一、Celery简介

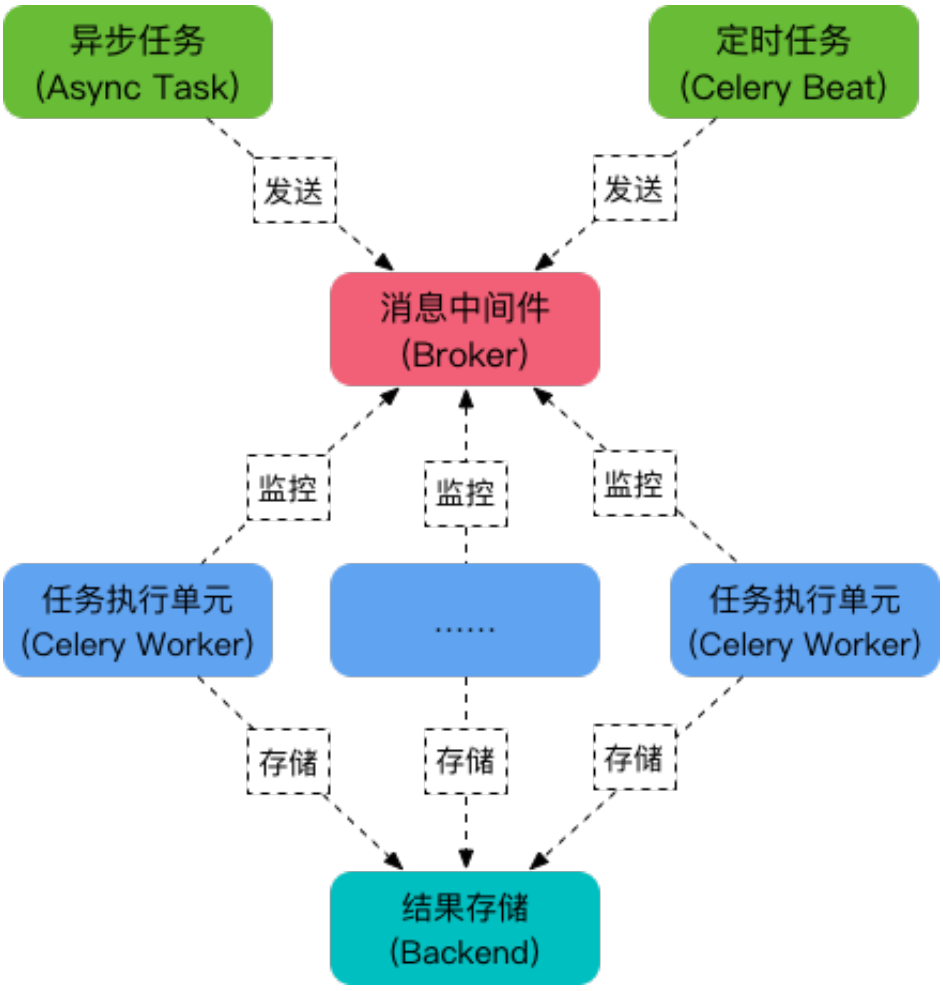
Celery 是一个 基于python开发的异步任务队列/基于分布式消息传递的作业队列，通过它可以轻松的实现任务的异步处理。它侧重于实时操作，但对调度支持也很好。Celery用于生产系统每天处理数以百万计的任务。Celery是用Python编写的，但该协议可以在任何语言实现。它也可以与其他语言通过webhooks实现。Celery建议的消息队列是RabbitMQ，但提供支持Redis, Beanstalk, MongoDB, CouchDB, 和数据库（使用SQLAlchemy的或Django的 ORM） 。Celery是易于集成Django, Pylons 和 Flask，使用 django-celery, celery-pylons and Flask-Celery 附加包即可。它的特点：

- 方便查看定时任务的执行情况, 如 是否成功, 当前状态, 执行任务花费的时间等.
- 使用功能齐备的管理后台或命令行添加,更新,删除任务.
- 方便把任务和配置管理相关联.
- 可选多进程, Eventlet 和 Gevent 三种模型并发执行.
- 提供错误处理机制.
- 提供多种任务原语, 方便实现任务分组,拆分,和调用链.
- 支持多种消息代理和存储后端.
- Celery 是语言无关的，它提供了python 等常见语言的接口支持.

celery官方文档：http://docs.jinkan.org/docs/celery/getting-started/first-steps-with-celery.html#first-steps

二、Celery的相关概念

celery架构图



- task 就是任务，包括异步任务和定时任务
- broker 中间人，接收生产者发来的消息即Task，将任务存入队列。任务的消费者是Worker。Celery本身不提供队列服务，推荐用Redis或RabbitMQ实现队列服务。
- worker 执行任务的单元，它实时监控消息队列，如果有任务就获取任务并执行它。
- backend 用于存储任务的执行结果。Celery支持以不同方式存储任务的结果，包括AMQP, redis，memcached, mongodb，SQLAlchemy, Django ORM，Apache Cassandra, IronCache 等。
- beat 定时任务调度器，根据配置定时将任务发送给Broler。

三、应用场景

- 异步调用：那些用户不关心的但是又存在在我们API里面的操作 我们就可以用异步调用的方式来优化（发送邮件 或者上传头像）
- 定时任务：定期去统计日志，数据备份，或者其他的统计任务

四、Celery的安装

- 安装

```
pip install celery==4.4.0
pip install celery-with-redis==3.0
#django-celery-results库基于 Django ORM实现了结果存储后端
pip install django-celery-results==1.2.0
pip install django-celery==3.3.1
```

- 配置

  在 settings.py文件中设置

```
ALLOWED_HOSTS = ['*']
INSTALLED_APPS = (
      ...
      'celery',
      'django_celery_results',  #把 django_celery_results 加到
INSTALLED_APPS 中
      '自己的APP',
      'djcelery'
    }

BROKER_URL='redis://localhost:6379/5'
CELERY_RESULT_BACKEND = 'django-db'
CELERY_TASK_SERIALIZER = 'json'              # 任务序列化和反序列化使用
json
CELERY_RESULT_SERIALIZER = 'json'            # 结果序列化为json


# 邮件配置
# smtp服务的邮箱服务器
EMAIL_HOST = 'smtp.126.com'
# smtp服务固定的端口是25
EMAIL_PORT = 25   # 也有可能是465
#发送邮件的邮箱
EMAIL_HOST_USER = 'landmark_csl@126.com'
#在邮箱中设置的客户端授权密码
EMAIL_HOST_PASSWORD = 'csl111'
#收件人看到的发件人 <此处要和发送邮件的邮箱相同>
EMAIL_FROM = 'python<landmark_csl@163.com>'

import djcelery
```

```
djcelery.setup_loader()
```

- 创建celery实例

  在settings.py的同级目录下新建celery.py

```
from __future__ import absolute_import #绝对路径导入
from celery import Celery
from django.conf import settings
import os

#设置系统的环境配置用的是Django的
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "工程名字.settings")

#实例化celery
app = Celery('mycelery')

app.conf.timezone = "Asia/Shanghai"

#指定celery的配置来源 用的是项目的配置文件settings.py
app.config_from_object("django.conf:settings")

#让celery 自动去发现我们的任务（task）
app.autodiscover_tasks() #你需要在app目录下 新建一个叫tasks.py（一定不要写错）文件
```

在settings.py同级目录下的**init**.py加入

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

五、Celery的使用

1、创建任务

在需要使用异步任务的APP目录下新建tasks.py

```
from celery.signals import task_success
from celery import shared_task
```

```
import time

from django.core.mail import send_mail

from day10.settings import EMAIL_HOST_USER


@shared_task
def hello_celery(loop):
    for i in range(loop):
        print('hello')
        time.sleep(2)

@shared_task
def add(a,b):
    return a+b


@shared_task
def my_send():
    send_mail("hello","hello",EMAIL_HOST_USER,["313728420@qq.com"])

@task_success.connect(sender=add)
def task_done_handler(sender=None, result=None, **kwargs):
    # print('after_task_publish for task id {body[id]}'.format(
    #     body=body,
    # ))
    print("add-----")
    print(result)
```

2、调用

在views.py内的调用

```
任务函数名.delay(参数，，，，)
```

3、生成数据库表

```
python manage.py migrate django_celery_results
```

4、启动worker

```
celery -A 你的工程名 worker -l info
```

- 注意：修改tasks.py的内容后 要重启celery的服务

5 、获取任务执行结果

异步任务执行完毕后，会自动触发信号：

- before_task_publish
- after_task_publish
- task_prerun
- task_postrun
- task_success
- task_failure
- task_revoked

```
from celery.signals import task_success
@task_success.connect(sender=add)
def task_done_handler(sender=None,  result=None):
    print(result)
```

六、定时任务和计划任务

- 定时任务
    - 启动： celery -A 你的工程名称 beat -l info

在settings.py文件添加 CELERYBEAT_SCHEDULE = { 'schedule-test': { 'task': 'app的名字.tasks.hello_celery', 'schedule': timedelta(seconds=3), 'args': (2,) },

}

- 计划任务时间

```
#setting.py
from celery.schedules import crontab

CELERYBEAT_SCHEDULE = {
    "every-ten-second-run-my_task": {
        "task": "t07.tasks.my_task",
        "schedule": crontab(minute="01", hour="15"),
        "args": (2,)
    }
}
```

- 坑:
  - **我们启动定时任务服务时 也要先开启worker**

    如果只开启定时服务 没有开启worker服务 那么定时任务会被放入任务队列，但是由于没有干活儿的worker 那么任务是不会被执行，当worker服务被启动后 会立刻去任务队列领任务并执行
- 你的任务一定要确保是可以正常执行的

七、其它

1、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，

1. 安装flower:

   ```
   pip install flower
   ```

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

   ```
   celery flower --broker=redis://localhost:6379/5
   ```

3. 即可查看

   http://localhost:5555

2、内存泄漏

1. 说明

   长时间运行Celery有可能发生内存泄露，可以像下面这样设置

2. 示例代码

```
CELERYD_MAX_TASKS_PER_CHILD = 1000 # 每个worker执行了多少任务就会死掉
```

常用配置清单

1. 说明

2. 配置信息

```
#from kombu import Queue, Exchange
# 设置Broker和backend
BROKER_URL = 'redis://127.0.0.1:6379/0'
# 将数据存放到redis1数据库，redis默认有16个数据库
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/1'


CELERY_TASK_SERIALIZER = 'json'              # 任务序列化和反序列化使用json
CELERY_RESULT_SERIALIZER = 'json'            # 结果序列化为json
CELERY_ACCEPT_CONTENT = ['json']             # 分布式接受数据的类型为json
CELERY_TIMEZONE = 'Asia/Shanghai'             #使用中国上海时区
CELERY_ENABLE_UTC = True

CELERY_TASK_RESULT_EXPIRES = 60 * 60 * 24  # 后端存储任务超过一天，则自动清理数据，单位为秒
CELERYD_MAX_TASKS_PER_CHILD = 1000           # 每个worker最多执行1000个任务就会被销毁，可防止内存泄露
```