python框架Sanic添加任务定时or间隔执行

Mar 9, 2022

最近看到Sanic框架，看了下大概的用法，有sync异步、主进程add_task、子进程add_task，测试了下用法

1. 常规add_task：

```
# coding=utf8

"""
python3
pip install asyncio sanic
"""

import asyncio
from sanic import Sanic, text

app = Sanic('myapp')

async def add_task_before_server_start():
    await asyncio.sleep(5)
    print('Server successfully started worker!')

# 在app.run 之前添加任务，每个子进程都会有该任务
app.add_task(add_task_before_server_start())

# 只为主进程添加任务，注意：任务没完成子进程将无法启动。
@app.main_process_start
async def main_start(*_):
    print(">>>>>>main_process_start 为主进程添加任务，成功后，启动子进程<<<<<<")

@app.get("/")
async def foo_handler(request):
    return text("foo_handler!")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, workers=2)  # 两个worker都会执行add_task
```

```
# 输出结果：
"""
[2022-03-09 07:57:36 +0000] [109] [INFO] Goin' Fast @ http://0.0.0.0:8000
>>>>>>main_process_start 为主进程添加任务，成功后，启动子进程<<<<<<
[2023-04-17 07:57:36 +0000] [110] [INFO] Starting worker [110]
[2023-04-17 07:57:36 +0000] [111] [INFO] Starting worker [111]
Server successfully started worker!
Server successfully started worker!
"""
```

2. 主进程add_task：

```
import os
import asyncio
from sanic import Sanic, text

app = Sanic('myapp')

async def add_task_before_server_start():
    await asyncio.sleep(5)
    print('Server successfully started worker!')

# 在app.run 之前添加任务，每个子进程都会有该任务
app.add_task(add_task_before_server_start())


# 只为主进程添加任务，注意：任务没完成子进程将无法启动。
@app.main_process_start
async def main_start(*_):
    print(">>>>>>main_process_start 为主进程添加任务，成功后，启动子进程<<<<<<")

    p=1
    while True:
        print(f"主进程：PID：{os.getpid()}. cnt={p}")
        p +=1
        await asyncio.sleep(3)

@app.get("/")
async def foo_handler(request):
    return text("foo_handler!")


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, workers=2)  # 注意worker=2
```

```
# 输出结果：
"""
>>>>>>main_process_start 为主进程添加任务，成功后，启动子进程<<<<<<
主进程:PID:123.cnt=1
主进程: PID:123.cnt=2
主进程:PID:123.cnt=3
主进程:PID:123cnt=4
主进程:123PID:cnt=5
主进程:PID:123cnt=6
主进程:PID:123cnt=7
"""
```

此时，请求服务，子进程还未启动，服务不可用：

```
# curl -i http://127.0.0.1:8000
curl: (7) Failed to connect to 127.0.0.1 port 8000: Connection refused
```

3. 实现间隔执行，定时任务：

app.add_task 会被覆盖
多个worker，定时 / 间隔会被每个worker都执行一次

```
# coding=utf8

import time
import asyncio
from sanic import Sanic, text
from apscheduler.schedulers.blocking import BlockingScheduler

app = Sanic('myapp')

# 多个workers时，会被多次执行
async def interval_task():
        while True:
                await asyncio.sleep(3)
                print("interval_task  asyncio >>>>> {}".format(time.time()))

app.add_task(interval_task()) # 注意，这个会被 sched 覆盖

def task_func():
        print("apscheduler task interval go >>>>>>{}".format(time.time()))

def task_min_func():
        print("apscheduler task cron go >>>>>>{}".format(time.time()))
```

```python
async def aps_task():
        while True:
                sched = BlockingScheduler()
                sched.add_job(task_func, 'interval', seconds=2, id='job_sec_2')
                sched.add_job(task_min_func, 'cron', minute='*/1')  #  crontab 执
                await sched.start()

app.add_task(aps_task())

@app.get("/")
async def foo_handler(request):
    return text("apscheduler apscheduler apscheduler!")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, workers=2)  # worker=2


# 输出结果:
"""
[2023-04-17 10:54:55 +0000] [210] [INFO] Goin' Fast @ http://0.0.0.0:8000
apscheduler task interval go >>>>>>1681728897.6770253
apscheduler task interval go >>>>>>1681728897.6774502

apscheduler task interval go >>>>>>1681728899.6779418
apscheduler task interval go >>>>>>1681728899.6783068
apscheduler task cron go >>>>>>1681728900.0017438
apscheduler task cron go >>>>>>1681728900.001731
"""
```