Celery

Celery是一个使用Python开发的分布式任务调度模块，因此对于大量使用Python构建的系统，使用起来方便。

Celery目前版本4.x，仅支持Django 1.8以上版本。Celery 3.1只可以支持Django1.8以下版本。

优点

- 简单：调用接口简单易用
- 高可用：客户端、Worker之间连接自动重试，Broker自身支持HA
- 快速：单个Celery进程每分钟可以数以百万计的任务
- 灵活：Celery的每一个部分都能扩展，或直接使用，或用户自己定义。

常见应用

Celery可以支持实时实时异步任务处理，也支持任务的定时调度。

1. 异步发邮件
   celery执行队列
2. 间隔半小时同步天气信息等
   celery定时操作

角色

- 任务Task：对应一个Python函数
- 队列Queue：待执行任务的队列
- 工人Worker：一个新的进程，负责执行任务
- 代理人Broker：负责调度，在任务环境中使用RabbitMQ、Redis等

Celery需要依靠RabbitMQ等作为消息代理，同时也支持Redis甚至是Mysql、Mongo等，当然，官方默认推荐的是RabbitMQ，如果使用Redis需要配置。

本次采用Redis来作为Broker，也是Redis存储任务结果。

安装

```
$ pip install celery==4.2.0

安装对redis的支持，并自动升级相关依赖。安装Redis作为Broker，通过配置把结果也放到Redis中
$ pip install -U "celery[redis]"
```

测试

Celery库使用前，必须初始化，所得实例叫做"应用application或app"。应用是线程安全的。不同应用在同一进程中，可以使用不同配置、不同组件、不同结果

```
from celery import Celery

app = Celery('mytask')
```

```
print(app)

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

print(add) # <@task: mytask.add of mytask at 0x1a30c4ad400>
print(app.tasks) # {'mytask.add': <@task: mytask.add of mytask at 0x1a30c4ad400>, .....省略}

print(add.name) # mytask.add
print(app.conf)
print(*list(app.conf.items()), sep='\n')
```

```
默认使用amqp连接到本地amqp://guest:**@localhost:5672//
```

本次使用Redis

Redis安装配置

使用Epel源的rpm安装

```
redis安装，使用提供的rpm安装，redis依赖jemalloc
# yum install jemalloc-3.6.0-1.el7.x86_64.rpm redis-3.2.12-2.el7.x86_64.rpm

# rpm -pql redis-3.2.12-2.el7.x86_64.rpm
/etc/logrotate.d/redis
/etc/redis-sentinel.conf
/etc/redis.conf
/usr/bin/redis-cli
/usr/bin/redis-sentinel
/usr/bin/redis-server
/usr/lib/systemd/system/redis-sentinel.service
/usr/lib/systemd/system/redis.service

编辑redis配置文件
# vi /etc/redis.conf
bind 192.168.142.131
protected-mode no
```

启动、停止redis服务

```
# systemctl start redis
# systemctl stop redis
```

broker配置使用

redis连接字符串格式 `redis://:password@hostname:port/db_number`

```
app.conf.broker_url = 'redis://192.168.142.131:6379/0'
# 意思是，指定服务器的redis端口6379，使用0号库
```

Celery使用

生成任务

```
# test1.py 注意模块名，后面命令中用
from celery import Celery
import time

app = Celery('mytask')
app.conf.broker_url = 'redis://192.168.142.131:6379/0' # 0号库存执行任务队列
# 重复执行问题解决
# 如果超过visibility_timeout，Celery会认为此任务失败
# 会重分配其他worker执行该任务，这样会造成重复执行。visibility_timeout这个值大一些
# 注意，如果慢任务执行时长超过visibility_timeout依然会多执行
app.conf.broker_transport_options = {'visibility_timeout': 43200} # 12 hours
app.conf.result_backend = 'redis://192.168.142.131:6379/1' # 1号库存执行结果

app.conf.update(
    enable_utc = True,
    timezone = 'Asia/Shanghai'
)

#@app.task
@app.task(name="firsttask")
#@app.task(ignore_result=True) # 不关心执行的结果
def add(x, y):
    print('in add. ~~~~~~')
    time.sleep(5)
    print('in add, timeout 5s. ~~~~~~')
    return x + y


if __name__ == '__main__':
    # 添加任务到Broker中
    print('in main. Send task')
    add.delay(4, 5)
    add.apply_async((10, 30), countdown=5)  # 5秒后执行
    print('end ~~~~~')
```

注意，上面代码执行，使用add.delay等加入任务到Redis中。在启动celery命令消费Redis的任务，执行并返回结果到Redis中。

```
# 增加任务的常用方法
T.delay(arg, kwarg=value)
always a shortcut to .apply_async.

T.apply_async((arg, ), {'kwarg': value})

T.apply_async(countdown=10)
executes 10 seconds from now.
```

执行任务

如果在Linux下可能出现下面的问题，可如下配置

```
from celery import platforms
# Linux下，默认不允许root用户启动celery，可使用下面的配置
platforms.C_FORCE_ROOT = True
```

使用命令执行Redis中的任务

```
-A APP, --app APP 指定app名称，APP是模块名
worker 指定worker工作
--loglevel 指定日志级别
-n 名称，%n指主机名
--concurrency 指定并发多进程数，缺省CPU数

$ celery -A test1 worker --loglevel=INFO --concurrency=5 -n worker1@%n
```

windows下可能下面问题

```
[ERROR/MainProcess] Task handler raised error: ValueError('not enough values to unpack (expected
3, got 0)',)
Traceback (most recent call last):
  File "e:\classprojects\venvs\p18test\lib\site-packages\billiard\pool.py", line 358, in
workloop
    result = (True, prepare_result(fun(*args, **kwargs)))
  File "e:\classprojects\venvs\p18test\lib\site-packages\celery\app\trace.py", line 537, in
_fast_trace_task
    tasks, accept, hostname = _loc
ValueError: not enough values to unpack (expected 3, got 0)
```

安装eventlet解决问题

```
$ pip install eventlet
```

重新执行任务

```
-P, --pool 指定进程池实现，默认prefork，windows下使用eventlet

$ celery -A test1 worker -P eventlet --loglevel=INFO --concurrency=5 -n worker1@%n

2任务，运行日志如下
[2018-03-27 10:08:23,598: INFO/MainProcess] Connected to redis://192.168.142.131:6379/0
[2018-03-27 10:08:23,607: INFO/MainProcess] mingle: searching for neighbors
[2018-03-27 10:08:24,661: INFO/MainProcess] mingle: all alone
[2018-03-27 10:08:24,676: INFO/MainProcess] worker@DESKTOP-D34H5HF ready.
[2018-03-27 10:08:24,689: INFO/MainProcess] pidbox: Connected to redis://192.168.142.131:6379/0.
[2018-03-27 10:08:25,102: INFO/MainProcess] Received task: firsttask[b642b80a-1180-4525-b3c7-
d4a89474d4de]
[2018-03-27 10:08:25,103: WARNING/MainProcess] in add. ~~~~~~
[2018-03-27 10:08:25,104: INFO/MainProcess] Received task: firsttask[9e624d2d-8116-46ce-af03-
4dd00ab93466]  ETA:[2018-03-27 10:07:50.606347+08:00]
[2018-03-27 10:08:25,105: WARNING/MainProcess] in add. ~~~~~~
[2018-03-27 10:08:30,101: WARNING/MainProcess] in add, timeout 5s. ~~~~~~
[2018-03-27 10:08:30,105: WARNING/MainProcess] in add, timeout 5s. ~~~~~~
[2018-03-27 10:08:30,106: INFO/MainProcess] Task firsttask[b642b80a-1180-4525-b3c7-d4a89474d4de]
succeeded in 5.014999999999418s: 9
[2018-03-27 10:08:30,109: INFO/MainProcess] Task firsttask[9e624d2d-8116-46ce-af03-4dd00ab93466]
succeeded in 5.0s: 40
在redis的1号库当中也能看到运行的结果
```

发邮件

在用户注册完激活时，或修改了用户信息，或遇到故障等情况时，都会发送邮件或发送短信息，这些业务场景不需要一直阻塞等待这些发送任务完成，一般都会采用异步执行。也就是说，都会向队列中添加一个任务后，直接返回。

Django中发送邮件需要在settings.py中配置，如下

```
# settings.py
# magedu.com设置
# SMTP
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = "smtp.exmail.qq.com"
EMAIL_PORT = 465 #缺省25，SSL的一般465
EMAIL_USE_SSL = True #缺省False
EMAIL_HOST_USER = "magetest@magedu.com"
EMAIL_HOST_PASSWORD = "Python123"
EMAIL_USE_TLS = False #缺省False
```

注意，不同邮箱服务器配置不太一样

邮件发送测试代码如下

```
#https://docs.djangoproject.com/en/1.11/topics/email/
# 使用 SMTP
# 这个函数测试时，可以写在任意模块中，需要时被调用就可以了
from django.core.mail import send_mail
```

```
def email():

    send_mail(
        'first test email',
        'Right here waiting',
        settings.EMAIL_HOST_USER,
        ['wei.xu@magedu.com'],
        fail_silently=False,
        html_message="<h1>test title<a href='http://www.magedu.com'
target='_blank'>magedu.com</a></h1>"
    )
    print('++++++++++++++++++')


# 测试用的视图函数
def test1(request):
    try:
        email()
    except Exception as e:
        print(e)
        return HttpResponseBadRequest()

    return HttpResponse('test1 ok')
```

Celery集成

新版Celery集成到Django方式改变了。

目录结构

```
blogpro
    blog
        __init__.py
        settings.py
        celery.py
        urls.py
    app1
        __init__.py
        tasks.py
        view.py
        models.py
```

在Django全局目录中（settings.py所在目录）

1、定义一个celery.py

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

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

app = Celery('blog')

# Using a string here means the worker doesn'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()

# 增加以下内容配置
app.conf.broker_url = 'redis://192.168.142.131:6379/0'
# 如果超过visibility_timeout，Celery会认为此任务失败
# 会重分配其他worker执行该任务，这样会造成重复执行。visibility_timeout这个值大一些
# 注意，如果慢任务执行时长超过visibility_timeout依然会多执行
app.conf.broker_transport_options = {'visibility_timeout': 43200} # 12 hours
app.conf.result_backend = 'redis://192.168.142.131:6379/1' #  执行结果存储

app.conf.update(
    enable_utc = True,
    timezone = 'Asia/Shanghai'
)
```

2、修改__init__.py

```
from __future__ import absolute_import, unicode_literals
# 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',)
```

在user应用下

1、urls.py

```
from django.conf.urls import url
from .views import reg, login, test, logout, test1, testsendmail

urlpatterns = [
    url(r'^reg$', reg),
    url(r'^login$', login),
    url(r'^test$', test),
    url(r'^test1$', test1),
    url(r'^logout$', logout),
    url(r'^mail$', testsendmail) # 测试发邮件
]
```

2、views.py

```
from .tasks import sendmail

def testsendmail(request):
    try:
        # 用户注册了，注册信息保存了，然后发邮件给他，里面写
        sendmail.delay() # 阻塞效果 => 非阻塞的异步调用
    except Exception as e:
        print(e, '~~~~~~~~~~~~~~~~')
        return HttpResponseBadRequest()
    return HttpResponse('邮件已发送，请等待5分钟查收')
```

3、tasks.py

```
# Create your tasks here
from __future__ import absolute_import, unicode_literals
from blog.celery import app

from django.core.mail import send_mail
from django.conf import settings
import datetime

# celery -A blog worker -P eventlet -l INFO -c 5 -n worker@%n
@app.task(name='sendemail')
def sendmail():
    send_mail(
        '发邮件测试',
        '测试用',
        settings.EMAIL_HOST_USER, #'from@example.com', # 谁发的
        ['wei.xu@magedu.com'], # 发给谁们
        fail_silently=False,
        html_message="<p>这是一封测试邮件 {:%Y%m%d-%H:%M:%S}<br /><a href='{}' target='_blank'>官
网</a></p>".format(
            datetime.datetime.now(), 'http://www.magedu.com')
    )
    print('++++++++++++++++++')
```

访问http://127.0.0.1:8000/user/mail 会调用test1视图函数，会执行email.delay()，会在redis中增加任务。

执行任务

```
$ celery -A blog -P eventlet worker --loglevel=INFO --concurrency=5 -n worker@%n
```