

11 июля 2011 в 22:22

Асинхронные задания в Django с помощью Celery

[Django](#)*

Приветствую!

Думаю, большинство разработчиков **Django** слышали о [Celery](#) — системе асинхронного выполнения заданий, а многие даже активно её используют.

Около года назад на хабре была довольная [хорошая статья](#), рассказывающая о том, как использовать Celery. Однако, как было упомянуто в заключении, уже вышла Celery 2.0 (на данный момент стабильной версией является 2.2.7), где интеграция с django была вынесена в отдельный пакет, а также произошли [другие изменения](#).

Данная статья будет полезна прежде всего новичкам, которые начинают работать с Django, и им требуется что-то, способное выполнять асинхронные и/или периодические задания в системе (например очистку устаревших сессий). Я покажу как установить и настроить Celery для работы с Django от начала до конца, а также расскажу про некоторые другие полезные настройки и подводные камни.

Прежде всего проверим наличие в системе пакета **python-setuptools**, и установим его в случае отсутствия:

```
aptitude install python-setuptools
```

Установка Celery

Сама **Celery** устанавливается очень просто:

```
easy_install Celery
```

Подробнее в оригинале: <http://celeryq.org/docs/getting-started/introduction.html#installation>

В статье, ссылка на которую дана в начале, в качестве бэкенда использовалась **MongoDB**, здесь я покажу как в качестве бэкенда и брокера сообщений использовать ту же самую БД, в которой хранят данные остальные приложения Django.

django-celery

Устанавливаем пакет [django-celery](#):

```
easy_install django-celery
```

Как уже было сказано, **django-celery** предоставляет удобную интеграцию Celery и Django. В частности он использует **Django ORM** как бэкенд для сохранения результатов выполнения заданий Celery, а также автоматически находит и регистрирует задания Celery для приложений Django, перечисленных в **INSTALLED_APPS**.

После установки django-celery нужно сконфигурировать:

- добавить **djcelery** в список **INSTALLED_APPS**:

```
INSTALLED_APPS += ("djcelery", )
```

- добавить следующие строки в файл настроек django `{{settings.py}}`:

```
import djcelery
djcelery.setup_loader()
```

- Создать необходимые таблицы в БД:

```
./manage.py syncdb
```

- Задаём БД в качестве места хранения периодических заданий, добавляем в **settings.py**:

```
CELERYBEAT_SCHEDULER = "djcelery.schedulers.DatabaseScheduler"
```

С помощью этой опции мы сможем добавлять/удалять/редактировать периодические задания через админку django.

При использовании `mod_wsgi` добавить следующие строки в конфигурационный файл WSGI:

```
import os
os.environ["CELERY_LOADER"] = "django"
```

django-kombu

Теперь нам осталось найти подходящего брокера сообщений (message broker) для Celery, в этой статье я буду использовать [django-kombu](#) — пакет, позволяющий использовать базу данных Django в качестве хранилища сообщений (message store) для [Kombu](#) (реализация AMQP на питоне).

Устанавливаем пакет:

```
easy_install django-kombu
```

Настраиваем:

- добавляем `django-kombu` в список **INSTALLED_APPS**:

```
INSTALLED_APPS += ("django-kombu",)
```

- Задаём `django-kombu` в качестве брокера в **settings.py**:

```
BROKER_BACKEND = "django-kombu.transport.DatabaseTransport"
```

- Создаём необходимые таблицы в БД:

```
./manage.py syncdb
```

Запускаем

Запускаем процессы `celery` и `celerybeat`:

(Без `celerybeat` можно запускать и выполнять обычные (regular) задания. Для выполнения периодических заданий по расписанию необходим запуск `celerybeat`)

- В `linux` оба процесса можно запустить одновременно с помощью ключа **-B**:

```
# ./manage.py celeryd -B
----- celery@test v2.2.7
---- *** ----
--- * * * * -- [Configuration]
-- * - *** --- . broker: django-kombu.transport.DatabaseTransport://guest@localhost0/
- ** ----- . loader: djcelery.loaders.DjangoLoader
- ** ----- . logfile: [stderr]@WARNING
- ** ----- . concurrency: 16
- ** ----- . events: OFF
- *** --- * --- . beat: ON
-- * * * * * ---
--- * * * * * --- [Queues]
----- . celery: exchange:celery (direct) binding:celery
```

- В `windows` `celery` и `celerybeat` необходимо запускать отдельно:

```
./manage.py celeryd --settings=settings
./manage.py celerybeat
```

Опция **--settings** может потребоваться, если возникает следующее исключение:

```
ImportError: Could not import settings 'app_name.settings' (Is it on sys.path?): No module named app_name.settings
```

Подробнее о проблеме: http://groups.google.com/group/celery-users/browse_thread/thread/43a95be6865a636/d91ab2492885f3d4?lnk=gst&q=settings#d91ab2492885f3d4
 Полный список известных проблем с `celery` на `Windows`: <http://celeryproject.org/docs/faq.html#windows>

После запуска можем посмотреть как выглядят периодические задания в админке `django`:

Django administration Welcome, admin. Change password / Log out

Home > Djangery > Periodic tasks

Select periodic task to change Add periodic task +

Action: Go 0 of 3 selected

<input type="checkbox"/>	Periodic task
<input type="checkbox"/>	users.tasks.clean_sessions: 0 0 * (m/h/d)
<input type="checkbox"/>	celery.backend_cleanup: 0 4 * (m/h/d)

2 periodic tasks

Если в качестве бэкенда celery использовать что-либо отличное от Django ORM (RabbitMQ например), то в админке Django можно было бы также просматривать состояние всех остальных заданий, выглядит примерно так:

Django administration Welcome, askadmin. Change password / Log out

Home > Djangery > Tasks

Tasks

2010

Action: Go 0 of 172 selected

<input type="checkbox"/>	UUID	State	Name	Args	Kwargs	ETA	Worker
<input type="checkbox"/>	5456f8ea-f8c8-4778-91b1-a468abc4f91f	SUCCESS	[.]statuses_twitter_update	('e97c687d13d87f614a6d81768d8f3d8e', '96576157-e6ca-48d1-b8d4-8b...	{}	None	h9
<input type="checkbox"/>	b6fa561c-8f8a-4e39-9894-58403a21fb81	SUCCESS	portaloperacom[.]refresh_portal_blog	()	{}	None	h10
<input type="checkbox"/>	ef256274-7a73-4834-90fb-03f89e15aca6	SUCCESS	[.]statuses_twitter_update	('e97c687d13d87f614a6d81768d8f3d8e', '96576157-e6ca-48d1-b8d4-8b...	{}	None	h9
<input type="checkbox"/>	3359ba9e-7387-4d8f-ba7e-1fb6269ba396	SUCCESS	d[.]refresh_feed	()	{'feed_id': 231L, 'feed_url': u'http://www.cnbc.com/id/19746125/...	None	h6
<input type="checkbox"/>	51681ec8-842b-46e4-9998-7632460b40b3	SUCCESS	[.]statuses_twitter_update	('e97c687d13d87f614a6d81768d8f3d8e', '96576157-e6ca-48d1-b8d4-8b...	{}	None	h10
<input type="checkbox"/>	ce47e922-c76c-45aa-8885-c2d853e05dbb	SUCCESS	[.]statuses_twitter_update	('e97c687d13d87f614a6d81768d8f3d8e', '96576157-e6ca-48d1-b8d4-8b...	{}	None	h9
<input type="checkbox"/>	2ebb8254-6891-4b6c-a0e7-	SUCCESS	[.]statuses_twitter_update	('e97c687d13d87f614a6d81768d8f3d8e', '96576157-e6ca-48d1-b8d4-8b...	{}	None	h10

Filter
By state
 All
 RETRY
 REVOKED
 SUCCESS
 STARTED
 FAILURE
 PENDING
By name
 All
 djangofeeds.tasks.refresh_feec
 opalfmevents.lastfm_events_up
 opaltweets.statuses_twitter_up
 portaloperacom.tasks.refresh_j
By event received at
 Any date
 Today
 Past 7 days
 This month
 This year
By ETA
 Any date
 Today
 Past 7 days
 This month
 This year
By worker
 All
 h10
 h8
 h6

Подробнее: <http://stackoverflow.com/questions/5449163/django-celery-admin-interface-showing-zero-tasks-workers>

UPDATE: дописываю немного про демонизацию, так как может получиться не с первого раза.

Запускаем celery в виде сервиса

Скачиваем скрипт запуска celery отсюда: <https://github.com/ask/celery/tree/master/contrib/generic-init.d/> и помещаем его в каталог **/etc/init.d** с соответствующими правами.

В каталоге **/etc/default** создаём файл **celeryd**, из него скрипт будет брать настройки запуска:

```
# Where the Django project is.
CELERYD_CHDIR="/var/www/myproject"

# Path to celeryd
CELERYD_MULTI="$CELERYD_CHDIR/manage.py celeryd_multi"

CELERYD_OPTS="--time-limit=300 --concurrency=8 -B"
CELERYD_LOG_FILE=/var/log/celery/%n.log

# Path to celerybeat
CELERYBEAT="$CELERYD_CHDIR/manage.py celerybeat"
CELERYBEAT_LOG_FILE="/var/log/celery/beat.log"
CELERYBEAT_PID_FILE="/var/run/celery/beat.pid"

CELERY_CONFIG_MODULE="settings"

export DJANGO_SETTINGS_MODULE="settings"
```

Опция **--concurrency** задаёт число процессов celery (по умолчанию это число равно количеству процессоров). После этого можно запустить celery с помощью **service**:

```
service celeryd start
```

Подробнее: docs.celeryproject.org/en/latest/tutorials/daemonizing.html#daemonizing

Работа с celery

После установки django-celery задания celery регистрируются автоматически из всех модулей tasks.py из всех приложений, перечисленных в **INSTALLED_APPS**. Помимо модулей tasks можно также задать дополнительные модули с помощью параметра **CELERY_IMPORTS**:

```
CELERY_IMPORTS=('myapp.my_task_module',)
```

Также полезно активировать опцию **CELERY_SEND_TASK_ERROR_EMAILS**, благодаря которой Celery будет уведомлять обо всех ошибках по адресам, перечисленным в переменной **ADMINS**.

Написание заданий для celery практически не изменилось со времён предыдущей статьи:

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

@periodic_task(ignore_result=True, run_every=crontab(hour=0, minute=0))
def clean_sessions():
    Session.objects.filter(expire_date__lt=datetime.now()).delete()
```

Единственное отличие — декораторы теперь следует импортировать из **celery.task**, модуль **decorators** стал deprecated.

Пара замечаний о производительности:

- Если задание не возвращает никакого результата, то лучше установить опцию **ignore_result=True**
- Выключить [ограничение скорости выполнения заданий \(Rate limits\)](#), если ваши задания не используют их:

```
CELERY_DISABLE_RATE_LIMITS = True
```

Подробнее об этих и других советах по Celery: <http://celeryproject.org/docs/userguide/tasks.html#tips-and-best-practices>

[django](#), [celery](#), [taskqueue](#)

+46

165

black_bunny ^{31,0}

комментарии (63) отслеживать новые: ☐ в почте ☐ в треке



[printf](#), 11 июля 2011 в 23:23 #

+2

Классная штука, каждый раз собираюсь попробовать.



[black_bunny](#), 11 июля 2011 в 23:25 #



+2

Вот и мануал удобный есть, я надеюсь.



[mktums](#), 12 июля 2011 в 00:09 #

+5

pony powered



[prudis](#), 12 июля 2011 в 02:36 #

0

А чем стандартный крон и management commands не устраивает?



[black_bunny](#), 12 июля 2011 в 05:59 #



0

Скрп может случить заменой периодических заданий celery, но используя celery вы получаете удобство задания расписания непосредственно внутри приложений django.

Про management commands не понял вопроса — ведь это команды типа **syncdb**, выполняемые из консоли. Асинхронные задания в celery — это когда вы, например, после выполнения запроса от пользователя, запускаете выполняться что-то ещё, что потребуете в дальнейшем, а пользователь тем временем уже видит отрендеренную страничку.

 [artem_dev](#), 12 июля 2011 в 08:42 #



0

можно писать свои management commands и запускать через cron\shell



[Outspector](#), 12 июля 2011 в 08:57 #



+1

Главная особенность celery вовсе не в том, что с ее помощью можно запускать периодические задания и использовать вместо cron-а (эта плюшка вообще приделана сбоку, и ее можно не использовать). Больше пользы в том, что долго выполняющееся действие можно легко отвязать от кода, который выполняется во view в ответ на запрос пользователя.

Фактически, используя celery, вы добавляете задание в очередь и на время забываете про него. Затем запущенный отдельно демон по имени celeryd это задание выполняет, а результат выполнения складывает обратно.



[solshark](#), 12 июля 2011 в 10:32 #



-5

Это лишняя сущность. Все то что Вы описали делается, как верно заметил artem_dev, связкой cron+management commands. Причем без установки кучи шлака.

Хотя конечно «перфекционистам с дедлайнами» подойдет, особенно тем кто «ниасилил» синтаксис crontab. Но я бы использовать не стал.



[black_bunny](#), 12 июля 2011 в 10:50 #



0

С другой стороны, то же самое можно сказать и про management commands. Зачем писать дополнительную management команду и запускать всё это через шелл если можно обойтись одной строчкой с декоратором?



[Outspector](#), 12 июля 2011 в 11:02 #



+6

Если не использовать сторонний «шлак», то этот шлак придется написать самому. Насколько я понял, асинхронные команды вы предлагаете использовать следующим образом (предполагаю, потому что сам так делал, и в некоторых случаях этот подход действительно хорошо работает):

- куда-нибудь в базу пишем команду, которую предполагается выполнить, или каким-то способом отмечаем «необработанные» объекты
- в cron добавляем задание, которое периодически (раз в минуту) проверяет список и выполняет действия с необработанными объектами.

Все работает нормально до тех пор, пока у вас 1-2 таких действия, они выполняются достаточно быстро, и задержка в 1-2 минуты с момента добавления задания до момента начала его исполнения вас устраивает. Если хотя бы одно из этих действий не выполняется, то наивный подход перестает работать, вы начинаете добавлять костыли, и вот уже незаметно для себя изобретаете велосипед.

- если задание не успевает выполниться за 1-2 минуты, то последующие задания начинают «наступать на пятки» уже выполняющемуся, и нужно придумывать блокировки. Либо на уровне задания cron (flock), либо на уровне записей в базе (вводить состояния задач: «не выполнялось», «выполняется», «выполнено»). В celery эта задача уже решена.
- если заданий у вас становится больше, чем 1-2, то либо для каждого задания (resize картинки, получение rss фида с внешнего сервера, загрузка товаров в базу и т.п.) вам нужно писать отдельную management команду, либо придумывать способ выделить общую часть, отвечающую за выполнение команд. В celery код, который непосредственно занимается работой, отделяется от кода, который занимается управлением очереди, с помощью декоратора task. Довольно элегантно, на коленке такое не скрутишь.
- если вам нужно отслеживать результат выполнения команды (было бы клёво сообщить пользователю, что всё сделано), то нужно придумать отдельное хранилище для этого результата, и удобный способ его получения.

Ну, и так далее. Не факт, что при возникновении подобных проблем решить их у вас получится быстрее и лучше, чем у перфекционистов с дедлайнами, не освоившими синтаксис crontab.



[solshark](#), 12 июля 2011 в 11:41 #



-2

Ну, я потому и сослался на перфекционизм — это будет работать замечательно если раньше не писал связку cron + commands. Но, написав однажды, получается не менее гибкий механизм. Статусы и локи, которые Вы упомянули — достаточно тривиальная вещь.

Встречный вопрос, раз уж мы заговорили про трудоемкие задачи (вроде генерации большой пачки pdf или ресайза картинок) — как на счет писе? Позволяет описанная система управлять приоритетами? Опять же, если вдруг что-то пошло не так, команду можно банально выполнить вручную и глянуть что происходит. А как с этим в Celery?



[Outspector](#), 12 июля 2011 в 13:45 #



+3

Никто не говорит, что добавить к задаче статус выполнения — это сложно. Речь идет о том, что это уже получается не просто «связка cron+management commands», а «связка cron+management commands+дополнительная логика, про которую нужно не забыть, и которую нужно сделать».

С задачей расстановки приоритетов я не сталкивался, но принципиально не вижу никаких сложностей. Тот, что лежит на поверхности — запустить несколько демонов `celeryd`, каждый из которых будет обрабатывать свою очередь. Задачи разных приоритетов будут раскладываться в разные очереди. Интенсивность загрузки можно регулировать количеством процессов демона, обслуживающего очередь, или запустив один из демонов под `найсом`.

> Опять же, если вдруг что-то пошло не так, команду можно банально выполнить вручную и глянуть что происходит. А как с этим в Celery?

Создавая задачу, вы навешивает декоратор на функцию. Эта же функция может быть вызвана и напрямую, без обращения к брокеру.



[Outspector](#), 12 июля 2011 в 14:02 #



0

Никто не говорит, что добавить к задаче статус выполнения — это сложно. Речь идет о том, что это уже получается не просто «связка `cron+management commands`», а «связка `cron+management commands`+дополнительная логика, про которую нужно не забыть, и которую нужно сделать».

С задачей расстановки приоритетов я не сталкивался, но принципиально не вижу никаких сложностей. Например, довольно просто запустить несколько демонов `celeryd`, каждый из которых будет обрабатывать свою очередь. Задачи разных приоритетов будут раскладываться в разные очереди. Интенсивность загрузки можно регулировать количеством процессов демона, обслуживающего очередь, или запустив один из демонов под `найсом`.

> Опять же, если вдруг что-то пошло не так, команду можно банально выполнить вручную и глянуть что происходит. А как с этим в Celery?

Создавая задачу, вы навешивает декоратор на функцию. Эта же функция может быть вызвана и напрямую, без обращения к брокеру.



[Outspector](#), 12 июля 2011 в 14:21 #



+1

Сорри, не удержался, и ответил дважды.



[pradis](#), 12 июля 2011 в 13:33 #



0

Ну конечно же `cron+custom management commands` не замена фиче асинхронных вычислений `celery`, изначальный мой коммент относился к фиче `celery` запускать периодические задания.

Если вы говорите о высоконагруженных платформах то выполнять асинхронные задания нужно как говорится `offline`. Юзер не должен влиять на процесс выполнения оффлайновых задач иначе вы начнете бороться с проблемой нагруженности серверов в пиковые моменты которая будет еще возрастать если юзер будет добавлять туда задачи самостоятельно а они сразу же выполняться.

Если речь о маленьких сайтах где практически нет никакой нагрузки мне сложно представить ситуацию когда нужно выполнить какие-то действия асинхронно.

"— если вам нужно отслеживать результат выполнения команды (было бы клёво сообщить пользователю, что всё сделано), то нужно придумать отдельное хранилище для этого результата, и удобный способ его получения." — ну так и сообщайте прямо из этой `management command`. Или как вы по другому собираетесь сообщить? Сохранить в базе что таск завершен а при каждой загрузке страницы юзером трекать базу с задачами (читать лишние дополнительные запросы к базе) и сообщать ему как-то? Получается не просто добавить декоратор чтобы задача выполнялась асинхронно а еще и кусок кода для проверки статусов и репорта юзерам...



[Outspector](#), 12 июля 2011 в 15:00 #



0

> Если речь о маленьких сайтах где практически нет никакой нагрузки мне сложно представить ситуацию когда нужно выполнить какие-то действия асинхронно.

Приведу такой пример, который встречается, к сожалению, довольно часто. Есть интернет-магазин, на который приходят 1,5 посетителя в сутки, и в основное время нагрузка на сайт практически нулевая. А еще у администратора есть поле `upload` и кнопка типа «загрузить нomenclature», с помощью которой он раз в сутки обновляет информацию на сайте. Через `upload` передается многометровый XML или XLS, который начинает тут же обрабатываться и записываться в базу. Даже если этот процесс хорошо оптимизирован, все равно время на обработку легко может превысить максимальное время ожидания ответа веб-сервером, и процесс будет принудительно завершен. Это типичная задача, которая просится в оффлайн, и мы (я имею в виду техподдержку NetAngels) всегда советуем обрабатывать такие задачи как раз с помощью `cron-a`.

Другая подобная задача возникает уже у меня. Панель управления хостингом не оперирует большим количеством данных, и большинство времени нагрузка на ней тоже близка к нулевой. Однако эта панель в ответ на действия пользователя отправляет команды на удаленные серверы (создай сайт, обнови конфиг и т.п.). Если этот сервер чрезмерно нагружен или находится в другом месте и есть проблемы на каналах связи, то работать с панелью станет, как минимум, некомфортно. С другой стороны, если пользователь нажал на кнопку «изменить пароль ftp», то заставлять ждать его 1-3 минуты, пока задание выполнит `cron`, тоже негуманно. Поэтому мы вместо этого формируем задания, которые `celery` выполняет так быстро, как ей это позволяют условия.

> ну так и сообщайте прямо из этой management command

Да, тут я согласен. Действительно, обычно не будет проблемой сообщить что-то клиенту из задачи celery или из management команды. С другой стороны, если возникла необходимость отслеживать состояние задачи из вызывающей функции, то достаточно просто сохранить `async_result.task_id` и периодически обращаться к задаче, проверяя, не выполнена ли она. Объект `request.session` выглядит удобным местом для сохранения таких идентификаторов. В тех случаях, когда задача выполняется за время, сопоставимое с интервалом между запросами клиента, overhead будет минимальным.



[prudis](#), 12 июля 2011 в 15:28 # ↑

0

«Это типичная задача, которая просится в оффлайн, и мы (я имею в виду техподдержку NetAngels) всегда советуем обрабатывать такие задачи как раз с помощью cron-а.» ну вот и вернулись к крону, причем тут асинхронные задачи?

Если у вас есть проблемы на каналах связи между хостинговой панелью и актуальным хостинговым сервером то да, у вас большие проблемы, Сомневаюсь что celery поможет в этом случае. Запрос от хостинговой панели на хост сервер это обычный запрос, и опять же, если он выполняется больше 30 сек то кому нужен такой хостинг?



[Outspector](#), 12 июля 2011 в 15:44 # ↑

0

> ну вот и вернулись к крону, причем тут асинхронные задачи?

А разве выполнение задач через cron — это не один из способов реализации асинхронных задач? Это был пример того, что даже на ненагруженном сайте не все действия нужно выполнять синхронно, то есть в цикле обработки HTTP-запроса.

И раз уж речь зашла о «ненужном» хостинге. Панель управления управляет не только shared-хостингом. Она работает с разными серверами, в т.ч. и не на нашей площадке, а также с серверами клиентов (которые заказали VDS или разместились на collocation). При всем желании, если канал между площадками просел, или клиент загрузил свой сервер так, что тот еле ворочается, хостер ничем помочь не сможет.

Если вам не нравится такой пример, представьте, что вы парсите выдачу яндекса, обращаетесь к стороннему REST API или ходите по другому сайту, на скорость работы которого никак повлиять не можете. В любом случае, асинхронное выполнение задач, даже если клиент всего один — ваш браузер, будет удачным решением.



[gigimon](#), 12 июля 2011 в 16:00 # ↑

0

Простой пример, юзер заливает картинку/видео и надо ее преобразовать в фоне, кидаете задание в celery и забываете про него. С management командами будет сильно сложнее



[HoldenCaulfield](#), 12 июля 2011 в 02:40 #

-2

В чем смысл такой статьи? Вы поверхностно перечислили несколько основных моментов, о которых уже есть тысяча таких же статей и раздел «Getting started» документации. Только энтропию увеличиваете.



[black_bunny](#), 12 июля 2011 в 06:06 # ↑

0

Я конечно же против увеличения энтропии, но в чём смысл статьи написал в самом начале, и мне не удалось найти актуальной информации по установке и настройке, собранной в одном месте.



[Bteam](#), 12 июля 2011 в 06:29 # ↑

+2

Статья отличная, а вы пахоже не заметили что «Данная статья будет полезна прежде всего новичкам»



[seriyPS](#), 12 июля 2011 в 14:14 #

+1

Полезная штука, очень активно используем в Pylons проекте, RabbitMQ как брокер. Но вот почему-то часто появляются зависшие задания / zombu — процессы, ну и по Ctrl+C не всегда возможно прибить. Недостаточно стабильное в общем.

А так да — если даже и не применять активно, то знать о существовании такой штуки совершенно точно нужно, иначе будешь жестко велосипедить.



[kmike](#), 12 июля 2011 в 22:50 # ↑

0

У меня такое было, если воркер на eventlet или gevent. Точнее, eventlet как-то вообще через раз заводился, а gevent вис под нагрузкой, особенно если в задачах выпадали исключения. Никак в итоге не решил, пока забил (ресурсов сейчас хватает). Видимо, на какой-нибудь торнадо буду отдельно потом неблокирующего работника писать.



[seriyPS](#), 12 июля 2011 в 23:25 # ↑

0

Хм... неблокирующий работник? Не представляю зачем... Если только работники занимаются загрузкой веб-страничек...



[kmike](#), 13 июля 2011 в 00:10 #



0

Ну да, 2 очереди, io (сеть, диск) параллелить неблокирующими работниками, сри — процессами, схема замечательная, если бы работала)

Сделал пока тоже 2 очереди (под io-задачи и сри-задачи), но обе просто на процессах. В сри-очереди смысла особого нет запускать процессов больше, чем ядер у машины, а вот в io — чем больше, тем лучше.



[serivPS](#), 12 июля 2011 в 14:17 #

+1

Да, еще интересная возможность — прозрачный запуск задач на разных машинах. Т.е. запускаешь воркеров на 10 серверах и один RabbitMQ брокер — задачи распределяются по кластеру уже самостоятельно.



[deeGraYve](#), 12 июля 2011 в 15:46 #

+3

написали на работе ztask для всего этого, Celery показался слишком громоздким и тяжелым для того, что мы делали. ztask использует OMQ, если кому интересно — вэлкам github.com/dmgctrl/django-ztask



[black_bunny](#), 12 июля 2011 в 16:01 #



+1

Кажется вот здесь написано про историю создания ztask: <http://www.zeromq.org/story:3>

Очень интересно, чуваки решили что слишком много всего устанавливается и надо бы написать что-то своё очень простое, и судя по всему, у них это получилось довольно хорошо.

Кстати там написано что ztask умеет перезагружать изменённые файлы, это действительно так?



[deeGraYve](#), 12 июля 2011 в 16:09 #



0

ага, мы с Джэйсоном работали над нашим внутренним проектом и нам не хотелось устанавливать больше 1й сторонней библиотеки, а у Дэйва это было вообще одним из условий сдачи его собственного проекта. Так что, объединив усилия, они написали ztask, который мы теперь почти везде используем для асинхронных задач.

Насчет перезагрузки — да, он умеет перезагружать измененные файлы.



[oduvan](#), 23 января 2012 в 16:36 #



0

```
pip install pyzmq
pip install -e git+git@github.com:dmgctrl/django-ztask.git#egg=django_ztask
```

or

```
easy_install Celery
easy_install django-celery
```



[deeGraYve](#), 23 января 2012 в 17:13 #



0

django_ztask для нас сторонней библиотекой не является



[oduvan](#), 23 января 2012 в 17:17 #



0

а django-celery — является?



[deeGraYve](#), 23 января 2012 в 17:22 #



0

да, к django-celery мы отношения не имеем, а django_ztask мы писали сами



[oduvan](#), 23 января 2012 в 18:23 #



0

Понятно. Тогда довольно странное условие задачи проекта :)

Заказчик в курсе, что Django тоже не Вами написано



[deeGraYve](#), 23 января 2012 в 18:25 #



0

заказчик — мы, на все вопросы ответил? «вкурсе» кстати пишется раздельно :)



[oduvan](#), 23 января 2012 в 18:29 #



0

«вкурсе» кстати пишется как хочеш)



[black_bunny](#), 23 января 2012 в 18:30 #



0

Ну хватит уже :)



[deeGraYve](#), 23 января 2012 в 18:32 #



0

ну раз человек спрашивает — почему бы не ответить :) только б спрашивал еще по делу...



[oduvan](#), 23 января 2012 в 18:31 #



0

не, не на все. даже появилось еще больше.

почему такое странное условие для задачи проекта?

А еще проект надо написать на свеже поставленной убунте не подключенной к интернету))) Ну чтоб ну вообще... чтоб даже и мысли небыло глянуть в чужой код)



[deeGraYve](#), 23 января 2012 в 18:48 #



0

в чей чужой код?



[oduvan](#), 23 января 2012 в 18:50 #



0

"и нам не хотелось устанавливать больше 1й сторонней библиотеки, а у Дэйва это было вообще одним из условием сдачи его собственного проекта." — я вот эту вот фразу перетираю



[deeGraYve](#), 23 января 2012 в 19:00 #



0

коммерческая тайна



[oduvan](#), 23 января 2012 в 19:03 #



0

«ну раз человек спрашивает — почему бы не ответить :) только б спрашивал еще по делу...»

Ну... спасибо, что хоть с русским языком помогли.



[deeGraYve](#), 23 января 2012 в 19:09 #



0

не обижайся, я просто так и не понял к чему это все было, к тому что нужно было пользоваться celery?



[oduvan](#), 23 января 2012 в 19:15 #



0

обиделся :(

"и нам не хотелось устанавливать больше 1й сторонней библиотеки, а у Дэйва это было вообще одним из условием сдачи его собственного проекта."

Как может быть условие сдачи проекта — «устанавливать не больше 1ой сторонней библиотеки»

Мне кажется условие «не устанавливать сторонние библиотеки» — само по себе глупо, а не больше одно вообще мне вынесло можечок



[deeGraYve](#), 23 января 2012 в 19:19 #



0

ну извини, в следующий раз обязательно спросим твоего совета об использовании сторонних библиотек. судя по твоему блогу, ты ярый приверженец использования готовых библиотек, я ниче против не имею, как говорится «when it makes sense», здесь же условия оказались другими и, мне кажется, время мы потеряли не зря, django_task живет и используются не только нами, но и другими людьми.



[oduvan](#), 23 января 2012 в 19:30 #



0

ох... совета спросим... приверженец готовых библиотек... ну вы даете...

Я пытаюсь выяснить в чем причина. Ну хорошо... Возможные варианты.

— Мы думаем, что в опенсорсе один гавнокод

или

— Мы любим вилосипеды

или

— Мы в проекте не имеем права использовать библиотеки с определенным типом лицензий

или

— У нас такой дурной бюджет, что чем больше мы кодим, тем больше зарабатываем, так что мы еще и свою джангу замутили

или

— Мы считаем что только сцикуны используют готовые библиотеки а настоящие мужики кодят все сами

или

— Нам легче писать свой код, чем поддержать чужой

или

— в следующий раз обязательно спросим твоего совета об использовании сторонних библиотек

а, не стоп, Вы так и ответили... Последнее вычеркиваем... А какой я, кстати, совет дал?



deeGraYve, 23 января 2012 в 19:48 # ↑

0

«Мне кажется условие «не устанавливать сторонние библиотеки» — само по себе глупо» звучит как призыв к установке сторонних библиотек. (здесь я игнорирую факт, что по сути вы назвали несколько довольно неглупых людей глупыми, но сами знаете на кого не обижаются...)

- 1) множество наших проектов opensource, говнокодом их не считаю
- 2) если бы вы посмотрели код celery и код django_task, вы бы поняли разницу между ними, и не сравнивали бы проект с велосипедом
- 3) это тоже отчасти правда, но это за рамками NDA
- 4) клиентов мы не дурим, благо клиенты все люди довольно хорошо разбирающиеся в программировании
- 5) абсолютно неверно, когда есть смысл — используем готовые библиотеки, но чтобы это делать хорошо, нужно знать на что они способны, их + и —
- 6) писать свой код почти что всегда легче, чем поддерживать чужой, но не всегда целесообразно
- 7) уже озвучил



oduvan, 23 января 2012 в 22:39 # ↑

0

Ух!!! Ну Вы даете!!!)))))))))))

Вот так и знал, надо было сказать — только без номерков!!!))))

«Мне кажется условие «не устанавливать сторонние библиотеки» — само по себе глупо» звучит как призыв к установке сторонних библиотек.

Я Вам крайне рекомендую еще раз утричком на свежую голову прочитать то что Вы написали.

Я просто хотел узнать. Какова причина того, что вы не дали добро на использование сторонних библиотек.

Фразы между или — это примеры. Возможных варинатов ответов. Это не вопросы к Вам. Там половину вариантов было шуточных. Это я просто Вам показал каким мог быть ответ на мой вопрос. Которким и лаконичным без лишних букв никому не нужных.

Но пожалуй ответ мне уже не нужен, достаточно начитался. Надо работать.

Опять же бложик обновить, раз у меня такой фан появился)))



deeGraYve, 23 января 2012 в 22:49 # ↑

0

увольте, когда я увидел количество скобочек — даже продолжать и пересчитывать свой же комментарий расхотелось)))))))))))



Fak3, 14 июля 2011 в 00:53 # ↑

0

А django-ztask предоставляет API для периодических задач?
И интегрируется ли он в админку, как это делает Celery?



deeGraYve, 15 июля 2011 в 08:47 # ↑

0

не совсем понял вопрос про api для периодических задач...
в admin ztask напрямую не интегрируется, но это решается довольно просто — добавьте модель Task так, чтобы она была видна в admin.



black_bunny, 15 июля 2011 в 09:17 # ↑

0

Кажется я понял вопрос, в модуле decorators у ztask нет periodic_task, то есть периодические задания нельзя создавать.



glader, 16 декабря 2011 в 13:29 #

0

Не совсем понял, если у меня несколько сайтов, можно ли им пользоваться одной очередью? Или надо для каждого запускать отдельный celeryd?



[black_bunny](#), 16 декабря 2011 в 13:32 # ↑

0

можно пользоваться одной, по умолчанию так и получится.



[glader](#), 16 декабря 2011 в 14:15 # ↑

0

Соответственно, надо следить, чтобы задачи были названы по-разному? Или оно само разруливается?



[black_bunny](#), 16 декабря 2011 в 14:41 # ↑

0

Вообще говоря да, но я сейчас понял, что тут есть фундаментальная проблема :) — если в качестве celery backend используется БД, то все сайты должны использоваться одну и ту же БД чтобы все это работало, поэтому в твоём случае стоит наверное переключиться на RabbitMQ для хранения заданий, оно работает по сети и все смогут с ним взаимодействовать. И будет неважно откуда его запускать.



[glader](#), 16 декабря 2011 в 14:24 # ↑

0

И ещё уточни плз. насколько я понял, celery запускается из каталога проекта. Если проектов больше одного — из какого каталога его запускать?



[Ravall](#), 8 декабря 2012 в 12:58 #

0

Есть вопрос. А как из интерфейса вызывать какой-нибудь зарегистрированный task? в django в разделе Djselery / Tasks нет кнопки добавить задачу.



[black_bunny](#), 8 декабря 2012 в 13:09 # ↑

+1

Скорее всего в **django-celery** нет такой функциональности в базовом комплекте. Но к нему есть различные расширения, например вот это — github.com/mattcaldwell/django-celery-admin-ext, позволяющее запускать периодические задачи прямо из админки.



[Ravall](#), 8 декабря 2012 в 13:11 # ↑

0

спасибо!



[suguby](#), 25 июня 2013 в 17:35 #

+1

Спасибо за статью! Поправьте ссылочку на демонизацию docs.celeryproject.org/en/latest/tutorials/daemonizing.html#daemonizing

[С камерой в облака. Часть 2](#)

[Steam OS: Linux с игровым привкусом](#)

[Петиции, требующие запретить игру «Company of Heroes-2» в СНГ, набрали около 15 тысяч подписей](#)

[Все мозги в одном месте](#)

[Заказы для фрилансеров](#)

[Вакансии для айтишников](#)

[Уютная и дружелюбная](#)