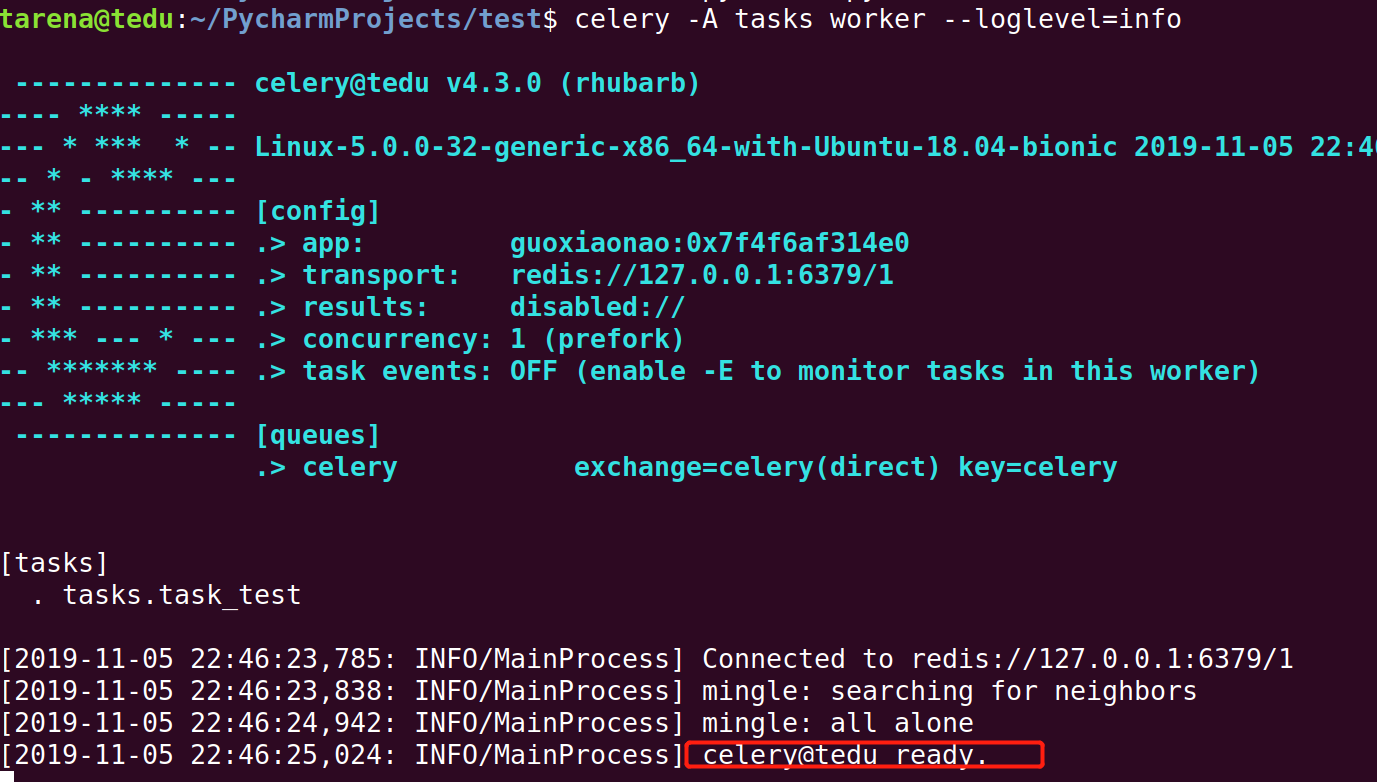


### 2 使用Celery

#### 1, 创建worker

#创建 tasks.py 文件  
  
from celery import Celery  
#初始化celery, 指定broker  
app = Celery('guoxiaonao', broker='redis://:password@127.0.0.1:6379/1')  
  
#若redis无密码，password可省略  
#app = Celery('guoxiaonao', broker='redis://:@127.0.0.1:6379/1')  
  
# 创建任务函数  
@app.task  
def task\_test():  
 print("task is running....")

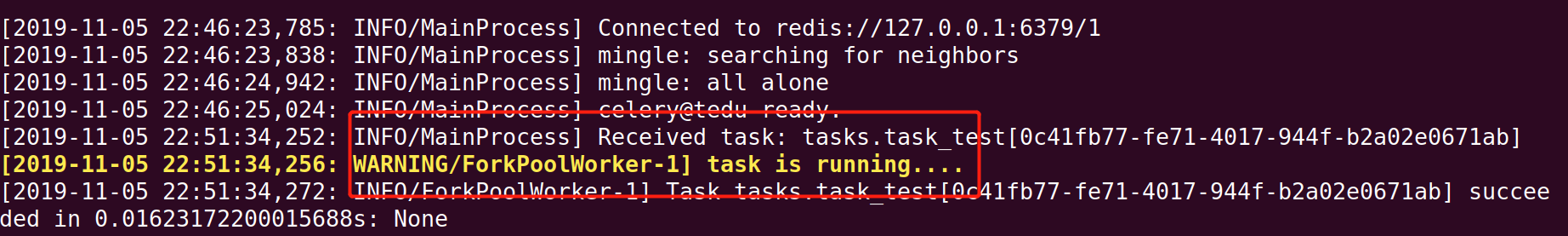
#Ubuntu 终端中, tasks.py文件同级目录下 执行  
celery -A tasks worker --loglevel=info  
#执行后终端显示如下，证明成功！



#### 2,创建生产者 - 推送任务

在tasks.py文件的同级目录进入 ipython3 执行 如下代码

from tasks import task\_test  
task\_test.delay()  
#执行后，worker终端中现如如下



#### 存储执行结果

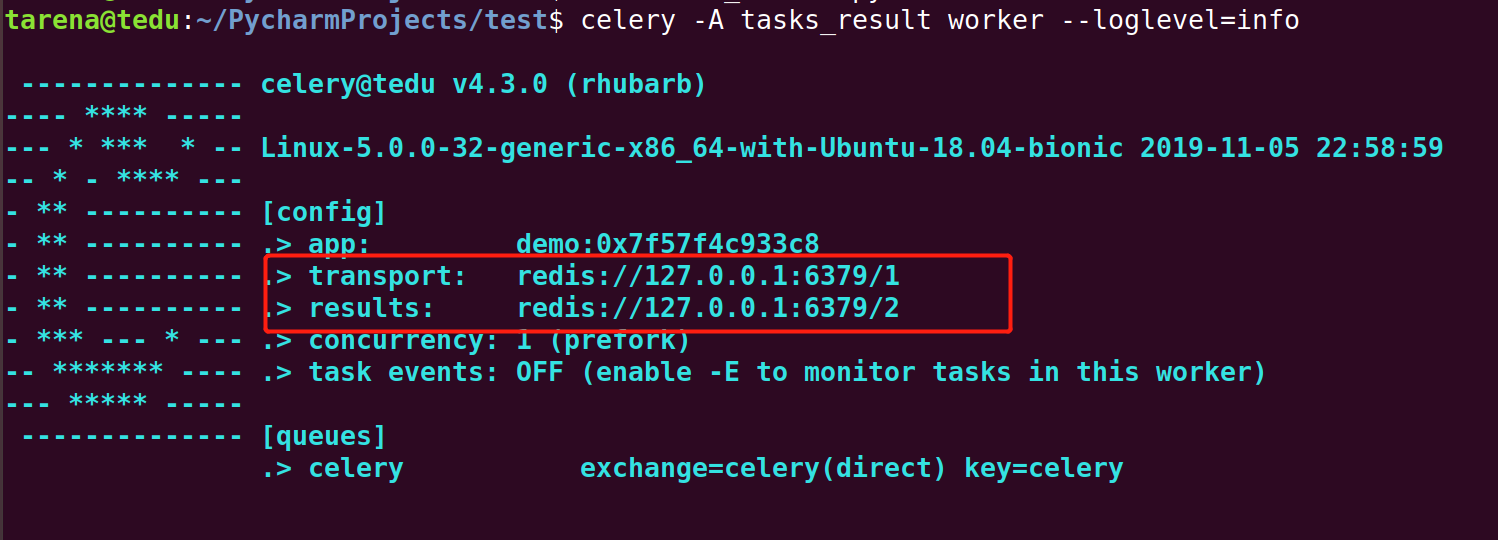
Celery提供存储任务执行结果的方案，需借助 redis 或 mysql 或Memcached 等

详情可见 http://docs.celeryproject.org/en/latest/reference/celery.result.html#module-celery.result

#创建 tasks\_result.py  
from celery import Celery  
app = Celery('demo',  
 broker='redis://@127.0.0.1:6379/1',  
 backend='redis://@127.0.0.1:6379/2',  
 )  
  
# 创建任务函数  
@app.task  
def task\_test(a, b):  
 print("task is running")  
 return a + b

tasks\_result.py 同级目录终端中-启动celery worker

celery -A tasks\_result worker --loglevel=info



在相同目录下 打开终端创建生产者 - 同【上步】；执行成功后，可调用如下方法取得执行结果

from tasks\_result import task\_test  
s = task\_test.delay(10,100)  
s.result

### 3 Django + Celery

1，创建项目+应用

#常规命令  
django-admin startproject test\_celery  
python manage.py startapp user

2，创建celery.py

在settings.py同级目录下 创建 celery.py文件

文件内容如下：

from celery import Celery  
from django.conf import settings  
import os  
  
# 为celery设置环境变量  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'test\_celery.settings')  
  
# 创建应用  
app = Celery("test\_celery")  
# 配置应用  
app.conf.update(  
 # 配置broker  
 BROKER\_URL='redis://:@127.0.0.1:6379/1',  
)  
# 设置app自动加载任务  
app.autodiscover\_tasks(settings.INSTALLED\_APPS)

3, 在应用模块【user目录下】创建tasks.py文件

文件内容如下：

from test\_celery.celery import app  
import time  
  
@app.task  
def task\_test():  
 print("task begin....")  
 time.sleep(10)  
 print("task over....")

4, 应用视图编写；内容如下：

from django.http import HttpResponse  
from .tasks import task\_test  
import datetime  
  
def test\_celery(request):  
 task\_test.delay()  
 now = datetime.datetime.now()  
 html = "return at %s"%(now.strftime('%H:%M:%S'))  
 return HttpResponse(html)

5, 分布式路由下添加 test\_celery函数对应路由，此过程略

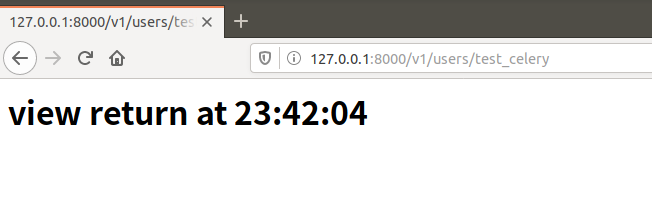
6, 启动django python3 manage.py runserver

7, 创建 celery worker

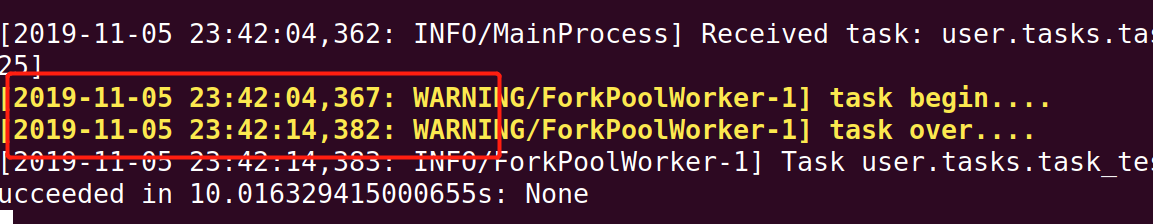
在项目路径下，即test\_celery 下 执行如下

celery -A test\_celery worker -l info

8，浏览器中执行对应url



worker终端中显示



### 4，生产环境 启动

#### 1，并发模式切换

默认并发采用 - prefork

推荐采用 - gevent 模式 - 协程模式

celery -A proj worker -P gevent -c 1000  
# P POOL Pool implementation: 支持 perfork or eventlet or gevent  
# C CONCURRENCY 并发数

#### 2，后台启动命令

nohup celery -A proj worker -P gevent -c 1000 > celery.log 2>&1 &  
  
#1，nohup: 忽略所有挂断（SIGHUP）信号  
#2，标准输入是文件描述符0。它是命令的输入，缺省是键盘，也可以是文件或其他命令的输出。  
#标准输出是文件描述符1。它是命令的输出，缺省是屏幕，也可以是文件。  
#标准错误是文件描述符2。这是命令错误的输出，缺省是屏幕，同样也可以是文件。  
#3，&符号：代表将命令在后台执行