

关于 Celery 的爬虫用法

[前言](#)

[简单任务讲解](#)

[改进](#)

[在celery里面执行的代码](#)

[获取celery执行任务的结果代码：](#)

[回顾总结](#)

[京东爬虫](#)

前言

为什么会写这篇文章呢？因为最近在爬取京东的时候，很深刻的体会到了celery的实用之处，我们就拿爬取京东的实战代码，来讲解一个celery在爬虫的过程中的一些用法。

我们来情景引入一下，京东中有一些商品是非常多评论的，而我们在爬取评论的时候，难道们的程序就需要白白浪费在爬取评论的时候吗？少说我们爬取的商品也有几千条，也不是说很多都有评论，但是起码五分之一是比较多评论的，我实际的测量过，等待的时间还是比较久的，之后我就考虑到，能不能换一种方式，将我们的爬虫程序分为两个部分。

- 爬取商品和店铺的基本信息
- 爬取商品的评论

这样，我们在爬取商品信息的同时，可以将爬取celery的任务分发给celery去执行，将任务存在缓存中，让他一个一个去执行这些，从而不会阻塞住我们主进程的代码运行，这样的效率是不是大大提升了呢？

简单任务讲解

我们通过一些小例子来理解一下大概的过程：

1. 主线程运行：爬取一些不怎么耗时的任务 --> 将耗时的任务给到celery去执行
2. celery执行任务：将任务存在 redis 1号数据库中，让结果存放在 redis 2号数据库中
3. 获取celery执行结果：从 redis 2号数据库中获取结果

```
# @author: Yxinmiracle
# @project: celeryke
# @file: main.py
# @time: 2020/4/25 15:54
import time

def getComments(count):
    time.sleep(5)
    comment = f"comment {count}"
    return comment

def getCommodity(count):
    comments = getComments(count)
    print(f"getCommodity {count} {comments}")
```

```
def main():
    for i in range(10):
        getCommodity(i)

if __name__ == '__main__':
    start_time = time.time()
    main()
    end_time = time.time()
    print(f"The total time is {end_time-start_time}")

# output
getCommodity 0 comment 0
getCommodity 1 comment 1
getCommodity 2 comment 2
getCommodity 3 comment 3
getCommodity 4 comment 4
getCommodity 5 comment 5
getCommodity 6 comment 6
getCommodity 7 comment 7
getCommodity 8 comment 8
getCommodity 9 comment 9
The total time is 50.005030393600464
```

从这段代码中我们很容易的看出来getComments是一个非常耗时的操作，直接拖慢了我们的程序的运行，但是getCommodity却是一个运行的很快的函数，我们能不能吧这两个函数分开来，并且通过异步去执行getComments函数，这样我们的效率就会提高很多，当然，你可能有疑问，为什么是异步呢？

回答：

Celery是基于Python开发的一个分布式任务队列框架，支持使用任务队列的方式在分布的机器/进程/线程上执行任务调度。它是Python写的库，但是它实现的通讯协议也可以使用ruby, php, javascript等调用。异步任务除了消息队列的后台执行的方式，还有一种则是定时计划任务。

Celery 是一个强大的分布式任务队列，它可以让任务的执行完全脱离主程序，甚至可以被分配到其他主机上运行。我们通常使用它来实现异步任务（async task）和定时任务（crontab）。

改进

在celery里面执行的代码

所以Celery是非常强大的，我们现在尝试一下将上面的任务改成celery来执行一下。

我们将耗时的任务放在另外一个.py文件去执行，在这里我给他命名：getSome.py

代码为：

```
# @author: Yxinmiracle
# @project: celeryke
# @file: getSome.py
# @time: 2020/4/25 16:32

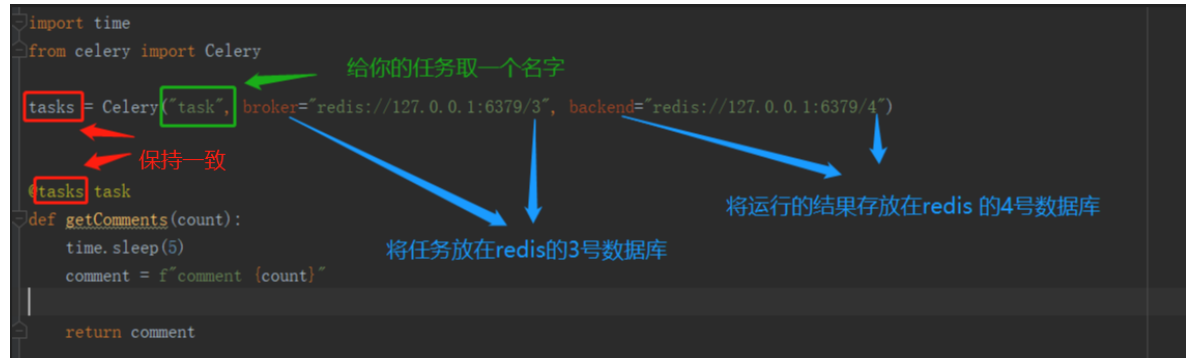
import time
from celery import Celery

tasks = Celery("task", broker="redis://127.0.0.1:6379/3",
backend="redis://127.0.0.1:6379/4")
```

```
@tasks.task
def getComments(count):
    time.sleep(5)
    comment = f"comment {count}"

    return comment
```

我们具体的看看里面的意思：



ok 我们给celery任务的代码已经写完了，我们先在来看看我们的主线程(main.py)的代码是怎么样子的：

```
# @author: Yxinmiracle
# @project: celeryke
# @file: main.py
# @time: 2020/4/25 15:54
import time
from getSome import getComments

def getCommodity(count):
    comments = getComments.delay(count)
    # 关于delay的解释请看下面
    print(f"getCommodity celery task ID is {comments}")

def main():
    for i in range(10):
        getCommodity(i)

if __name__ == '__main__':
    start_time = time.time()
    main()
    end_time = time.time()
    print(f"The total time is {end_time-start_time}")
```

- delay是一个celery 调用任务的方法的方法，还有其他调用任务的方法，不太常用，大家可以私底下去了解。

```
task.delay(args1, args2, kwargs=value_1, kwargs2=value_2)
```

要是执行任务的函数需要传参，那我们直接加在delay后面就好了。

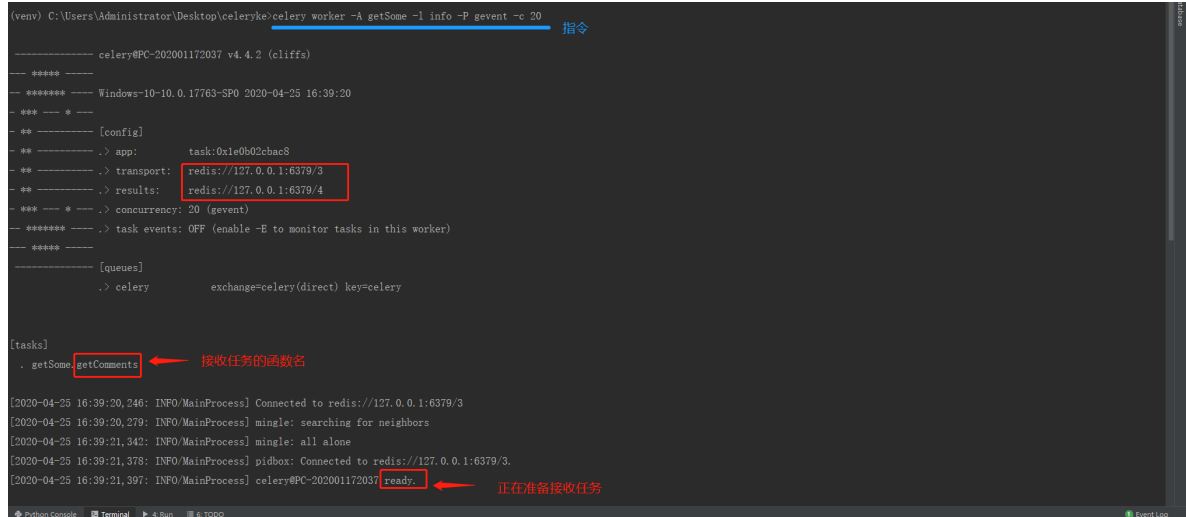
ok 现在我们万事俱备，如何启动celery准备接收任务呢？

我们需要一个启动命令，在我们的pycharm的终端中输入：

```
celery worker -A getSome -l info -P gevent -c 20
```

其中getSome的意思是，你的celery任务在哪个.py文件里面，我们就写哪个名字，就像我这里celery任务放在getSome.py文件里面，这里就写getSome，然后其中的20意思是有20个worker来帮助你做任务，理论上这个worker的数值是可以无限大的，但是我们没有必要弄这么大，默认是6个，只要设定小于等于6个都是6个worker，然后按下你的回车，celery就可以等待你开始发起任务了。

当你们看见如下图面就代表你们成功了：



```
(venv) C:\Users\Administrator\Desktop\celery\celery worker -A getSome -l info -P gevent -c 20 指令
----- celery@PC-202001172037 v4.4.2 (cliffs)
*****
***** Windows-10-10.0.17763-SP0 2020-04-25 16:39:20
***
** [config]
** .> app: task:0x1e0b02cbac8
** .> transport: redis://127.0.0.1:6379/3
** .> results: redis://127.0.0.1:6379/4
** .> concurrency: 20 (gevent)
** .> task events: OFF (enable -E to monitor tasks in this worker)
*****
** [queues]
** .> celery exchange=celery(direct) key=celery

[tasks]
. getSome getComments 接收任务的函数名

[2020-04-25 16:39:20.246: INFO/MainProcess] Connected to redis://127.0.0.1:6379/3
[2020-04-25 16:39:20.279: INFO/MainProcess] mingling: searching for neighbors
[2020-04-25 16:39:21.342: INFO/MainProcess] mingling: all alone
[2020-04-25 16:39:21.378: INFO/MainProcess] pidbox: Connected to redis://127.0.0.1:6379/3.
[2020-04-25 16:39:21.397: INFO/MainProcess] celery@PC-202001172037 ready. 正在准备接收任务
```

ok 我们现在右键运行我们的main.py程序，如下为我们的运行结果：

```
getCommodity celery task ID is f04b9477-2ba6-4f1e-8651-e4f634b5aa96
getCommodity celery task ID is fbd1c244-5d4c-4197-882f-bdb472a866f9
getCommodity celery task ID is 3e8bdd8e-de68-4eda-8357-bc68c81861ff
getCommodity celery task ID is b54cbd20-d896-4149-8ad8-0a7d4522ff4b
getCommodity celery task ID is 49ffe1fe-27c9-4c24-b4e2-4675aa2fdf3a
getCommodity celery task ID is 9f3e1bba-79d3-4e21-86d2-f59335ad9188
getCommodity celery task ID is 1c1abf8e-6b95-4ea7-9aaf-0cc9751ebc0e
getCommodity celery task ID is 9e27b2c1-073b-4970-89f9-16ab98b1ab6b
getCommodity celery task ID is 6fc4c1ae-e339-4cb3-931a-030f7d50d90e
getCommodity celery task ID is 89b2f7b3-9ec1-4ca9-90e5-9d6e2107d3b6
The total time is 0.4207894802093506
```

解释一下为什么只有0.42...

因为我们的主程序是不会等待celery去执行的，这个main.py的任务就是将产生的任务放在缓存里面，让worker一个一个去取出来慢慢执行，而不会影响到main.py的运行，所以只有0.42...秒

我们执行了10个任务，产生了10个任务的ID，可以发现这些ID很像UUID，这个ID有什么用呢？

答：

用法一：用来区分不同的任务并且拿到相应的结果

我们看一下celery那边打印的怎么样了？

```
Terminal: Local +
[2020-04-25 16:39:20,246: INFO/MainProcess] Connected to redis://127.0.0.1:6379/3
[2020-04-25 16:39:20,279: INFO/MainProcess] mingle: searching for neighbors
[2020-04-25 16:39:21,342: INFO/MainProcess] mingle: all alone
[2020-04-25 16:39:21,378: INFO/MainProcess] pidbox: Connected to redis://127.0.0.1:6379/3.
[2020-04-25 16:39:21,397: INFO/MainProcess] celery@PC-202001172037 ready.
[2020-04-25 16:40:16,280: INFO/MainProcess] Received task: getSome.getComments[06d26a65-3bd3-4a84-9ca5-59decbb401c6]
[2020-04-25 16:40:16,284: INFO/MainProcess] Received task: getSome.getComments[ab50fe79-f3f7-46e2-bc97-a5cb1879468e]
[2020-04-25 16:40:16,291: INFO/MainProcess] Received task: getSome.getComments[0234274f-0595-489a-9c8c-c9c3b7ec2d95]
[2020-04-25 16:40:16,295: INFO/MainProcess] Received task: getSome.getComments[e4c23531-721f-4e9e-bf72-2364fc27faf]
[2020-04-25 16:40:16,301: INFO/MainProcess] Received task: getSome.getComments[ac02d6b1-4c57-4983-9795-11700b34b4ee]
[2020-04-25 16:40:16,306: INFO/MainProcess] Received task: getSome.getComments[828046d6-f475-4006-bbc0-3a6b5fafd5be]
[2020-04-25 16:40:16,312: INFO/MainProcess] Received task: getSome.getComments[4e64c74b-d9d2-4789-b23b-d9979b132ec6]
[2020-04-25 16:40:16,316: INFO/MainProcess] Received task: getSome.getComments[2e2b301a-ebd4-42dd-921f-2e13dcba6a68]
[2020-04-25 16:40:16,326: INFO/MainProcess] Received task: getSome.getComments[5b996fc6-ef19-4c56-ba55-fd3c-7fe1b28e]
[2020-04-25 16:40:16,330: INFO/MainProcess] Received task: getSome.getComments[60970f68-3439-4f4a-aa95-59dbf5d741e4]
[2020-04-25 16:40:21,303: INFO/MainProcess] Task getSome.getComments[06d26a65-3bd3-4a84-9ca5-59decbb401c6] succeeded in 5.0310000000172295s: 'comment 0'
[2020-04-25 16:40:21,306: INFO/MainProcess] Task getSome.getComments[ab50fe79-f3f7-46e2-bc97-a5cb1879468e] succeeded in 5.01500000001397s: 'comment 1'
[2020-04-25 16:40:21,320: INFO/MainProcess] Task getSome.getComments[0234274f-0595-489a-9c8c-c9c3b7ec2d95] succeeded in 5.0310000000172295s: 'comment 2'
[2020-04-25 16:40:21,322: INFO/MainProcess] Task getSome.getComments[828046d6-f475-4006-bbc0-3a6b5fafd5be] succeeded in 5.01600000000326s: 'comment 3'
[2020-04-25 16:40:21,323: INFO/MainProcess] Task getSome.getComments[e4c23531-721f-4e9e-bf72-2364fc27faf] succeeded in 5.0310000000172295s: 'comment 3'
[2020-04-25 16:40:21,325: INFO/MainProcess] Task getSome.getComments[ac02d6b1-4c57-4983-9795-11700b34b4ee] succeeded in 5.01600000000326s: 'comment 4'
[2020-04-25 16:40:21,338: INFO/MainProcess] Task getSome.getComments[4e64c74b-d9d2-4789-b23b-d9979b132ec6] succeeded in 5.03200000000619s: 'comment 6'
[2020-04-25 16:40:21,342: INFO/MainProcess] Task getSome.getComments[2e2b301a-ebd4-42dd-921f-2e13dcba6a68] succeeded in 5.03200000000619s: 'comment 6'

工作 (worker) 们拿到了10个任务, 然后分别去执行, 他们是异步执行的。
分别拿到了他们相对应的结果
```

ok 现在我们来对比一下, 用了celery和没有用celery的时间区别到底有多大

这个是没有用celery的结果 --> The total time is 50.005030393600464

这个是用celery的结果 --> The total time is 0.4207894802093506 + 5.01600000000326 (celery是个任务中执行最久的一个任务)

这样一对比可以看见很大的区别了吧。

那么我们的结果在哪?

那么我们应该怎么把结果取出来呢?

带着你的疑问继续往下看

获取celery执行任务的结果代码:

我们上面说到过, 我们的结果存储在了redis的4号数据库中, 到底是怎么样的呢, 我们现在去瞧一眼看看个究竟:

```
127.0.0.1:6379[3]> select 4
OK
127.0.0.1:6379[4]> keys *
1) "celery-task-meta-9f3e1bba-79d3-4e21-86d2-f59335ad9188"
2) "celery-task-meta-9e27b2c1-073b-4970-89f9-16ab98b1ab6b"
3) "celery-task-meta-f04b9477-2ba6-4f1e-8651-e4f634b5aa96"
4) "celery-task-meta-1c1abf8e-6b95-4ea7-9aaf-0cc9751ebc0e"
5) "celery-task-meta-3e8bdd8e-de68-4eda-8357-bc68c81861ff"
6) "celery-task-meta-89b2f7b3-9ec1-4ca9-90e5-9d6e2107d3b6"
7) "celery-task-meta-b54cbd20-d896-4149-8ad8-0a7d4522ff4b"
8) "celery-task-meta-6fc4c1ae-e339-4cb3-931a-030f7d50d90e"
9) "celery-task-meta-fbd1c244-5d4c-4197-882f-bdb472a866f9"
10) "celery-task-meta-49ffe1fe-27c9-4c24-b4e2-4675aa2fdf3a"
```

可以很明显的发现这10个任务ID就是我们上面main.py产生的10个任务ID, 那么我们现在来编写一个程序, 将里面存储的结果取出来, 我的程序名称是redis_back.py, 代码如下:

```
# @author: Yxinmiracle
```

```
# @project: celeryke
# @file: redis_back.py
# @time: 2020/4/25 18:20

import redis
import json

redis_data = redis.Redis(host="127.0.0.1", port=6379, db=4) # 在redis的4号数据库取数据
keys = redis_data.keys()

for key in keys:
    print(key) # 之所以写那么多print是想让你们看清楚为什么这么取
    res = redis_data.get(key)
    print(res)
    res = json.loads(res.decode("utf8"))
    print(res)
    result = res["result"]
    print(result)
```

我们来仔细研究一下输出：

```
b'celery-task-meta-9f3e1bba-79d3-4e21-86d2-f59335ad9188'
b'{"status": "SUCCESS", "result": "comment 5", "traceback": null, "children": [],
"date_done": "2020-04-25T12:16:13.582702", "task_id": "9f3e1bba-79d3-4e21-86d2-f59335ad9188"}'
{'status': 'SUCCESS', 'result': 'comment 5', 'traceback': None, 'children': [],
'date_done': '2020-04-25T12:16:13.582702', 'task_id': '9f3e1bba-79d3-4e21-86d2-f59335ad9188'}
comment 5

b'celery-task-meta-9e27b2c1-073b-4970-89f9-16ab98b1ab6b' # 这一行来自于print(key),
拿到了一个byte类型的数据，很明显这个数据就是任务ID
b'{"status": "SUCCESS", "result": "comment 7", "traceback": null, "children": [],
"date_done": "2020-04-25T12:16:13.590737", "task_id": "9e27b2c1-073b-4970-89f9-16ab98b1ab6b"}' # 这一行来自于第一个print(res)，我们从数据库中get出来，就是一个byte类型的数据了。
{'status': 'SUCCESS', 'result': 'comment 7', 'traceback': None, 'children': [],
'date_done': '2020-04-25T12:16:13.590737', 'task_id': '9e27b2c1-073b-4970-89f9-16ab98b1ab6b'} # 这一行来自于第二个print(res)，将上面的byte类型的数据解码一下，就可以拿到这个字典取值了，很明显看见"result"就是我们想要的答案
comment 7 # 来自于print(result)取得答案
....
# 后面就不放了，都是一样的
```

回顾总结

看到这里，celery的基础内容就讲完了，我们现在来回顾一下，我们今天创建了三个py文件

```
main --> 运行爬虫主程序，将不耗时的代码放在里面
getSome --> 里面放的是很消耗时间的程序，我们用celery执行，这不仅是与main隔离了开来，而且任务本身还是异步执行的，这会大大的提高我们的爬虫效率
redis_back --> 这里是将我们任务放在缓存里面的数据存储下来，我们通过特定的程序去将结果取出来
```

京东爬虫

京东爬虫也是一个较为简单的爬虫，一般的初学者应该是有问题的，本次的这个项目主要联系celery去爬取京东的评论，因为这是耗时的操作，正好我们可以用celery去解决，我会将代码放在github上，如果提前想要的可以私聊我我发给你线程池+celery+mysql+redis的爬取京东全站的代码，大家一起交流学习，大家也可以拿京东练练手，本次的讲解就到这，谢谢大家观看，如果有错的话请私聊我。