Celery

# Asynchronous

Asynchronous means controlling the timing of operations to be performed by the use of signals sent when the previous operation is completed rather than at regular intervals.

# RabbitMQ

It provides a Queue which performs tasks in the background by running a server of its own.

# Celery

Celery provides an easy way of connecting and sending tasks to the Queue (RabbitMQ).

Celery is Python Task-Queue systems that handle distribution of tasks on workers across threads or network nodes.

# Installation

pip install celery

# proj/proj/celery.py

**import os**

**from celery import Celery**

# set the default Django settings module for the 'celery' program.

**os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'proj.settings')**

**app = Celery('proj')**

# 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()**

**@app.task(bind=True)**

**def debug\_task(self):**

**print(f'Request: {self.request!r}')**

# proj/proj/\_\_init\_\_.py:

from .celery import app as celery\_app

\_\_all\_\_ = ('celery\_app',)

# app/tasks.py:

# Create your tasks here

from celery import shared\_task

from demoapp.models import Widget

@shared\_task

def add(x, y):

return x + y

@shared\_task

def mul(x, y):

return x \* y

@shared\_task

def xsum(numbers):

return sum(numbers)

@shared\_task

def count\_widgets():

return Widget.objects.count()

@shared\_task

def rename\_widget(widget\_id, name):

w = Widget.objects.get(id=widget\_id)

w.name = name

w.save()

# django-celery-results

pip install django-celery-results

INSTALLED\_APPS = (

...,

'django\_celery\_results',

)

python manage.py migrate django\_celery\_results

CELERY\_RESULT\_BACKEND = 'django-db'

celery -A proj worker -l INFO

Multithreading

# Thread

* A thread is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System).
* A thread is a sequence of instructions within a program that can be executed independently of other code

# Multithreading

Multithreading is defined as the ability of a processor to execute multiple threads concurrently.

import threading

def print\_cube(num):

"""

function to print cube of given num

"""

print("Cube: {}".format(num \* num \* num))

def print\_square(num):

"""

function to print square of given num

"""

print("Square: {}".format(num \* num))

if \_\_name\_\_ == "\_\_main\_\_":

# creating thread

t1 = threading.Thread(target=print\_square, args=(10,))

t2 = threading.Thread(target=print\_cube, args=(10,))

# starting thread 1

t1.start()

# starting thread 2

t2.start()

# wait until thread 1 is completely executed

t1.join()

# wait until thread 2 is completely executed

t2.join()

# both threads completely executed

print("Done!")

# Multiprocessing

Multiprocessor is a computer with more than one central processor.

Multi-core processor is a single computing component with two or more independent actual processing units (called “cores”).

import multiprocessing

def print\_cube(num):

"""

function to print cube of given num

"""

print("Cube: {}".format(num \* num \* num))

def print\_square(num):

"""

function to print square of given num

"""

print("Square: {}".format(num \* num))

if \_\_name\_\_ == "\_\_main\_\_":

# creating processes

p1 = multiprocessing.Process(target=print\_square, args=(10, ))

p2 = multiprocessing.Process(target=print\_cube, args=(10, ))

# starting process 1

p1.start()

# starting process 2

p2.start()

# wait until process 1 is finished

p1.join()

# wait until process 2 is finished

p2.join()

# both processes finished

print("Done!")

|  |  |  |
| --- | --- | --- |
| S.NO | MULTIPROCESSING | MULTITHREADING |
| 1. | In Multiprocessing, CPUs are added for increasing computing power. | While In Multithreading, many threads are created of a single process for increasing computing power. |
| 2. | In Multiprocessing, Many processes are executed simultaneously. | While in multithreading, many threads of a process are executed simultaneously. |
| 3. | Multiprocessing are classified into Symmetric and Asymmetric. | While Multithreading is not classified in any categories. |
| 4. | In Multiprocessing, Process creation is a time-consuming process. | While in Multithreading, process creation is according to economical. |
| 5. | In Multiprocessing, every process owned a separate address space. | While in Multithreading, a common address space is shared by all the threads. |