Расширение django-celery для хранения информации о задачах

Выпуск 1.0.0

БАРС Груп

23 September 2014

Оглавление

AsyncTask - модуль, расширяющий функционал работы с задачами celery и надстройкой django-celery.

Глава 1

Предпосылки

Один из случаев применения задач celery - организация выполнения длительных процессов и отслеживание их состояния. Это так называемые “фоновые задачи”, которые выполняются параллельно с основным функционалом web-приложения.

Для подобного случая необходимо расширить функционал задач celery:

- Обеспечить классификацию “фоновых задач” и расширить атрибуты задачи в соответствие с потребностями приложения.
- Отслеживать состояние “фоновых задач”.

Глава 2

Архитектура

AsyncTask использует для хранение информации о состоянии/результате задачи базу данных и модель TaskMeta.

AsyncTaskDatabaseBackend (наследник DatabaseBackend из django-celery) - это менеджер хранения задач.

Для дополнительных атрибутов задачи можно создать собственную модель, отнаследовавшись от модели TaskMeta, и указав собственную модель как модель хранения задачи в менеджере.

Функционал передачи дополнительных атрибутов задачи в менеджер хранения обеспечивает класс AsyncTask (наследник Task из celery).

Для упрощения объявления таких задач создан декоратор async_task.

Дополнительные атрибуты устанавливаются через парметр ‘fields’. Он может указываться в декораторе async_task, вызове задачи и методе обновления состояния задачи.

Задача сохраняется в базу данных, при любых изменениях состояния, в том числе и при постановке в очередь (в отличие от функционала django-celery).

Также, при изменении состояния, менеджер вызывает дополнительный метод notify, который может обеспечить отправку уведомления о состоянии задачи клиенту (например, через websocket).

Глава 3

Пример использования

3.1 Настройка и описание

Опишем модель с дополнительными атрибутами задачи:

```
class AsyncTaskMeta(TaskMeta):
    owner = models.ForeignKey(User, verbose_name=u'Владелец',
                              null=True, blank=True, on_delete=CASCADE)
    provider = models.ForeignKey(Provider, verbose_name=u'Учреждение',
                                 on_delete=CASCADE, null=True, blank=True)
    name = models.CharField(u'Наименование', max_length=200,
                            null=True, blank=True)
    current = models.IntegerField(u"Текущий пункт", default=0)
    total = models.IntegerField(u"Всего пунктов", default=100)
    started = models.DateTimeField(u'Время начала', auto_now_add=True)
    finished = models.DateTimeField(u'Время завершения', auto_now=True)
    task_type = models.CharField(verbose_name=u'Тип задачи', max_length=20,
                                 choices=AsyncTaskType.get_choices(),
                                 default=AsyncTaskType.DEFAULT)

    objects = AsyncTaskManager()

    class Meta:
        db_table = 'async_task'
```

Примечание: Важно наследовать модель от TaskMeta и указать AsyncTaskManager в качестве менеджера запросов

Опишем менеджера хранения и укажем ему модель для хранения:

```
class AsyncTaskBackend(AsyncTaskDatabaseBackend):
    TaskModel = AsyncTaskMeta
```

Укажем celery использовать новый менеджер хранения в settings.py:

```
CELERY_RESULT_BACKEND = 'project.async.AsyncTaskBackend'
import djcelery
djcelery.setup_loader()
```

Опишем фоновую задачу через декоратор. Внутри задачи будем обновлять состояние задачи и ее параметры:

```
@async_task(fields={'name': u'Долгое сложение', 'task_type': AsyncTaskType.LONG})
def test_task(x, y, **kwargs):
    sleep(5)
    test_task.update_state(meta={'fields': {'current': 33}})
    sleep(5)
    // можно указать собственное состояние
    test_task.update_state(state='PROGRESS', meta={'fields': {'current': 67}})
    sleep(5)
    res = x + y
    if res == 0:
        raise Exception()
    test_task.update_state(meta={'fields': {'current': 100}})
    return res
```

Примечание: Важно указать в параметрах задачи **kwargs для передачи дополнительных параметров

Для прикладных случаев, можно создать упрощающие функции:

```
def update_task(task, current=0, total=None, state=None, result=None):
    update_dict = {
        'current': current,
    }

    if state is not None:
        update_dict['status'] = state

    if result is not None:
        update_dict['result'] = result

    task.update_state(meta={'fields': update_dict})

update_task(test_task, 33)
update_task(test_task, 67, state='PROGRESS')
```

3.2 Работа с задачами

Вызов задачи с параметрами:

```
test_task.delay(1,2, fields={'owner': request.user, 'provider': provider_id})
```

Для прикладных случаев, можно создать функцию упрощающую вызов:

```
def run_task(task, params, user, provider):
    return task.delay(*params, fields={'owner': user, 'provider': provider})

run_task(test_task, (1,2), user, provider_id)
```

Получение списка задач по учреждению:

```
# за исключением отмененных
tasks = AsyncTaskMeta.objects.filter(provider=provider_id).exclude(status='REVOKED')
```

Получение задачи по task_id и ее отмена:

```
task = AsyncTaskMeta.objects.get(task_id=task_id)
# остановка выполенения задачи в celery
celery.control.revoke(task_id=task_id, terminate=True)
# удаление из хранилища
task.delete()
```

3.3 Уведомление о состоянии задачи

Для уведомления об изменении состоянии задачи достаточно указать в менеджере хранения объект notifier, у которого будет вызван метод notify и передана задача:

```
class WebsocketSender(object):
    u""" Синглтон клиента-отправителя """
    def __new__(cls, *args, **kwargs):
        if not hasattr(cls, 'sender'):
            from pushme.mq import get_sender
            cls.sender = get_sender(
                settings.WEBSOCKET_BACKEND,
                (settings.WEBSOCKET_QUEUE_HOST, settings.WEBSOCKET_QUEUE_PORT)
            )
        return cls.sender

    @classmethod
    def notify(cls, task):
        if settings.USE_WEBSOCKET:
            data = task.serialize() or {}
            json_data = simplejson.dumps(data)
            cls().send(
                data=json_data,
                uid=task.owner_id,
                topic='task_state_'+str(task.provider_id)
            )


class AsyncTaskBackend(AsyncTaskDatabaseBackend):
    TaskModel = AsyncTaskMeta
    notifier = WebsocketSender
```