



27 августа 2010 в 09:35

Celery — распределенная очередь заданий

[Блог компании Бигго](#)

На этот раз мы решили рассказать о замечательном продукте, который мы используем в нашей работе. Речь пойдет о Celery — «distributed task queue». Это распределенная асинхронная очередь заданий, которая обладает широким функционалом. В нашем [конструкторе сайтов](#) нам часто приходится запускать асинхронные с точки зрения ответа пользователю задачи. На хабре, к сожалению, не много информации по данному продукту, а он заслуживает отдельного упоминания, это мы и хотим исправить.

И так, что же умеет [Celery](#):

- Выполнять задания асинхронно или синхронно
- Выполнять периодические задания(умная замена crond)
- Выполнять отложенные задания
- Распределенное выполнение (может быть запущен на N серверах)
- В пределах одного worker'a возможно конкурентное выполнение нескольких задач(одновременно)
- Выполнять задание повторно, если вылез exception
- Ограничивать количество заданий в единицу времени(rate limit, для задания или глобально)
- Routing заданий(какому worker'у что делать)
- Несложно мониторить выполнение заданий
- Выполнять подзадания
- Присылать отчеты об exception'ах на email
- Проверять выполнилось ли задание(удобно для построения Ajax приложений, где юзер ждет факта завершения)

Заинтересовало? Просим под кат.

Картинка с официального сайта:



Начнем с конфигурации worker'a. Это демон, который собственно получает задания из очереди и выполняет их. Рекомендуемая очередь — RabbitMQ, но мы пока ограничились ghettoq, через MongoDB. Также поддерживается Redis и PCУБД.

celeryconfig.py:

```
CARROT_BACKEND = "ghettoq.taproot.MongoDB"

BROKER_HOST = "xxx"
BROKER_PORT = 27017
BROKER_VHOST = "celery"

CELERY_SEND_TASK_ERROR_EMAILS = True
ADMINS = ( ('Admin', 'admin@localhost'), )

CELERYD_MAX_TASKS_PER_CHILD = 5

CELERY_IMPORTS = ("tasks", )
CELERY_DISABLE_RATE_LIMITS = True

CELERY_RESULT_BACKEND = "mongodb"
CELERY_MONGODB_BACKEND_SETTINGS = {
    "host": "xxx",
    "port": 27017,
    "database": "celery",
    "taskmeta_collection": "my_taskmeta_collection",
}
```

Запуск демона: `celeryd -I INFO -B`

Включаем логгирование в консоль и опция -B запуск демона периодических заданий. Последний можно запустить отдельно командой `celerybeat`

Теперь создадим тестовое задание. В конфиге мы импортируем tasks, поэтому и файл заданий у нас tasks.py:

```
from celery.decorators import task
from celery.decorators import periodic_task
from celery.task.schedules import crontab

@periodic_task(run_every=timedelta(seconds=60))
def mail_queue():

    print "Task is executed every minute"

@periodic_task(run_every=crontab(hour=0, minute=10))
def transactions():
    print "Task is executed every day on 0:10"

@task
def delayed_function(id):
    some_function()

@task
def delayed_heavy_function(id):
    some_heavy_function()
```

Итак, у нас 4 задания в tasks. Первые два выполняются по расписанию, т.к. они отмечены декоратором @periodic_task. А вот два последних будут вызваны непосредственно из кода программы. Вот таким образом:

```
from tasks import delayed_function, delayed_heavy_function

delayed_function.apply_async(args=[id], countdown=300) # Будет запущена через 300 секунд

r = delayed_heavy_function.delay(id) # Будет запущена сразу(как только появится возможность), в асинхронном режи
```

Теперь для того чтобы отследить результат и факт завершения последнего задания выполним:

```
r.ready() # Вернет True если задание отработало
r.result # Вернет значение выполненной функции или None если еще не выполнено(асинхронно)
r.get() # Будет ждать выполнения задания и вернет ее результат(синхронно)
```

Переменную r можно прогнать через cPickle, положить значение в кеш и аяксом опрашивать статус задания. Либо можно получить task id, и положить в кеш его. Кроме того, task id вы можете задавать самостоятельно, главное чтоб он был уникальным.

После плотного использования celery мы обнаружили несколько ошибок, связанных с отложенным выполнением задач, с менеджером очередей ghettoq, но они все были поправлены автором в день создания issue на github, за что ему спасибо.

Не так давно вышла версия 2.0, которая перестала быть django-зависимой, а интеграция с django теперь вынесена в отдельный подпроект celery-django.

Из ограничений celery можно выделить два, точнее это просто особенности: на штатной FreeBSD worker'ы не будут работать, т.к. там нет питонового multiprocessing, хотя в сети есть рецепты по сборке ядра для celery; для перегрузки заданий необходимо рестартовать воркер, чтобы он загрузил новый python-код заданий и связанных функций. На linux работает замечательно.

[celery](#), [task queue](#), [очередь заданий](#), [mongodb](#)

+8 15165 70 Biggo -1,0

комментарии (25) отслеживать новые: ☐ в почте ☐ в трежере



Zada, 27 августа 2010 в 11:29 #

0

А тер mongodb, зачем?

 **Biggo**, 27 августа 2010 в 11:46 # ↑ +1

Так бекенд — mongodb в примере.

 **Zada**, 27 августа 2010 в 11:47 # ↑ 0


Ага, как-то пропустил. Это несомненно круто.
Спасибо.

 **mechmind**, 30 августа 2010 в 05:57 # 0

А в чем заключается распределенность очереди заданий? Пока что я вижу централизованный учет и раздачу их.

 **Biggo**, 30 августа 2010 в 09:12 # ↑ 0

Распределенность в плане выполнения заданий, то есть они делятся между запущенными воркерами, однако если вы хотите чтобы раздача заданий была отказоустойчивой, есть смысл использовать [RabbitMQ](#)

 **Tonik**, 30 августа 2010 в 07:56 # 0

Не флейму для, а интереса ради, а [gearman.org/](#) не рассматривали? Вроде как значительно более стабильный и проверенный вариант

 **Biggo**, 30 августа 2010 в 09:22 # ↑ 0

Нет, не рассматривали. В будущем планируем перевести раздачу очередей на AMQP(RabbitMQ), а это тоже промышленное решение.

 **greevex**, 12 сентября 2013 в 17:45 # ↑ 0

Instagram перешел с gearmand на celery.

Мы, кстати, у себя гирман используем очень и очень много где и тоже подумываем переходить, привысоких нагрузках гирман начинает подтупливать, а именно выполняя одну очередь, забывает на другие полностью, и даже приоритеты задач не помогают.

 **serio**, 30 августа 2010 в 09:02 # 0

офтопик:


интересно, сколько человек тянут такой проект?

 **serio**, 30 августа 2010 в 09:03 # ↑ 0

это я про конструктор сайтов

 **Biggo**, 30 августа 2010 в 09:25 # ↑ 0

4 пока

 **immaculate**, 30 августа 2010 в 16:23 # ↑ 0

Четыре программиста или админ, дизайнер, программист, менеджер?

 **Biggo**, 30 августа 2010 в 17:00 # ↑ 0


всего 4, так что второе.

 **mythmaker**, 30 августа 2010 в 09:15 # 0

Есть ли математические применения этой штуки? Например, если у меня есть кластер и суперкомпьютер, могу я что-то для себя сложное на ней посчитать?

 **Biggo**, 30 августа 2010 в 09:24 # ↑ 0

В принципе почему бы нет, вопрос в том что это даст в плане выигрыша в производительности. Надо смотреть особенности задачи.

 **professor_kuvalda**, 30 августа 2010 в 09:37 # +1

> `delayed_function.apply_async(args=[id], countdown=300)` # Будет запущена через 300 секунд

как-нибудь проверить можно, есть ли данное задание в очереди на выполнение? отменить его можно?

 **Biggo**, 30 августа 2010 в 09:55 # ↑ 0

Да, можно:

```
r = delayed_function.apply_async(args=[id], countdown=300)
r.status — Если PENDING, значит еще ждет
```

Отменить можно:

```
celery.task.control.revoke(task_id, destination=None, **kwargs)
```



[professor_kuvalda](#), 30 августа 2010 в 10:56 #



+1

благодарю



[insa](#), 30 августа 2010 в 12:54 #

0

Очередной RabbitMQ ради RabbitMQ.



[TravisBickle](#), 30 августа 2010 в 13:27 #

0

Могли поюзать tailable cursors в MongoDB для очередей =) Было прикольнее чем RabbitMQ.



[voihi](#), 30 августа 2010 в 13:35 #

0

Переменную r можно прогнать через cPickle, положить значение в кеш и аяксом опрашивать статус задания.

Ой-ой-ой-ой!.. не надо такие опасности советовать людям! Уточняйте четко, что значение cPickle надо хранить только на стороне сервера, а опрашивать аяксом только по какому-нибудь связанному идентификатору, а то ведь люди будут передавать клиенту результат cPickle и по нему опрашивать. А там:

docs.python.org/library/pickle.html

Warning: The `pickle` module is not intended to be secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.



[Biggo](#), 30 августа 2010 в 13:55 #



0

Да, конечно лучше в кеш только id класть, а вообще по ситуации смотреть надо.



[bobry](#), 30 августа 2010 в 21:56 #

0

спасибо, еще один перевод tutorial'a на хабре

я думал роман воружин исчерпал тему введений на русском

НЛО прилетело и опубликовало эту надпись здесь

НЛО прилетело и опубликовало эту надпись здесь