Python Assíncrono

Nilo Menezes

pythonnilo@gmail.com

Assíncrono?

- O que muda?
- Suporte da linguagem
- Vantagens
- Onde usar
- Como usar
- Bibliotecas

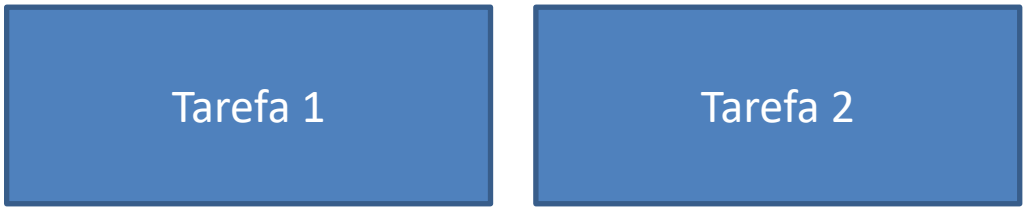
O que muda?

- Modelo de execução diferente
  - CPU compartilhada entre várias tarefas
  - Código pára e volta a executar em função de notificações de entrada e saída
- Multitarefa cooperativa
- Tudo roda em um só thread

Modelos de execução

- Sequencial

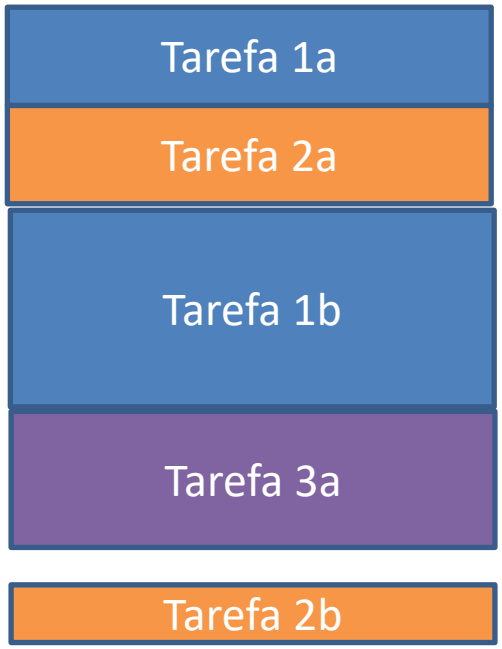
- Paralelo



- A) threads
- B) processos

Modelos de Execução

- Assíncrono



Vantagens

- Tudo roda em um só thread
- Mais fácil de escrever que código multithread
- Mais econômico que multiprocessos
- Aproveita a capacidade de I/O e melhor utiliza a CPU disponível
- Centenas de operações em um único programa

Desvantagens

- Tudo roda em um só thread
- Código mais difícil de entender
- Mais difícil de debugar

Onde usar?

- Problemas I/O bound
  - Rede (transferência de arquivos, DNS, etc)
  - Disco
  - Banco de dados
  - Problemas onde o tempo que se passa esperando pela entrada e/ou a saída é maior que o tempo de processamento

Como funciona?

- O loop de eventos gerencia quais tarefas devem executar
- Quando código assíncrono é chamado, o loop de eventos passa o controle para esta função.
- Quando está chama await ou retorna, o controle volta ao loop
- O código é reativado quando um evento acontece, como I/O, timers, etc

Como usar

- Python >= 3.5
- Suporte a async/await
- async declara métodos e funções assíncronas
- await chama uma função assíncrona e espera seu retorno

Como usar

- PEP492
- Suporte a gerenciadores de contexto assíncronos (async with)
  - async def __aenter__(self)
  - async def __aexit__(self, exc_type, exc, tb)
- async for
  - def __aiter__(self)
  - async def __anext__(self)

Bibliotecas

- Aiohttp
- Client

```python
import aiohttp
import asyncio
import async_timeout

async def fetch(session, url):
    with async_timeout.timeout(10):
        async with session.get(url) as response:
            return await response.text()

async def main():
    async with aiohttp.ClientSession() as session:
        html = await fetch(session, 'http://python.org')
        print(html)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

PYCON AMAZÔNIA

Bibliotecas

- Aiohttp
- Server

```python
from aiohttp import web

async def handle(request):
    name = request.match_info.get('name', "Anonymous")
    text = "Hello, " + name
    return web.Response(text=text)

app = web.Application()
app.router.add_get('/', handle)
app.router.add_get('/{name}', handle)

web.run_app(app)
```

Bibliotecas

- Aiopg – PostgreSQL Assíncrono

```python
import asyncio
import aiopg

dsn = 'dbname=aiopg user=aiopg password=passwd host=127.0.0.1'

async def go():
    pool = await aiopg.create_pool(dsn)
    async with pool.acquire() as conn:
        async with conn.cursor() as cur:
            await cur.execute("SELECT 1")
            ret = []
            async for row in cur:
                ret.append(row)
            assert ret == [(1,)]

loop = asyncio.get_event_loop()
loop.run_until_complete(go())
```

PYCON AMAZÔNIA

Bibliotecas

- aioredis – Redis Assíncrono

```
import asyncio
import aioredis

loop = asyncio.get_event_loop()

async def go():
    pool = await aioredis.create_pool(
        ('localhost', 6379),
        minsize=5, maxsize=10,
        loop=loop)
    await pool.execute('set', 'my-key', 'value')
    print(await pool.execute('get', 'my-key'))
    # graceful shutdown
    pool.close()
    await pool.wait_closed()

loop.run_until_complete(go())
```

Bibliotecas

- Sanic
  - Inspirado no Flask, mas assíncrono

```python
from sanic import Sanic
from sanic.response import json

app = Sanic()

@app.route("/")
async def test(request):
    return json({"hello": "world"})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

Mais exemplos

- https://github.com/timofurrer/awesome-asyncio
- http://junglecoders.blogspot.be/
- https://github.com/MagicStack/uvloop

Obrigado

- Nilo Menezes
- http://python.nilo.pro.br
- pythonnilo@gmail.com