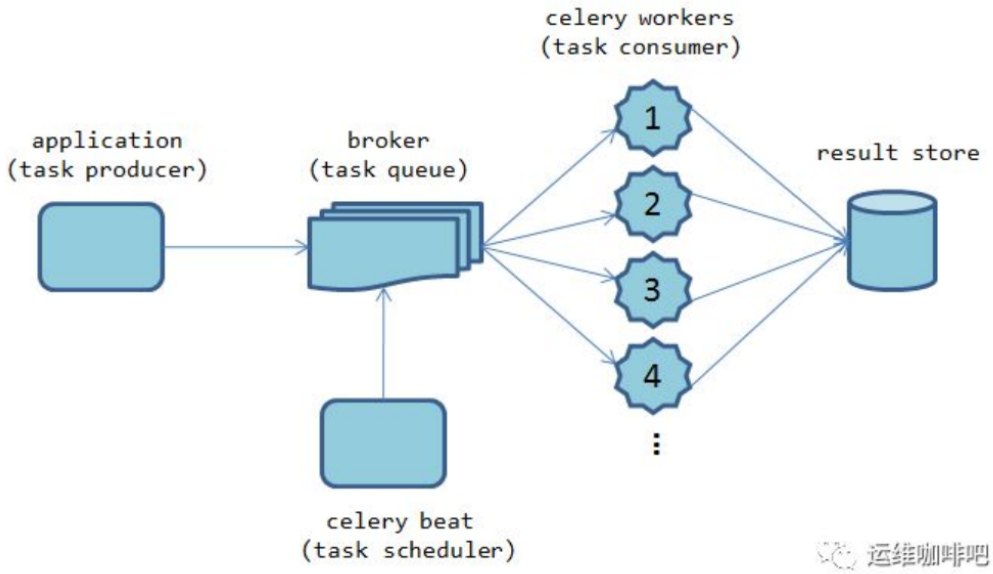
异步任务和定时任务

Django+Celery 执行异步任务和定时任务

原生Celery，非djcelery模块，所有演示均基于Django2.0

celery是一个基于python开发的简单、灵活且可靠的分布式任务队列框架，支持使用任务队列的方式在分布式的机器/进程/线程上执行任务调度。采用典型的生产者-消费者模型，主要由三部分组成：

1. 消息队列broker：broker实际上就是一个MQ队列服务，可以使用Redis、RabbitMQ等作为broker
2. 处理任务的消费者workers：broker通知worker队列中有任务，worker去队列中取出任务执行，每一个worker就是一个进程
3. 存储结果的backend：执行结果存储在backend，默认也会存储在broker使用的MQ队列服务中，也可以单独配置用何种服务做backend



图片来自互联网

异步任务

我的异步使用场景为项目上线：前端web上有个上线按钮，点击按钮后发请求给后端，后端执行上线过程要5分钟，后端在接收到请求后把任务放入队列异步执行，同时马上返回给前端一个任务执行中的结果。若果没有异步执行会怎么样呢？同步的情况就是执行过程中前端一直在等后端返回结果，页面转呀转的就转超时了。

异步任务配置

1.安装RabbitMQ，这里我们使用RabbitMQ作为broker，安装完成后默认启动了，也不需要其他任何配置

```
# apt-get install rabbitmq-server
```

2.安装celery

```
# pip3 install celery
```

3.celery用在django项目中，django项目目录结构(简化)如下

```
website/
|-- deploy
|   |-- admin.py
|   |-- apps.py
|   |-- __init__.py
|   |-- models.py
|   |-- tasks.py
|   |-- tests.py
|   |-- urls.py
|   `-- views.py
|-- manage.py
|-- README
`-- website
    |-- celery.py
    |-- __init__.py
    |-- settings.py
    |-- urls.py
    `-- wsgi.py
```

4.创建`website/celery.py`主文件

```
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery, platforms

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'website.settings')

app = Celery('website')

# Using a string here means the worker don't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django app configs.
app.autodiscover_tasks()

# 允许root 用户运行celery
platforms.C_FORCE_ROOT = True

@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

5.在`website/__init__.py`文件中增加如下内容，确保django启动的时候这个app能够被加载到

```
from __future__ import absolute_import

# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app

__all__ = ['celery_app']
```

6.各应用创建tasks.py文件，这里为`deploy/tasks.py`

```
from __future__ import absolute_import
from celery import shared_task

@shared_task
def add(x, y):
    return x + y
```

- **注意tasks.py必须建在各app的根目录下，且只能叫tasks.py，不能随意命名**

7.views.py中引用使用这个tasks异步处理

```
from deploy.tasks import add

def post(request):
    result = add.delay(2, 3)
```

- 使用函数名.delay()即可使函数异步执行
- 可以通过`result.ready()`来判断任务是否完成处理
- 如果任务抛出一个异常，使用`result.get(timeout=1)`可以重新抛出异常
- 如果任务抛出一个异常，使用`result.traceback`可以获取原始的回溯信息

8.启动celery

```
# celery -A website worker -l info
```

9.这样在调用post这个方法时，里边的add就可以异步处理了

定时任务

定时任务的使用场景就很普遍了，比如我需要定时发送报告给老板~

定时任务配置

1.`website/celery.py`文件添加如下配置以支持定时任务crontab

```
from celery.schedules import crontab

app.conf.update(
    CELERYBEAT_SCHEDULE = {
        'sum-task': {
            'task': 'deploy.tasks.add',
            'schedule':  timedelta(seconds=20),
            'args': (5, 6)
```

```
        }
        'send-report': {
            'task': 'deploy.tasks.report',
            'schedule': crontab(hour=4, minute=30, day_of_week=1),
        }
    }
)
```

- 定义了两个task：
  - 名字为'sum-task'的task，每20秒执行一次add函数，并传了两个参数5和6
  - 名字为'send-report'的task，每周一早上4：30执行report函数
- timedelta是datetime中的一个对象，需要`from datetime import timedelta`引入，有如下几个参数
  - `days`：天
  - `seconds`：秒
  - `microseconds`：微妙
  - `milliseconds`：毫秒
  - `minutes`：分
  - `hours`：小时
- crontab的参数有：
  - `month_of_year`：月份
  - `day_of_month`：日期
  - `day_of_week`：周
  - `hour`：小时
  - `minute`：分钟

1. `deploy/tasks.py`文件添加report方法：

```
@shared_task
def report():
    return 5
```

3.启动celery beat，celery启动了一个beat进程一直在不断的判断是否有任务需要执行

```
# celery -A website beat -l info
```

Tips

1. 如果你同时使用了异步任务和计划任务，有一种更简单的启动方式`celery -A website worker -b -l info`，可同时启动worker和beat
2. 如果使用的不是rabbitmq做队列那么需要在主配置文件中`website/celery.py`配置broker和backend，如下：

```
# redis做MQ配置
app = Celery('website', backend='redis', broker='redis://localhost')
# rabbitmq做MQ配置
app = Celery('website', backend='amqp', broker='amqp://admin:admin@localhost')
```

1. celery不能用root用户启动的话需要在主配置文件中添加`platforms.C_FORCE_ROOT = True`
2. celery在长时间运行后可能出现内存泄漏，需要添加配置`CELERYD_MAX_TASKS_PER_CHILD = 10`，表示每个worker执行了多少个任务就死掉

celery是一个基于python开发的简单、灵活且可靠的分布式任务队列框架，支持使用任务队列的方式在分布式的机器/进程/线程上执行任务调度。采用典型的生产者-消费者模型，主要由三部分组成:

\1. 消息队列broker：broker实际上就是一个MQ队列服务，可以使用redis、rabbitmq等作为broker
\2. 处理任务的消费者workers：broker通知worker队列中有任务，worker去队列中取出任务执行，每一个worker就是一个进程
\3. 存储结果的backend：执行结果存储在backend，默认也会存储在broker使用的MQ队列服务中，也可以单独配置用何种服务做backend

flask,django是同步框架，所有的请求以队列形式完成。这样的话效率极差，用户体验不好，为了解决这个问题引入celery异步方式在后台执行这些任务（这里使用到了redis,3.0以下兼容性更好）

1，安装依赖

```
pip install celery

pip install celery-with-redis

pip install django-celery
```

2，settings.py设置

```
#配置celery
import djcelery
djcelery.setup_loader()
BROKER_URL = 'redis://127.0.0.1:6379'
CELERY_IMPORTS = ('mymac.tasks') #需执行异步的子应用

#将djcelery安装到应用中
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mysg',
    'lianxi',
    "rest_framework",
    'corsheaders',
    #异步
    'djcelery',
]
```

3,将异步的应用中注册celery.py

```
import os
import django
from celery import Celery
from django.conf import settings
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mymac.settings')
django.setup()
app = Celery('mymac')
app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

4，建立异步任务和定时任务的(tasks.py)

- **注意tasks.py必须建在各app的根目录下，且只能叫tasks.py，不能随意命名**

异步任务

```
import time,random
from celery import task
#发邮件
from django.core.mail import send_mail
from django.http import HttpResponse
#定义异步写文件方法
@task
def file_task():
        #写文件操作  文件对象
        file_object = open("./data.text",'a+',encoding='utf-8')
        file_object.write("hello")
        file_object.close()
        print("ok")

#定义异步发邮件的方法
@task
def email_():
        captcha_text = []
        for i in range(4):
        #定义验证码字符
                str = 'qwertyuiopasdfghjklzxcvbnm1234567890'
                c = random.choice(str)
                captcha_text.append(c)
        #返回随机生成的字符串
        captcha = "".join(captcha_text)
        res = send_mail("欢迎注册",'您的验证码是:'+ captcha,
['396961930@qq.com'],DEFAULT_FROM_EMAIL)
        if res:
                return HttpResponse("发送成功")
        else:
                return HttpResponse("发送失败")
```

views.py中引用使用这个tasks异步处理

```
from mymac.tasks import email,file_task

#异步发邮件
def email(request):
    print(email_.delay())
    return HttpResponse("异步发邮件")
#异步写入文件
def failtask(request):
    print(file_task.delay())
    return HttpResponse("success")#配好路由触发任务即可
```

定时任务

```
#导入定时任务库
from celery.decorators import periodic_task
from celery.schedules import crontab
#发短信
from twilio.rest import Client

#定义20点10分发送
#@periodic_task(run_every=crontab(minute=10,hour=20))
#定义10秒发送一次
@periodic_task(run_every=10)
def mail():
        #定义短信sid
        account_sid = 'ACbccc4d2127e888e6f6654dc8128c019e'
        #定义秘钥
        auth_token = 'a0f31b24c76c65c20c6400dc94537ac6'
        #定义客户端对象
        clinet = Client(account_sid,auth_token)
        #定义短信内容 1,发给谁 2,发信人 3,内容
        status =
clinet.messages.create(to="+8616637712137",from_="+12016361207",body="hello
world")
        if status:
                print("发送成功")

#注意时区 例如中国
#settings.py 设置  语言相关配置
LANGUAGE_CODE = 'zh-hans'
TIME_ZONE = 'Asia/Shanghai'

crontab的参数有:
month_of_year:月份
day_of_month:日期
day_of_week:周
hour:小时
minute:分钟
```

5，启动任务

启动服务的命令:

celery -A mymac beat -l info 定时任务
celery -A mymac worker -l info 异步任务

Django+Celery+xadmin实现异步任务和定时任务

关注公众号"轻松学编程"了解更多。

一、celery介绍

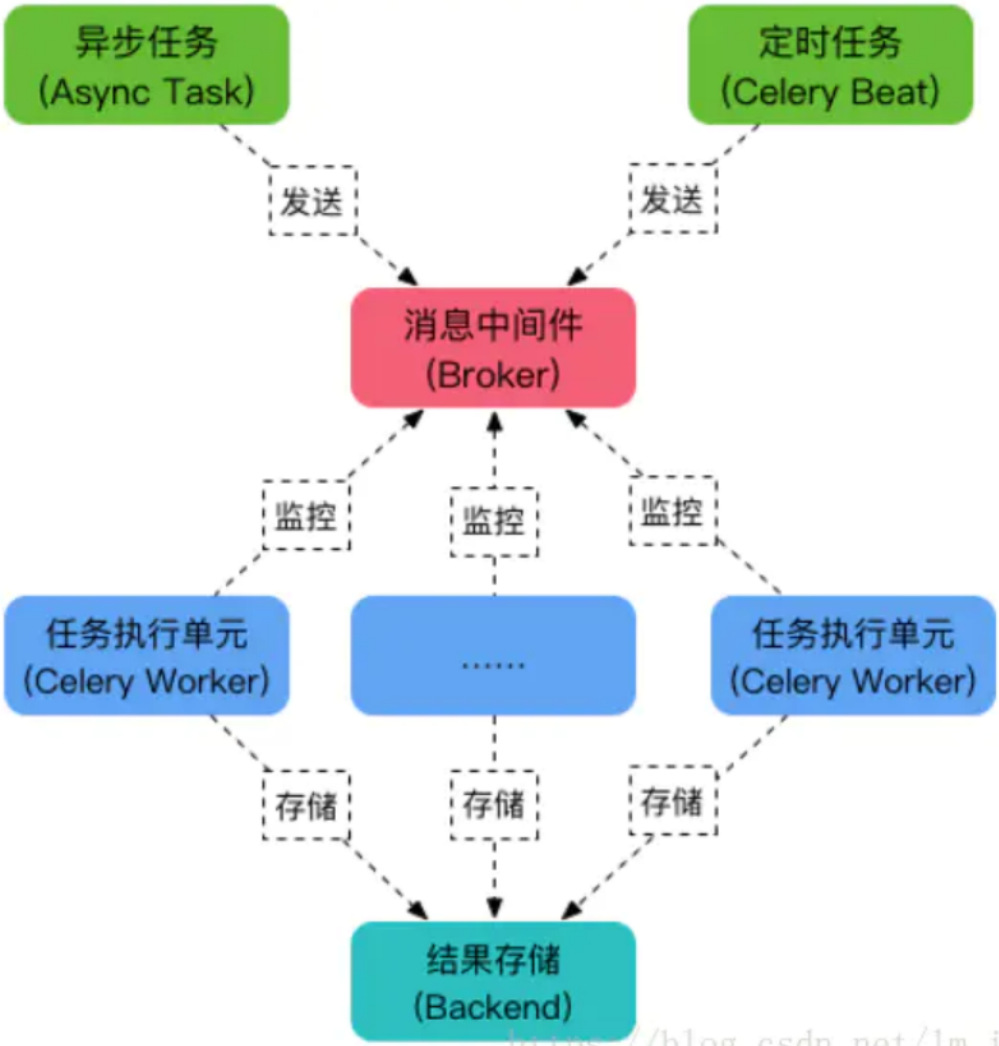
1、简介

【官网】http://www.celeryproject.org/

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）。

- 异步任务：比如发送邮件、短信，或者文件上传, 图像处理等等一些比较耗时的操作；
- 定时任务：需要在特定时间执行的任务。

架构组成如图：



image

2、Celery 主要包含以下几个模块

2.1 任务模块 Task

异步任务通常在业务逻辑中被触发并发往任务队列；

定时任务由 Celery Beat 进程周期性地将任务发往任务队列。

2.2 消息中间件 Broker

Broker，即为任务调度队列，接收任务生产者发来的消息（即任务），将任务存入队列。 Celery 本身不提供队列服务，官方推荐使用 RabbitMQ 和 Redis 等。

2.3 任务执行单元 Worker

Worker 是执行任务的处理单元，它实时监控消息队列，获取队列中调度的任务，并执行它。

2.4 任务结果存储 Backend

Backend 用于存储任务的执行结果，以供查询。同消息中间件一样，存储也可使用RabbitMQ, Redis 和 MongoDB 等。 MQ全称为Message Queue。

消息队列（MQ）是不同的应用程序相互通信的一种方法。

MQ是消费者-生产者模型的一个典型的代表，一端往消息队列中不断写入消息，而另一端则可以读取队列中的消息。

二、异步任务

假设已经有了一个Django项目(我的项目名是MySites)，下面演示如何使用Celery实现异步任务。

1、安装celery

```
pip install celery
pip install django-celery
pip install redis==2.10.6
pip install django_celery_beat
```

注意：如果出现以下错误，一般是celery版本不对，重新安装较高版本即可。

```
Running django in virtualenv - ImportError: No module name
django.core.management
```

2、配置settings.py

我使用Redis作为消息队列。

在settings.py中增加：

```
#django-celery
import djcelery

djcelery.setup_loader()
#使用本地redis服务器中的0号数据库，redis密码为123456
BROKER_URL = 'redis://:123456@127.0.0.1:6379/0'
```

```
CELERYBEAT_SCHEDULER = 'djcelery.schedulers.DatabaseScheduler'
CELERY_RESULT_BACKEND = 'redis://:123456@127.0.0.1:6379/1'
CELERY_ENABLE_UTC = False
CELERY_TIMEZONE = 'Asia/Shanghai'
CELERY_TASK_RESULT_EXPIRES = 10
CELERYD_LOG_FILE = BASE_DIR + "/logs/celery/celery.log"
CELERYBEAT_LOG_FILE = BASE_DIR + "/logs/celery/beat.log"
CELERY_ACCEPT_CONTENT = ['pickle', 'json', 'msgpack', 'yaml']
```

其中，当djcelery.setup_loader()运行时，Celery便会去查看INSTALLD_APPS下包含的所有app目录中的tasks.py文件，找到标记为task的方法，将它们注册为celery_task。

BROKER_URL和CELERY_RESULT_BACKEND分别指代你的Broker的代理地址以及Backend（result store）数据存储地址。

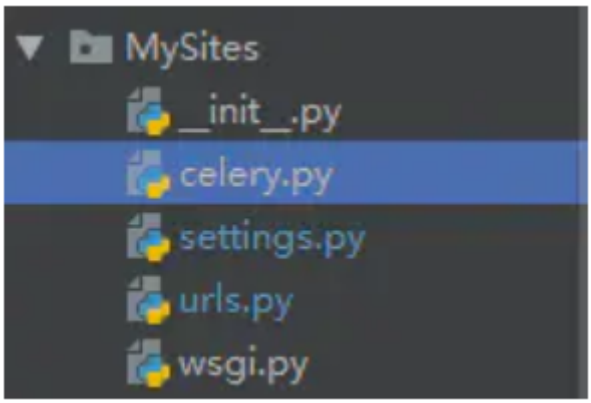
在Django中如果没有设置backend，会使用其默认的后台数据库用来存储数据。

注册celery应用：

```python
INSTALLED_APPS = [
 #其它应用
 ...
 #celery应用
 'djcelery',
]
```

3、创建celery.py文件

在项目中Mysites下创建celery.py文件(与settings.py同级)



image

内容为：

```
from celery import Celery
from django.conf import settings
import os

# 为celery设置环境变量，改为你项目的settings
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'MySites.settings')

# 创建应用
```

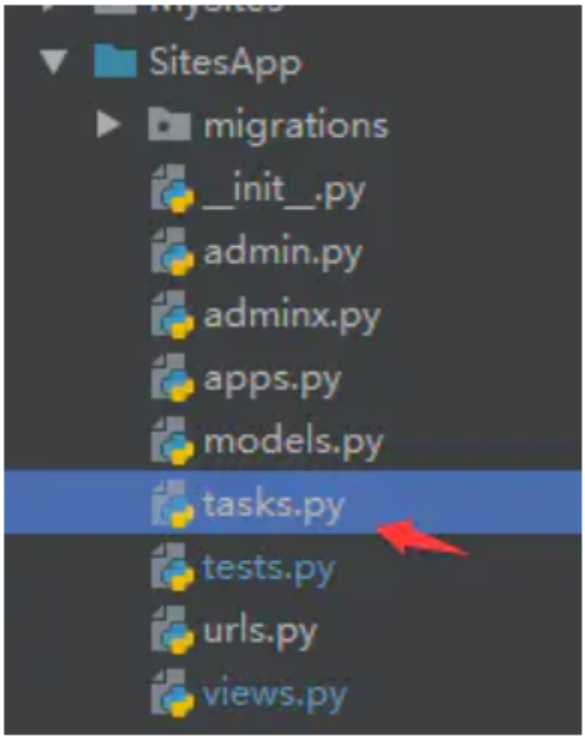
```
app = Celery('mysites')

# 配置应用
app.conf.update(
 # 本地Redis服务器
 BROKER_URL=settings.BROKER_URL,
)

app.autodiscover_tasks(settings.INSTALLED_APPS)
```

4、创建tasks.py文件

在子应用app下创建tasks.py：



image

内容为：

```
from MySites.celery import app

@app.task
def start_running(info):
 print(info)
 print('--->>开始执行任务<<---')
 print('比如发送短信或邮件')
 print('>---任务结束---<')
```

5、修改views.py

在views.py中增加需要执行的异步任务，比如：

```
from SitesApp.tasks import start_running

#celery测试
class CeleryTask(View):
 def get(self, request):
 print('>=====开始发送请求=====<')
 start_running.delay('发送短信')
 # start_running.apply_async(('发送短信',), countdown=10)  # 10秒后再执行异步任务
 return HttpResponse('<h2> 请求已发送 </h2>')
```

其实关键代码就一条`start_running.delay(参数)`,当执行这条代码时，系统会把tasks.py中的start_running函数推迟执行，即放入消息队列中。

系统相当于跳过`start_running.delay('发送短信')`执行后面的语句，这就是异步任务。

6、修改项目下(不是子应用下)的urls.py

```
from SitesApp import views

urlpatterns = [
 #其它url
 ...
 #celery测试url
 url('^celery/',views.CeleryTask.as_view()),
]
```

7、启动服务

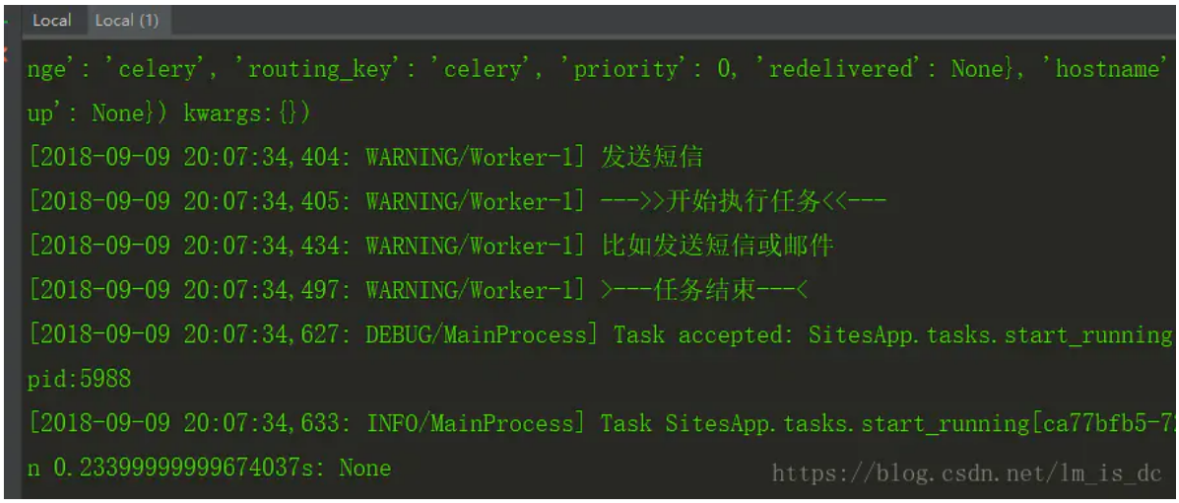
在Terminal中输入：

```
python manage.py runserver
```

在另一个Terminal中：

```
celery -A MySites worker --loglevel=DEBUG
```

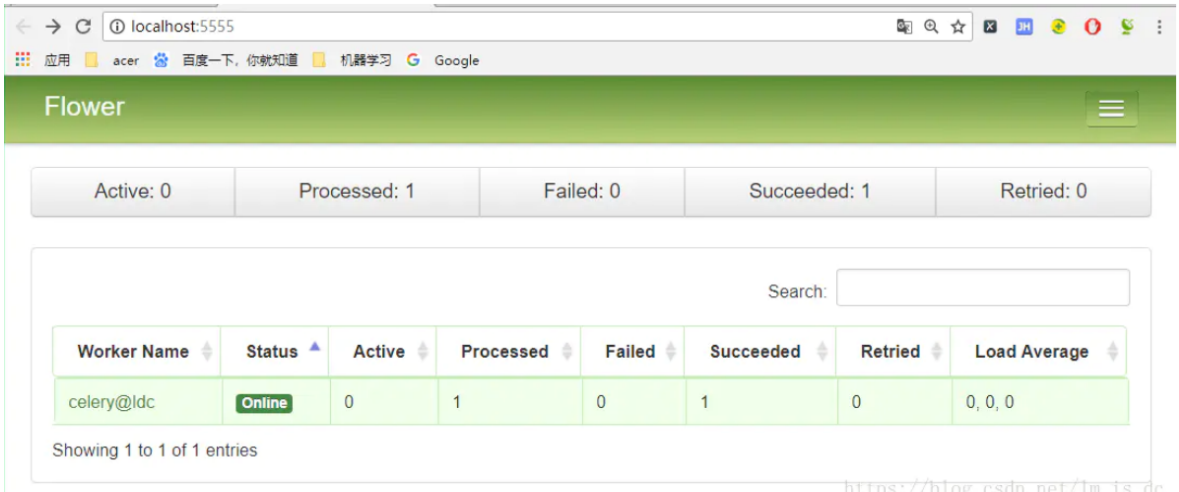
image



image

8、查看异步任务情况

Celery提供了一个工具flower，将各个任务的执行情况、各个worker的健康状态进行监控并以可视化的方式展现，如下图所示：



image

下实现的方式如下：

1. 安装flower:

pip install flower

```
2.  启动flower（默认会启动一个webserver，端口为5555）：

    在另一个Terminal中：

    ```python
python manage.py celery flower
```

3.进入http://localhost:5555即可查看。

9、一些错误解决方法

错误1：

```
Celery ValueError: not enough values to unpack (expected 3, got 0)
```

看别人描述大概就是说win10上运行celery4.x就会出现这个问题，解决办法如下,原理未知：

先安装一个`eventlet`

```
pip install eventlet
```

启动worker时加上参数`-P eventlet`

```
celery -A MySites worker --loglevel=DEBUG -P eventlet
```

三、定时任务

1、配置settings.py

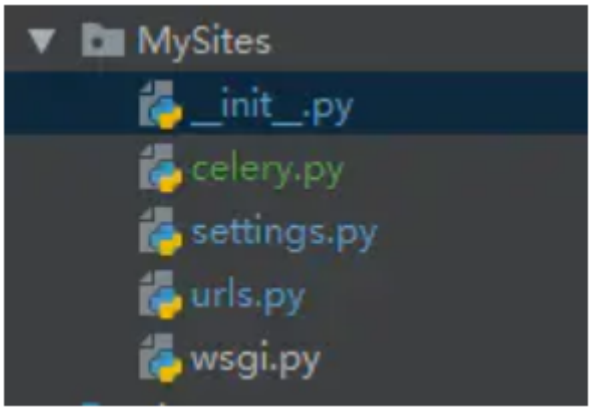
注册django_celery_beat应用:

```
INSTALLED_APPS = [
 #其它应用
 ...
 #django_celery_beat应用
 'django_celery_beat',
]
```

2、配置项目目录下的`__init__()`
```

```
from __future__ import absolute_import
from MySites.celery import app as celery_app

import pymysql
pymysql.install_as_MySQLdb()
```



image

3、修改celery.py文件

修改项目目录(和settings.py)同级的celery.py:

```
from __future__ import absolute_import

import os
from celery import Celery, platforms
from django.conf import settings

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'MySites.settings')

# MySites主应用名
app = Celery('mysites')
platforms.C_FORCE_ROOT = True

# 配置应用
app.conf.update(
    # 本地Redis服务器
    BROKER_URL=settings.BROKER_URL,

)

app.config_from_object('django.conf:settings')
app.autodiscover_tasks(settings.INSTALLED_APPS)

@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

4、修改adminx.py文件

我使用xadmin作为后台管理。增加以下代码:

```
#adminx.py
```

```
from __future__ import absolute_import, unicode_literals
from djcelery.models import (
  TaskState, WorkerState,
  PeriodicTask, IntervalSchedule, CrontabSchedule,
)

#celery

xadmin.site.register(IntervalSchedule) # 存储循环任务设置的时间
xadmin.site.register(CrontabSchedule) # 存储定时任务设置的时间
xadmin.site.register(PeriodicTask) # 存储任务
xadmin.site.register(TaskState) # 存储任务执行状态
xadmin.site.register(WorkerState) # 存储执行任务的worker
```

5、修改tasks.py文件

```
from __future__ import absolute_import
from MySites.celery import app
from celery import task, shared_task

@app.task
def start_running(info):
    print(info)
    print('--->>开始执行任务<<---')
    print('比如发送短信或邮件')
    print('>---任务结束---<')

@task
def pushMsg(uid,msg):
    print('推送消息',uid,msg)
    return True

@shared_task
def add(x,y):
    print('加法：',x + y)
    return x + y

@shared_task
def mul(x, y):
    print('乘法',x*y)
    return x * y
```

6、数据库迁移和创建管理员

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

8、启动服务器

```
python manage.py runserver
```

在浏览器打开http://127.0.0.1:10501/xadmin/

Crontabs:定时任务执行时间；

Intervals:简单的间隔执行时间，比如每10秒执行一次；

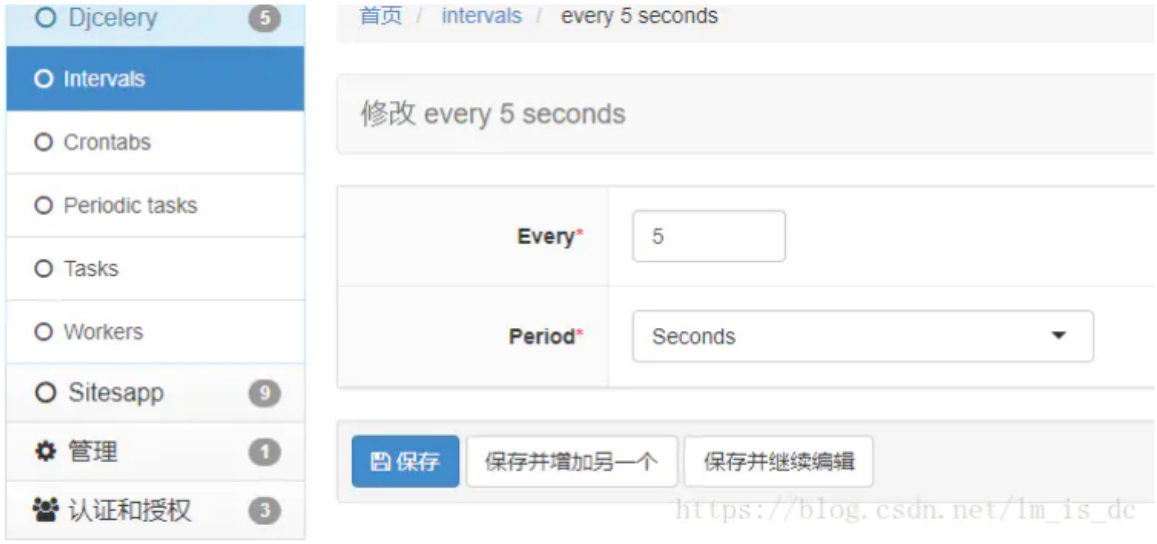
Periodic tasks:配置定时任务；

Tasks:任务监控；

Workers:运行的worker。

image

配置执行时间：



image

添加定时任务：

image

9、终端启动celery命令

```
# 启动
#linux下
celery -A MySites worker -B  # MySites为celery和setting所在文件夹名
#Windows下先启动celery-beat
celery -A MySites beat -l debug --max-interval=10
#然后再启动worker
celery -A MySites worker -l debug -P eventlet
#注意：在正式环境下把debug改为info

# 查看注册的task
celery -A MySites inspect registered

# flower监控celery
celery flower
ip:5555
```

注意：出现以下错误是windows不支持celery4.0以上版本，降低为3.1版本即可

```
SystemError: <class 'OSError'> returned a result with an error set
```

```
pip uninstall celery
pip install celery==3.1.22
```

补充：celery4.0以上版本不再支持 Microsoft Windows ， 不再支持使用Django ORM作为代理 。

10、服务器使用Supervisor后台运行Celery

```
pip install supervisor
```

我们可以使用echo_supervisord_conf命令得到supervisor配置模板，打开终端执行如下Linux shell命令：

```
echo_supervisord_conf > supervisord.conf
```

该命令输出文件到当前目录下（当然，你也可以指定绝对路径到具体位置），文件名为supervisord.conf 修改supervisord.conf文件，在文件最后加入：

```
[program:celery.worker]
;指定运行目录
directory=/home/你的项目名称
;运行目录下执行命令
command=celery -A 你的项目名称worker --loglevel info --logfile celery_worker.log

;启动设置
numprocs=1            ;进程数
autostart=true        ;当supervisor启动时,程序将会自动启动
autorestart=true      ;自动重启

;停止信号,默认TERM
;中断:INT (类似于Ctrl+C)(kill -INT pid)，退出后会将写文件或日志(推荐)
;终止:TERM (kill -TERM pid)
;挂起:HUP (kill -HUP pid),注意与Ctrl+Z/kill -stop pid不同
;从容停止:QUIT (kill -QUIT pid)
stopsignal=INT
;输出日志
stdout_logfile=celery_worker.log
stdout_logfile_maxbytes=10MB   ;默认最大50M
stdout_logfile_backups=10      ;日志文件备份数，默认为10

;错误日志
redirect_stderr=false          ;为true表示禁止监听错误
stderr_logfile=celery_worker_err.log
stderr_logfile_maxbytes=10MB
stderr_logfile_backups=10
```

启动supervisor输入如下命令，使用具体的配置文件执行： 先运行虚拟环境

```
supervisord -c supervisord.conf
```

关闭supervisor输入如下命令：

```
supervisorctl -c supervisord.conf shutdown
```

重启supervisor输入如下命令：

```
supervisorctl -c supervisord.conf reload
```