

# Celery 分布式的任务队列

## 与rabbitmq消息队列的区别与联系：

- rabbitmq 调度的是消息，而Celery调度的是任务。
- Celery调度任务时，需要传递参数信息，传输载体可以选择rabbitmq。
- 利用rabbitmq的持久化和ack特性，Celery可以保证任务的可靠性。

## 优点：

- 轻松构建分布式的Service Provider,提供服务。
- 高可扩展性，增加worker也就是增加了队列的consumer。
- 可靠性，利用消息队列的durable和ack，可以尽可能降低消息丢失的概率，当worker崩溃后，未处理的消息会重新进入消费队列。
- 用户友好，利用flower提供的管理工具可以轻松的管理worker。

The screenshot shows the Celery Flower web interface. At the top is a green navigation bar with links: Celery Flower, Dashboard, Tasks, Broker, Monitor, Docs, and Code. Below the navigation bar, the worker name 'celery@brianyang-Latitude-E5440' is displayed. A tabbed interface shows 'Pool' as the active tab. The main content area is divided into two sections. On the left, under 'Worker pool options', there is a table with the following data:

Processes	6476, 6477
Max concurrency	2
Timeouts	0, 0
Writes	{u'raw': u'', u'all': u'', u'total': 0, u'avg': u'0.00%', u'inqueues': {u'active': 0, u'total': 2}}
Put guarded by semaphore	False
Max tasks per child	N/A
Worker PID	6445
Prefetch Count	2

On the right, under 'Pool size control', there is a 'Pool size' dropdown menu set to '1', with 'Grow' and 'Shrink' buttons. Below this are 'Min/Max autoscale' input fields and an 'Apply' button. A 'Refresh' button is located in the top right corner of the main content area.

flower

- 使用tornado-celery,结合tornado异步非阻塞结构，可以提高吞吐量，轻松创建分布式服务框架。
- 学习成本低，可快速入门

## 快速入门

### 定义一个celery实例main.py:

```
from celery import Celery
app = Celery('route_check', include=['check_worker_path'],
            broker='amqp://user:password@rabbitmq_host:port//')
app.config_from_object('celeryconfig')
```

include指的是需要celery扫描是否有任务定义的模块路径。例如`add_task` 就是扫描`add_task.py`中的任务

celery的配置文件可以从文件、模块中读取，这里是从模块中读取，`celeryconfig.py`为：

```
from multiprocessing import cpu_count

from celery import platforms
from kombu import Exchange, Queue

CELERYD_POOL_RESTARTS = False
CELERY_RESULT_BACKEND = 'redis://:password@redis_host:port/db'
CELERY_QUEUES = (
    Queue('default', Exchange('default'), routing_key='default'),
    Queue('common_check', Exchange('route_check'), routing_key='common_check'),
    Queue('route_check', Exchange('route_check'), routing_key='route_check',
delivery_mode=2),
    Queue('route_check_ignore_result', Exchange('route_check'),
routing_key='route_check_ignore_result',
        delivery_mode=2)
)
CELERY_ROUTES = {
    'route_check_task.check_worker.common_check': {'queue': 'common_check'},
    'route_check_task.check_worker.check': {'queue': 'route_check'},
    'route_check_task.check_worker.check_ignore_result': {'queue':
'route_check_ignore_result'}
}
CELERY_DEFAULT_QUEUE = 'default'
CELERY_DEFAULT_EXCHANGE = 'default'
CELERY_DEFAULT_EXCHANGE_TYPE = 'direct'
CELERY_DEFAULT_ROUTING_KEY = 'default'
# CELERY_MESSAGE_COMPRESSION = 'gzip'
CELERY_ACKS_LATE = True
CELERYD_PREFETCH_MULTIPLIER = 1
CELERY_DISABLE_RATE_LIMITS = True
CELERY_TIMEZONE = 'Asia/Shanghai'
CELERY_ENABLE_UTC = True
CELERYD_CONCURRENCY = cpu_count() / 2
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TASK_PUBLISH_RETRY = True
CELERY_TASK_PUBLISH_RETRY_POLICY = {
    'max_retries': 3,
```

```

        'interval_start': 10,
        'interval_step': 5,
        'interval_max': 20
    }
platforms.C_FORCE_ROOT = True

```

这里面是一些celery的[配置参数](#)。

在上面include的add\_task.py定义如下：

```

#encoding:utf8

from main import app

@app.task
def add(x,y):
    return x+y

```

## 启动celery

```
celery -A main worker -l info -Ofair
```

- -A 后面是包含celery定义的模块,我们在main.py中定义了 `app = Celery...`

## 测试celery:

- -l 日志打印的级别，这里是info
- -[Ofair](#) 这个参数可以让Celery更好的调度任务

```

# encoding:utf8
__author__ = 'brianyang'

import add_task

result = add_task.add.apply_async((1,2))
print type(result)
print result.ready()
print result.get()
print result.ready()

```

## 输出是

```

<class 'celery.result.AsyncResult'>
False
3
True

```

当调用result.get()时，如果还没有返回结果，将会阻塞直到结果返回。这里需要注意的是，如果需要返回worker执行的结果，必须在之前的config中配置

`CELERY_RESULT_BACKEND`这个参数，一般推荐使用Redis来保存执行结果，如果不关

心worker执行结果，设置**CELERY\_IGNORE\_RESULT=True**就可以了，关闭缓存结果可以提高程序的执行速度。

在上面的测试程序中，如果修改为：

```
# encoding:utf8
__author__ = 'brianyang'
```

```
import add_task

result = add_task.add.(1,2)
print type(result)
print result
```

输出结果为：

```
<type 'int'>
3
```

相当于直接本地调用了add方法，并没有走Celery的调度。  
通过flower的dashbord可以方便的监控任务的执行情况：

Celery FlowerDashboardTasksBrokerMonitorDocsCode

Show 10 entriesSearch:

Name	UUID	State	args	kwargs	Result	Received	Started	Worker
add_task.add	2ed0870b-2cd2-4ad7-ac7a-42e88e3e014a	SUCCESS	[1, 2]	{}	'3'	2016-07-07 14:54:00.330	2016-07-07 14:54:00.334	celery@brianyang-Latitude-E5440
add_task.add	9d03850c-69a6-4542-93f6-86b34ba22ede	SUCCESS	[1, 2]	{}	'3'	2016-07-07 14:53:32.959	2016-07-07 14:53:32.963	celery@brianyang-Latitude-E5440
test_task.add	f9131f1e-2310-4326-86a9-935938786b8f	SUCCESS	[1, 2]	{}	'3'	2016-07-07 14:48:30.549	2016-07-07 14:48:30.595	celery@brianyang-Latitude-E5440

Showing 1 to 3 of 3 entriesPrevious1Next

task list

Celery FlowerDashboardTasksBrokerMonitor

add\_task.add 2ed0870b-2cd2-4ad7-ac7a-42e88e3e014a

Basic task options	
Name	add_task.add
UUID	2ed0870b-2cd2-4ad7-ac7a-42e88e3e014a
State	SUCCESS
args	[1, 2]
kwargs	{}
Result	'3'

Advanced task options	
Received	2016-07-07 14:54:00.330362 CST
Started	2016-07-07 14:54:00.334649 CST
Succeeded	2016-07-07 14:54:00.353423 CST
Retries	0
Worker	celery@brianyang-Latitude-E5440
Timestamp	2016-07-07 14:54:00.353423 CST
Runtime	0.0204996750108
Clock	139

task detail

还可以对worker进行重启，关闭之类的操作

Celery Flower

DashboardTasksBrokerMonitor

DocsCode

Active: 0

Processed: 3

Failed: 0

Succeeded: 3

Retried: 0

Shut Down

Shut Down

Restart Pool

Refresh

Search:

	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@brianyang-Latitude-E5440	Online	0	3	0	3	0	1.39, 1.46, 1.32

Showing 1 to 1 of 1 entries

使用Celery将一个集中式的系统拆分为分布式的系统大概步骤就是:

- 根据功能将耗时的模块拆分出来, 通过注解的形式让Celery管理
- 为拆分的模块设置独立的消息队列
- 调用者导入需要的模块或方法, 使用`apply_async`进行异步的调用并根据需求关注结果。
- 根据性能需要可以添加机器或增加worker数量, 方便弹性管理。

需要注意的是:

- 尽量为不同的task分配不同的queue,避免多个功能的请求堆积在同一个queue中。
- `celery -A main worker -l info -Ofair -Q add_queue`启动Celery时, 可以通过参数Q加queue\_name来指定该worker只接受指定queue中的tasks.这样可以使不同的worker各司其职。
- `CELERY_ACKS_LATE`可以让你的Celery更加可靠, 只有当worker执行完任务后, 才会告诉MQ, 消息被消费。
- `CELERY_DISABLE_RATE_LIMITS` Celery可以对任务消费的速率进行限制, 如果你没有这个需求, 就关闭掉它吧, 有益于会加速你的程序。

## tornado-celery

tornado应该是python中最有名的异步非阻塞模型的web框架, 它使用的是单进程轮询的方式处理用户请求, 通过epoll来关注文件状态的改变, 只扫描文件状态符发生变化的FD(文件描述符)。

由于tornado是单进程轮询模型，那么就不适合在接口请求后进行长时间的耗时操作，而是应该接收到请求后，将请求交给背后的worker去干，干完活儿后在通过修改FD告诉tornado我干完了，结果拿走吧。很明显，Celery与tornado很般配，而tornado-celery是celery官方推荐的结合两者的一个模块。整合两者很容易，首先需要安装：

- tornado-celery
- tornado-redis

tornado代码如下：

```
# encoding:utf8
__author__ = 'brianyang'

import tcclery
import tornado.gen
import tornado.web

from main import app
import add_task

tcclery.setup_nonblocking_producer(celery_app=app)

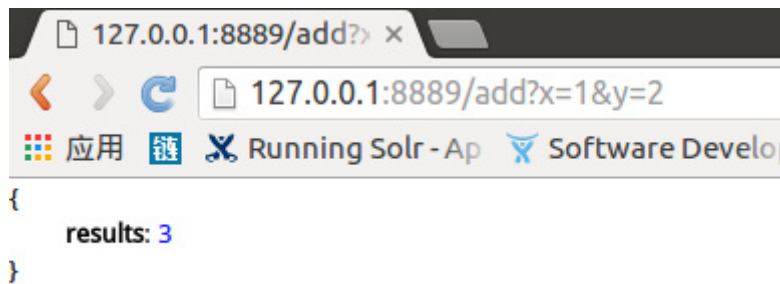
class CheckHandler(tornado.web.RequestHandler):
    @tornado.web.asynchronous
    @tornado.gen.coroutine
    def get(self):
        x = int(self.get_argument('x', '0'))
        y = int(self.get_argument('y', '0'))
        response = yield tornado.gen.Task(add_task.add.apply_async, args=[x, y])
        self.write({'results': response.result})
        self.finish

application = tornado.web.Application([
    (r"/add", CheckHandler),
])

if __name__ == "__main__":
    application.listen(8889)
    tornado.ioloop.IOLoop.instance().start()
```

在浏览器输入：<http://127.0.0.1:8889/add?x=1&y=2>

结果为：



通过tornado+Celery可以显著的提高系统的吞吐量。

## Benchmark

使用Jmeter进行压测，60个进程不间断地的访问服务器：

接口单独访问响应时间一般在200~400ms

- uwsgi + Flask方案：

uwsgi关键配置：

```
processes = 10
threads = 3
```

Flask负责接受并处理请求，压测结果：

qps是46,吞吐量大概是2700/min

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
HTTP请求	2364	1266	1272	1407	1458	1585	430	2226	0.00%	46.1/sec	530.5
总体	2364	1266	1272	1407	1458	1585	430	2226	0.00%	46.1/sec	530.5

uwsgi+Flask

- tornado+Celery方案：

Celery配置：

`CELERYD_CONCURRENCY = 10`也就是10个worker(进程),压测结果：

qps是139,吞吐量大概是8300/min

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
HTTP请求	7938	394	386	486	592	677	69	6575	0.00%	138.1/sec	2757.6
总体	7938	394	386	486	592	677	69	6575	0.00%	138.1/sec	2757.6

tornado+Celery

从吞吐量和接口相应时间各方面来看，使用tornado+Celery都能带来更好的性能。

## Supervisor

- 什么是supervisor

supervisor俗称Linux后台进程管理器

- 适合场景

-- 需要长期运行程序，除了nohup，我们有更好的supervisor  
-- 程序意外挂掉，需要重启，让supervisor来帮忙  
-- 远程管理程序，不想登陆服务器，来来来，supervisor提供了高大上的web操作界面。

之前启动Celery命令是`celery -A main worker -l info -Ofair -Q common_check`，当你有10台机器的时候，每次更新代码后，都需要登陆服务器，然后更新代码，最后再杀掉Celery进程重启，恶不恶心，简直恶心死了。

让supervisor来，首先需要安装：

```
pip install supervisor
```

配置文件示例：

```
[unix_http_server]
file=/tmp/supervisor.sock ; path to your socket file
chmod=0777
username=admin
password=admin

[inet_http_server]
port=0.0.0.0:2345
username=admin
password=admin

[supervisord]
logfile=/var/log/supervisord.log ; supervisord log file
logfile_maxbytes=50MB ; maximum size of logfile before rotation
logfile_backups=10 ; number of backed up logfiles
loglevel=info ; info, debug, warn, trace
pidfile=/var/run/supervisord.pid ; pidfile location
nodaemon=false ; run supervisord as a daemon
minfds=1024 ; number of startup file descriptors
minprocs=200 ; number of process descriptors
user=root ; default user
childlogdir=/var/log/ ; where child log files will live

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface
```



```
[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use unix:/// schem for a unix sockets.
username=admin
password=admin
[program:celery]
command=celery -A main worker -l info -Ofair

directory=/home/q/celeryTest
user=root
numprocs=1
stdout_logfile=/var/log/worker.log
stderr_logfile=/var/log/worker.log
autostart=true
autorestart=true
startsecs=10

; Need to wait for currently executing tasks to finish at shutdown.
; Increase this if you have very long running tasks.
stopwaitsecs = 10

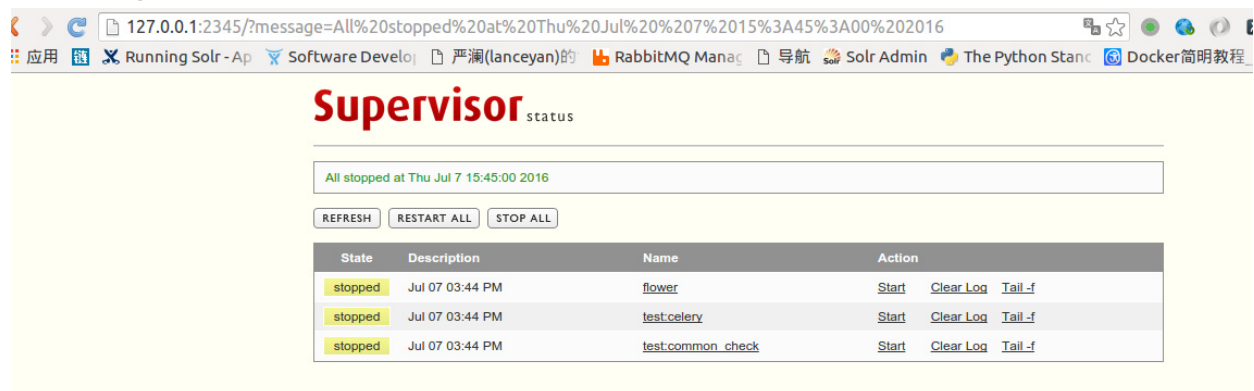
; When resorting to send SIGKILL to the program to terminate it
; send SIGKILL to its whole process group instead,
; taking care of its children as well.
killasgroup=true

; Set Celery priority higher than default (999)
; so, if rabbitmq is supervised, it will start first.
priority=1000
```

示例文件很长，不要怕，只需要复制下来，改改就可以  
比较关键的几个地方是：

```
[inet_http_server]
port=0.0.0.0:2345
username=admin
password=admin
```

这个可以让你通过访问<http://yourhost:2345>，验证输入admin/admin的方式远程管理supervisor,效果如下：



The screenshot shows the Supervisor web interface at <http://127.0.0.1:2345/>. The status bar indicates "All stopped at Thu Jul 7 15:45:00 2016". Below this are buttons for "REFRESH", "RESTART ALL", and "STOP ALL". A table lists the supervised processes:

State	Description	Name	Action
stopped	Jul 07 03:44 PM	flower	Start Clear Log Tail -f
stopped	Jul 07 03:44 PM	test:celery	Start Clear Log Tail -f
stopped	Jul 07 03:44 PM	test:common_check	Start Clear Log Tail -f

remote supervisor

[program:flower]这里就是你要托管给supervisor的程序的一些配置,其中 autorestart=true 可以在程序崩溃时自动重启进程,不信你用kill试试看。剩下的部分就是一些日志位置的设置,当前工作目录设置等, so easy~ supervisor优点:

- 管理进程简单,再也不用nohup & kill了。
- 再也不用担心程序挂掉了
- web管理很方便

缺点:

- web管理虽然方便,但是每个页面只能管理本机的supervisor,如果我有一百台机器,那就需要打开100个管理页面,太麻烦了。

怎么办~

## supervisor-easy闪亮登场

通过rpc调用获取配置中的每一个supervisor程序的状态并进行管理,可以分组,分机器进行批量/单个的管理。方便的不要不要的。来两张截图:

- 分组管理:

Group List

Server List

Group List custom by user

group : celery			Batch start	Batch stop	Batch restart
state	app	location	description		operation
STOPPED	celery	admin@xxx1.com:2345	Jul 07 03:44 PM		<a href="#">start</a> <a href="#">restart</a> <a href="#">stop</a> <a href="#">tail</a>
STOPPED	celery	admin@xxx2.com:2345	Jul 07 11:39 AM		<a href="#">start</a> <a href="#">restart</a> <a href="#">stop</a> <a href="#">tail</a>
group : flower			Batch start	Batch stop	Batch restart
state	app	location	description		operation
RUNNING	flower	admin@xxx1.com:2345	pid 15882, uptime 0:26:16		<a href="#">start</a> <a href="#">restart</a> <a href="#">stop</a> <a href="#">tail</a>

group

- 分机器管理:

Servers List

controlled by supervisor

admin@xxx2.com:2345

admin@xxx1.com:2345

admin@xxx2.com:2345

Batch start

Batch stop

Batch restart

state	app	location	description	operation
STOPPED	flower	admin@xxx2.com:2345	Jul 07 11:39 AM	<div>start</div> <div>restart</div> <div>stop</div> <div>tail</div>
STOPPED	celery	admin@xxx2.com:2345	Jul 07 11:39 AM	<div>start</div> <div>restart</div> <div>stop</div> <div>tail</div>
STOPPED	common_check	admin@xxx2.com:2345	Jul 07 11:39 AM	<div>start</div> <div>restart</div> <div>stop</div> <div>tail</div>

server

通过简单的配置，可以方便的进行管理。