概念：

在一个应用服务中，对于时效性要求没那么高的业务场景，我们没必要等到所有任务执行完才返回结果，例如用户注册场景中，保存了用户账号密码之后．就可以立即返回，后续的账号激活邮件，可以用一种异步的形式去处理，这种异步操作可以用队列服务来实现．否则，如果等到邮件发送成功可能几秒过去了．

Celery是Python语言实现的分布式队列服务，除了支持即时任务，还支持定时任务，Celery有5个核心角色．

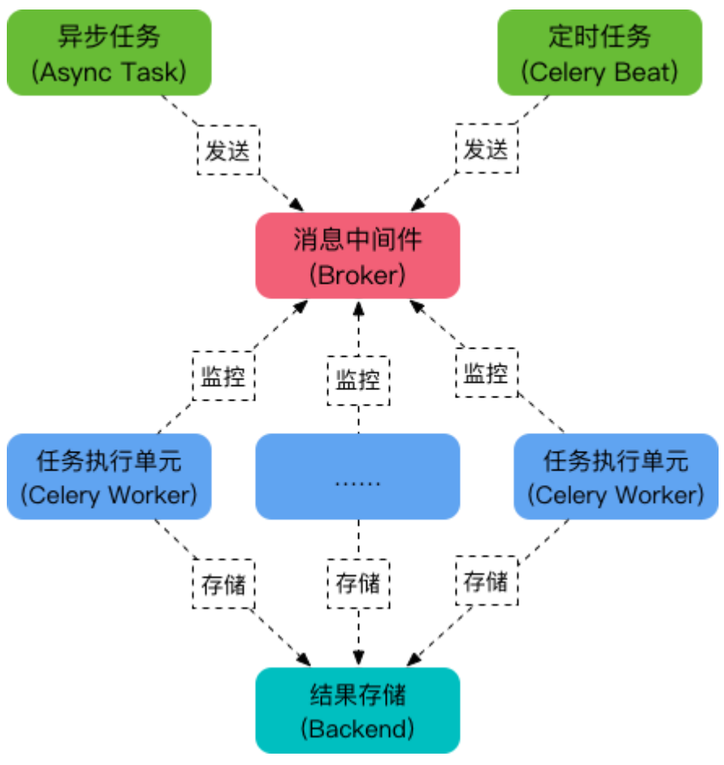
1.Task
任务(Task)就是你要做的事情，例如一个注册流程里面有很多任务，给用户发验证邮件就是一个任务，这种耗时任务可以交给Celery去处理；还有一种任务是定时任务，比如每天定时统计网站的注册人数，这个也可以交给Celery周期性的处理．

2.Broker
Broker的中文意思是经纪人，指为市场上买卖双方提供中介服务的人．在是Celery中它介于生产者和消费者之间经纪人，这个角色相当于数据结构中的队列．例如一个Web系统中，生产者是处理核心业务的Web程序，业务中可能会产生一些耗时的任务；比如短信 生产者会将任务发送给Broker，就是把这个任务暂时放到队列中，等待消费者来处理．消费者是Worker，是专门用于执行任务的后台服务．Worker将实时监控队列中是否有新的任务，如果有就拿出来进行处理．Celery本身不提供队列服务，一般用Redis或者RabbitMQ来扮演Broker的角色．

3.Worker
Worker 就是那个一直在后台执行任务的人，也称为任务的消费者，它会实时地监控队列中有没有任务，如果有就立即取出来执行．

4.Beat
Beat是一个定时任务调度器，它会根据配置定时将任务发送给Broker，等待Worker来消费．

5.Backend
Backend用于保存任务的执行结果，每个任务都有返回值，比如发送邮件的服务会告诉我们有没有发送成功，这个结果就是存在Backend中．



In [ ]:

1.1 创建Celery实例

In [ ]:
```
import time
from celery import Celery,  platforms

broker = 'redis://127.0.0.1:6379'
backend = 'redis://127.0.0.1:6379'
platforms.C_FORCE_ROOT = True  #允许在root下运行

app = Celery(__file__, broker=broker, backend=backend)

@app.task
def add(x, y):
    time.sleep(5)
    return  x + y
```

pip install redis==2.10.6 #推荐安装该版本python-redis 否则会报错

上面的代码做了几件事：

创建了一个Celery实例app
指定消息中间件用redis，URL为redis://127.0.0.1:6379
指定存储用 redis，URL为redis://127.0.0.1:6379
创建了一个Celery任务add，当函数被@app.task装饰后，就成为可被 Celery 调度的任务．

启动Celery任务

```
celery worker -A tasks --loglevel=info
```

参数 -A 指定了Celery实例的位置，本例是在 tasks.py 中，Celery会自动在该文件中寻找Celery对象实例
参数 --loglevel 指定了日志级别，默认为 warning，也可以使用 -l info 来表示．

任务调度：

现在，我们可以在应用程序中使用delay()方法来调用任务．

```
In [10]: from task import add
In [11]: t = add.delay(1, 2)
```

在上面，我们从tasks.py文件中导入了add任务对象，然后使用delay()方法将任务发送到消息中间件(Broker)，Celery Worker 进程监控到该任务后,就会进行执行，我们将窗口切换到Worker的启动窗口，会看到多了两条日志：

```
[2018-10-19 05:46:04,497: INFO/MainProcess] Received task: task.add[9307b28d-fba9-4084-bd0d-5c724d68dc3a]
[2018-10-19 05:46:09,513: INFO/ForkPoolWorker-1] Task task.add[9307b28d-fba9-4084-bd0d-5c724d68dc3a] succeeded in 5.013509025s: 3
```

以上说明任务已经被调度并执行成功．

```
In [2]: t = add.delay(1, 2)

In [3]: t
Out[3]: <AsyncResult: 791368fe-dfa8-4612-8c87-9235a3cd5a87>

In [4]: t.ready()  ##判断任务是否执行完毕
Out[4]: False

In [5]: t.ready()
Out[5]: False
```

```
In [8]: t.ready()
Out[8]: True

In [9]: t.get()  ##获取任务执行结果
Out[9]: 3
```

可以看到，虽然任务函数add需要等待5秒才返回执行结果，但由于它是一个异步任务，不会阻塞当前的主程序.

1.2 配置文件

在上面的例子中， 我们直接把Broker和 Backend的配置写在了程序当中，更好的做法是将配置项统一写入到一个配置文件中.

1.2.1目录结构：

```
celery_demo                    # 项目根目录
    ├──── celery_app             # 存放 celery 相关文件
    │    ├──── __init__.py
    │    ├──── config.py    # 配置文件
    │    ├──── task1.py           # 任务文件 1
    │    └──── task2.py           # 任务文件 2
    └──── client.py              # 应用程序
```

1.2.2 config.py文件中内容

```
BROKER_URL = 'redis://127.0.0.1:6379'               # 指定 Broker
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379/0'  # 指定 Backend

CELERY_TIMEZONE='Asia/Shanghai'                     # 指定时区，默认是 UTC
CELERY_IMPORTS = (                                  # 指定导入的任务模块
    'celery_app.task1',
    'celery_app.task2'
)
```

1.2.3 __init__.py文件内容

```
# -*- coding: utf-8 -*-
from celery import Celery,platforms
platforms.C_FORCE_ROOT = True
app = Celery('demo')                                    # 创建 Celery 实例
app.config_from_object('celery_app.config')   # 通过 Celery 实例加载配置模块
```

1.2.4 task1.py

```
import time
from celery_app import app
@app.task
def add(x, y):
    time.sleep(2)
    return x + y
```

1.2.5 task2.py

```
from celery_app import app
import time
@app.task
def say():
    time.sleep(2)
    return 'helo'
```

1.2.6 client.py

```
from celery_app import task1
from celery_app import task2

task1.add.delay(1, 2)
task2.say.delay()
```

1.2.7 启动 celery worker

```
(websocket) [root@gitlab celery_demo]# celery -A celery_app worker -l info
```

1.2.8 执行client.py

```
python client.py
```

1.2.9 运行python client.py后它会发送两个异步任务到Broker，在Worker的窗口我们可以看到如下输出：

```
[2018-10-19 06:52:31,389: INFO/MainProcess] Received task: celery_app.task1.add[b32962fe-dd61-443f-bc87-e666db957f24]
[2018-10-19 06:52:31,391: INFO/MainProcess] Received task: celery_app.task2.say[d945e419-aa93-4a2c-aec4-105f81031a64]
[2018-10-19 06:52:33,394: INFO/ForkPoolWorker-2] Task celery_app.task2.say[d945e419-aa93-4a2c-aec4-105f81031a64] succeeded in 2.00137619101s: 'helo'
[2018-10-19 06:52:33,394: INFO/ForkPoolWorker-1] Task celery_app.task1.add[b32962fe-dd61-443f-bc87-e666db957f24] succeeded in 2.00418746s: 3
```

1.3 定时任务

Celery 除了可以执行异步任务，也支持执行周期性任务(Periodic Tasks)，或者说定时任务，Celery Beat 进程通过读取配置文件的内容，周期性地将定时任务发往任务队列.

1.3.1 修改配置文件，增加定时任务

```
from celery.schedules import crontab
from datetime import timedelta
BROKER_URL = 'redis://127.0.0.1:6379'
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379'
CELERY_TIMEZONE='Asia/Shanghai'

CELERY_IMPORTS = (
    'celery_app.task1',
    'celery_app.task2'
)


CELERYBEAT_SCHEDULE = {
    "task1": {
        "task": "celery_app.task1.add",
        "schedule": timedelta(seconds=1),
        "args":(1, 2),
    },

    "task2": {
        "task": "celery_app.task2.say",
        "schedule": timedelta(seconds=2),
        "args":(),
    },
```

```
}
```

1.3.2 启动celery

```
celery -B -A celery_app worker --loglevel=info
```

1.3.3 在worker窗口查看任务输出

```
[2018-10-19 07:40:31,966: INFO/ForkPoolWorker-2] Task celery_app.task1.add[6b216f23-036a-48ed-b14f-e252bf3f1ffb] succeeded in 2.001080302s: 3
[2018-10-19 07:40:31,968: INFO/ForkPoolWorker-3] Task celery_app.task2.say[28246df4-0eff-49f0-b468-f92b85fe97b3] succeeded in 2.00303293001s: 'helo'
```

##定时任务遇到的问题就是不能对托管的定时任务做动态更新, 需要重启 celery beat ..