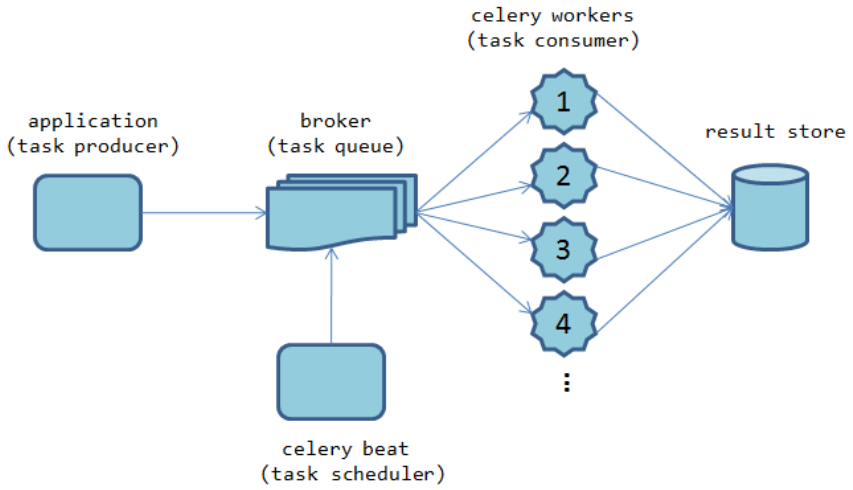
Celery: 基于 Python 的开源分布式任务调度模块

Celery 是一个用 Python 编写的分布式的任务调度模块，它有着简明的 API，并且有丰富的扩展性，适合用于构建分布式的 Web 服务。

图 1. Celery 的模块架构



Celery 的模块架构较为简洁，但是提供了较为完整的功能：

任务生产者 (task producer)

任务生产者 (task producer) 负责产生计算任务，交给任务队列去处理。在 Celery 里，一段独立的 Python 代码、一段嵌入在 Django Web 服务里的一段请求处理逻辑，只要是调用了 Celery 提供的 API，产生任务并交给任务队列处理的，我们都可以称之为任务生产者。

任务调度器 (celery beat)

Celery beat 是一个任务调度器，它以独立进程的形式存在。Celery beat 进程会读取配置文件的内容，周期性地将执行任务的请求发送给任务队列。Celery beat 是 Celery 系统自带的任务生产者。系统管理员可以选择关闭或者开启 Celery beat。同时在一个 Celery 系统中，只能存在一个 Celery beat 调度器。

任务代理 (broker)

任务代理方负责接受任务生产者发送过来的任务处理消息，存进队列之后再进行调度，分发给任务消费方 (celery worker)。因为任务处理是基于 message(消息) 的，所以我们一般选择 RabbitMQ、Redis 等消息队列或者数据库作为 Celery 的 message broker。

任务消费方 (celery worker)

Celery worker 就是执行任务的一方，它负责接收任务处理中间方发来的任务处理请求，完成这些任务，并且返回任务处理的结果。Celery worker 对应的就是操作系统中的一个进程。Celery 支持分布式部署和横向扩展，我们可以在多个节点增加 Celery worker 的数量来增加系统的高可用性。在分布式系统中，我们也可以在不同节点上分配执行不同任务的 Celery worker 来达到模块化的目的。

结果保存

Celery 支持任务处理完后将状态信息和结果的保存，以供查询。Celery 内置支持 rpc, Django ORM，Redis，RabbitMQ 等方式来保存任务处理后的状态信息。

构建第一个 Celery 程序

在我们的第一个 Celery 程序中，我们尝试在 Celery 中构建一个“将新鲜事通知到朋友”的任务，并且尝试通过编写一个 Python 程序来启动这个任务。

安装 Celery

```
1  Pip install celery
```

选择合适的消息代理中间件

Celery 支持 RabbitMQ、Redis 甚至其他数据库系统作为其消息代理中间件，在本文中，我们选择 RabbitMQ 作为消息代理中间件。

```
1  sudo apt-get install rabbitmq-server
```

创建 Celery 对象

Celery 对象是所有 Celery 功能的入口，所以在开始其它工作之前，我们必须先定义我们自己的 Celery 对象。该对象定义了任务的具体内容、任务队列的服务地址、以及保存任务执行结果的地址等重要信息。

```
1
2  # notify_friends.py
3  from celery import Celery
4  import time
5  app = Celery('notify_friends', backend='rpc://', broker='amqp://localhost')
6
7  @app.task
8  def notify_friends(userId, newsId):
9    print 'Start to notify_friends task at {0}, userID:{1} newsID:{2}'.format(time.c
10   time.sleep(2)
11   print 'Task notify_friends succeed at {0}'.format(time.ctime())
     return True
```

在本文中，为了模拟真实的应用场景，我们定义了 notify_friends 这个任务，它接受两个参数，并且在输出流中打印出一定的信息，

创建 Celery Worker 服务进程

在定义完 Celery 对象后，我们可以创建对应的任务消费者--Celery worker 进程，后续的任务处理请求都是由这个 Celery worker 进程来最终执行的。

```
1  celery -A celery_test worker --loglevel=info
```

在 Python 程序中调用 Celery Task

我们创建一个简单的 Python 程序，来触发 notify_friends 这个任务。

```
1  # call_notify_friends.py
```

```
2
3   from notify_friends import notify_friends
4   import time
5
6   def notify(userId, messageId):
7     result = notify_friends.delay(userId, messageId)
8     while not result.ready():
9     time.sleep(1)
10    print result.get(timeout=1)
11
12  if __name__ == '__main__':
13    notify('001', '001')
```

我们在 call_notify_friends.py 这个程序文件中，定义了 Notify 函数，它调用了我们之前定义的 notify_friends 这个 API，来发送任务处理请求到任务队列，并且不断地查询等待来获得任务处理的结果。

Celery worker 中的 log 信息：

```
1
2   [tasks]
3     . celery_test.notify_friends
4
5   [2015-11-16 15:02:31,113: INFO/MainProcess] Connected to amqp://guest:**@127.0.0.1
6   [2015-11-16 15:02:31,122: INFO/MainProcess] mingle: searching for neighbors
7   [2015-11-16 15:02:32,142: INFO/MainProcess] mingle: all alone
8   [2015-11-16 15:02:32,179: WARNING/MainProcess] celery@yuwenhao-VirtualBox ready.
9   [2015-11-16 15:04:45,474: INFO/MainProcess] Received task:
    celery_test.notify_friends[3f090a76-7678-4f9c-a37b-ceda59600f9c]
10  [2015-11-16 15:04:45,475: WARNING/Worker-2] Start to notify_friends task at
    Mon Nov 16 15:04:45 2015, userID:001 newsID:001
11  [2015-11-16 15:04:47,477: WARNING/Worker-2] Task notify_friends succeed at Mon Nov
12  [2015-11-16 15:04:47,511: INFO/MainProcess] Task celery_test.notify_friends
13                                          [3f090a76-7678-4f9c-a37b-ceda59600f9c] suc
14
```

我们可以看到，Celery worker 收到了 Python 程序的 notify_friends 任务的处理请求，并且执行完毕。

利用调度器创建周期任务

在我们第二个 Celery 程序中，我们尝试构建一个周期性执行“查询当前一小时最热门文献”的任务，每隔 100 秒执行一次，并将结果保存起来。后续的搜索请求到来后可以直接返回已有的结果，极大优化了用户体验。

创建配置文件

Celery 的调度器的配置是在 CELERYBEAT_SCHEDULE 这个全局变量上配置的，我们可以将配置写在一个独立的 Python 模块，在定义 Celery 对象的时候加载这个模块。我们将 select_populate_book 这个任务定义为每 100 秒执行一次。

```
1   # config.py
2   from datetime import timedelta
3
```

```
CELERYBEAT_SCHEDULE = {
  'select_populate_book': {
  'task': 'favorite_book.select_populate_book',
  'schedule': timedelta(seconds=100),
  },
}
```

创建 Celery 对象

在 Celery 对象的定义里，我们加载了之前定义的配置文件，并定义了 select_populate_book 这个任务。

```

#favorite_book.py
from celery import Celery
import time

app = Celery('select_populate_book', backend='rpc://', broker='amqp://localhost')
app.config_from_object('config')

@app.task
def select_populate_book():
  print 'Start to select_populate_book task at {0}'.format(time.ctime())
  time.sleep(2)
  print 'Task select_populate_book succeed at {0}'.format(time.ctime())
  return True
```

启动 Celery worker

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

启动 Celery beat

启动 Celery beat 调度器，Celery beat 会周期性地执行在 CELERYBEAT_SCHEDULE 中定义的任务，即周期性地查询当前一小时最热门的书籍。

```
celery -A favorite_book beat


yuwenhao@yuwenhao:~$ celery -A favorite_book beat
celery beat v3.1.15 (Cipater) is starting.
__ - ... __ - _
Configuration ->
  . broker -> amqp://guest:**@localhost:5672//
  . loader -> celery.loaders.app.AppLoader
  . scheduler -> celery.beat.PersistentScheduler
  . db -> celerybeat-schedule
  . logfile -> [stderr]@%INFO
  . maxinterval -> now (0s)
[2015-11-16 16:21:15,443: INFO/MainProcess] beat: Starting...
[2015-11-16 16:21:15,447: WARNING/MainProcess] Reset:
Timezone changed from 'UTC' to None
[2015-11-16 16:21:25,448: INFO/MainProcess] Scheduler:
Sending due task select_populate_book (favorite_book.select_populate_book)
[2015-11-16 16:21:35,485: INFO/MainProcess] Scheduler:
Sending due task select_populate_book (favorite_book.select_populate_book)
[2015-11-16 16:21:45,490: INFO/MainProcess] Scheduler:
```

```
Sending due task select_populate_book (favorite_book.select_populate_book)
```

我们可以看到，Celery beat 进程周期性地将任务执行请求 select_populate_book 发送至任务队列。

```
yuwenhao@yuwenhao:~$ celery -A favorite_book worker --loglevel=info
[2015-11-16 16:21:11,560: WARNING/MainProcess]
/usr/local/lib/python2.7/dist-packages/celery/apps/worker.py:161: CDeprecationWarn
Starting from version 3.2 Celery will refuse to accept pickle by default.

The pickle serializer is a security concern as it may give attackers
the ability to execute any command. It's important to secure
your broker from unauthorized access when using pickle, so we think
that enabling pickle should require a deliberate action and not be
the default choice.

If you depend on pickle then you should set a setting to disable this
warning and to be sure that everything will continue working
when you upgrade to Celery 3.2::

    CELERY_ACCEPT_CONTENT = ['pickle', 'json', 'msgpack', 'yaml']

You must only enable the serializers that you will actually use.


    warnings.warn(CDeprecationWarning(W_PICKLE_DEPRECATED))


    -------------- celery@yuwenhao-VirtualBox v3.1.15 (Cipater)
---- **** -----
--- * *** * -- Linux-3.5.0-23-generic-x86_64-with-Ubuntu-12.04-precise
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app: select_populate_book:0x1b219d0
- ** ---------- .> transport: amqp://guest:**@localhost:5672//
- ** ---------- .> results: rpc://
- *** --- * --- .> concurrency: 2 (prefork)
-- ******* ----
--- ***** ----- [queues]
    -------------- .> celery exchange=celery(direct) key=celery


[tasks]
    . favorite_book.select_populate_book

[2015-11-16 16:21:11,579: INFO/MainProcess] Connected to amqp://guest:**@127.0.0.1
[2015-11-16 16:21:11,590: INFO/MainProcess] mingle: searching for neighbors
[2015-11-16 16:21:12,607: INFO/MainProcess] mingle: all alone
[2015-11-16 16:21:12,631: WARNING/MainProcess] celery@yuwenhao-VirtualBox ready.
[2015-11-16 16:21:25,459: INFO/MainProcess] Received task:
favorite_book.select_populate_book[515f7c55-7ff0-4fcf-bc40-8838f69805fd]
[2015-11-16 16:21:25,460: WARNING/Worker-2]
Start to select_populate_book task at Mon Nov 16 16:21:25 2015
[2015-11-16 16:21:27,462: WARNING/Worker-2]
Task select_populate_book succeed at Mon Nov 16 16:21:27 2015
[2015-11-16 16:21:27,475: INFO/MainProcess] Task favorite_book.select_populate_boo
```

```
[515f7c55-7ff0-4fcf-bc40-8838f69805fd] succeeded in 2.015802141s: True
[2015-11-16 16:21:35,494: INFO/MainProcess] Received task:
favorite_book.select_populate_book[277d718a-3435-4bca-a881-a8f958d64aa9]
[2015-11-16 16:21:35,498: WARNING/Worker-1]
Start to select_populate_book task at Mon Nov 16 16:21:35 2015
[2015-11-16 16:21:37,501: WARNING/Worker-1]
Task select_populate_book succeed at Mon Nov 16 16:21:37 2015
[2015-11-16 16:21:37,511: INFO/MainProcess]
Task favorite_book.select_populate_book
[277d718a-3435-4bca-a881-a8f958d64aa9] succeeded in 2.014368786s: True
```

我们可以看到，任务 select_populate_book 的 Celery worker 周期性地收到 Celery 调度器的任务的处理请求，并且运行该任务。

结束语

任务队列技术可以满足 Web 服务系统后台任务管理和调度的需求，适合构建分布式的 Web 服务系统后台。Celery 是一个基于 Python 的开源任务队列系统。它有着简明的 API 以及良好的扩展性。本文首先介绍了队列技术的基本原理，然后介绍了 Celery 的模块架构以及工作原理。最后，本文通过实例介绍了如何在 Python 程序中调用 Celery API 并通过 Celery 任务队列来执行任务，以及如何通过 Celery beat 在 Celery 任务队列中创建周期性执行的任务。希望本文可以对 Web 后台开发者、以及 Celery 的初学者有所帮助。