PHP Assíncrono
cuidados e limitações que não te falam

Leonardo Tumadjian

Live PHPBA

Sobre mim

Tech Lead

PHP dev. desde 2009

Founder @DifferDev

Mentor da **Next-Gen PHP**

Professor/Mentor

Palestrante

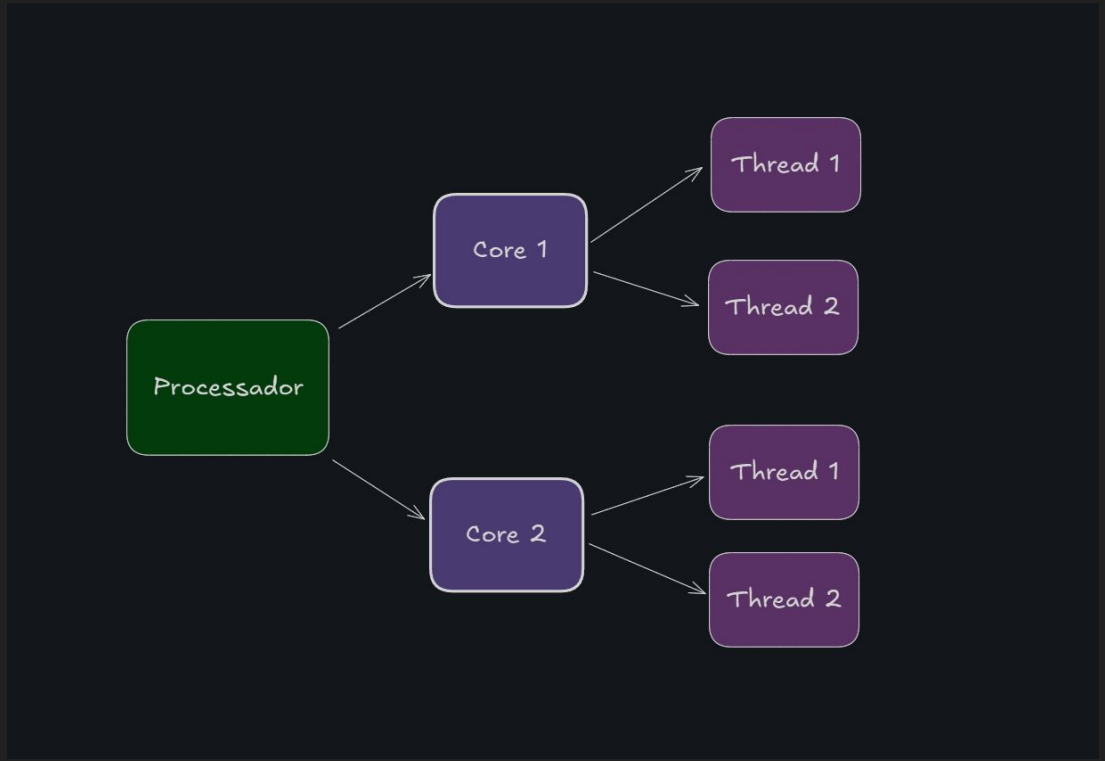
Arquiteto de software

Linkedin: Leonardo Tumadjian

https://differdev.com

https://youtube.com/c/differdev

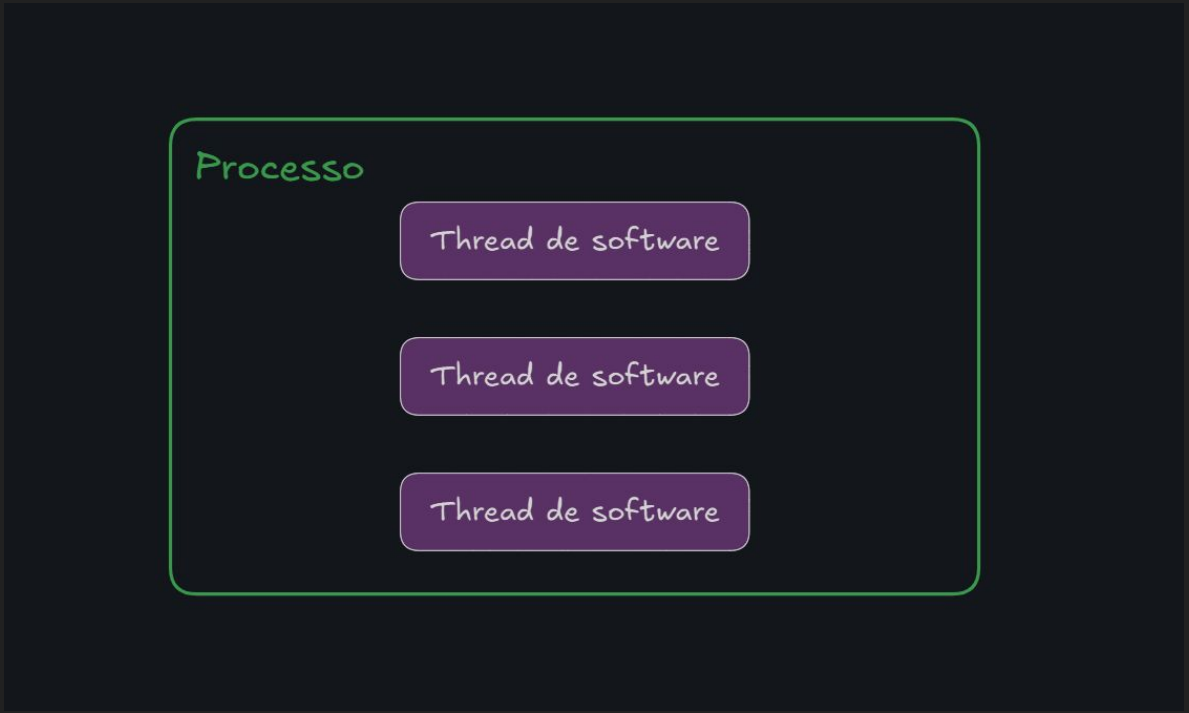
Revisando recursos do sistema



Parte física de computador como nosso hardware se subdivide.

Aqui onde temos a limitação de servidor para servidor.

Thread de software



O PHP tinha uma extensão para o uso de Threads em tempo de execução, chamada pthreads, hoje não mais recomendada, sendo substituída pela parallel.

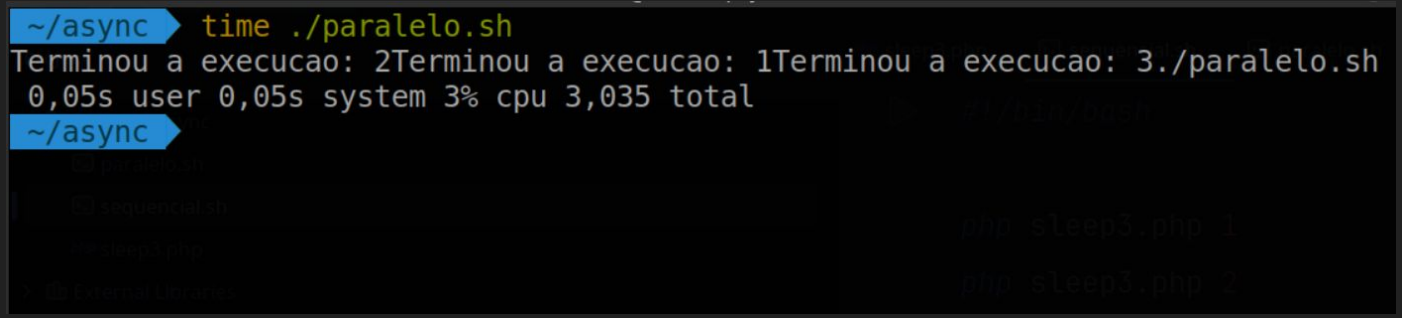
Threads são custosas e muito complexas para o desenvolvimento.

Paralelismo puro em PHP

```php
<?php

sleep( seconds: 3);

echo 'Terminou a execucao: ' . $argv[1] ?? 0;

```

```bash
#!/bin/bash

php sleep3.php 1 &
PID1=$!

php sleep3.php 2 &
PID2=$!

php sleep3.php 3 &
PID3=$!

wait $PID1
wait $PID2
wait $PID3
```

Resultado


~/async time ./paralelo.sh
Terminou a execucao: 2Terminou a execucao: 1Terminou a execucao: 3./paralelo.sh
0,05s user 0,05s system 3% cpu 3,035 total
~/async

Outra forma de paralelismo: Fork de processo

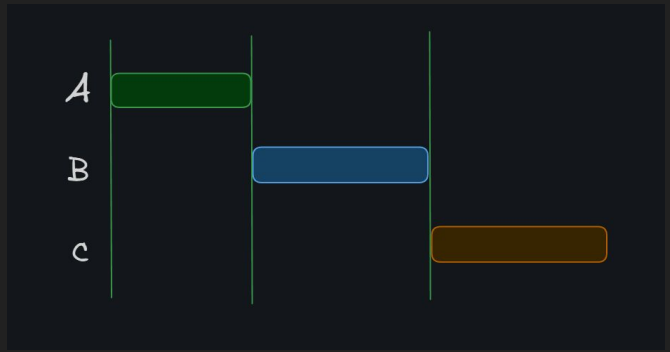
```php
$pid = pcntl_fork();

if ($pid == -1) {
    // Erro ao criar processo filho
    die("Erro ao criar processo filho.\n");

} elseif ($pid === 0) {
    // Processo filho
    echo "Processo filho iniciado. PID: " . getmypid() . "\n";
    sleep( seconds: 2);
    echo "Processo filho finalizado.\n";
    exit(0);

} else {
```

Conseguimos criar um fork do processo pela extensão pcntl com ela é possível no mesmo script criar processos separados que compartilham o mesmo conjunto de memória, porém há uma limitação em grande escala.

Limitação:
pode sofrer child process limit, em php-fpm.

Sincronismo vs Assincronismo

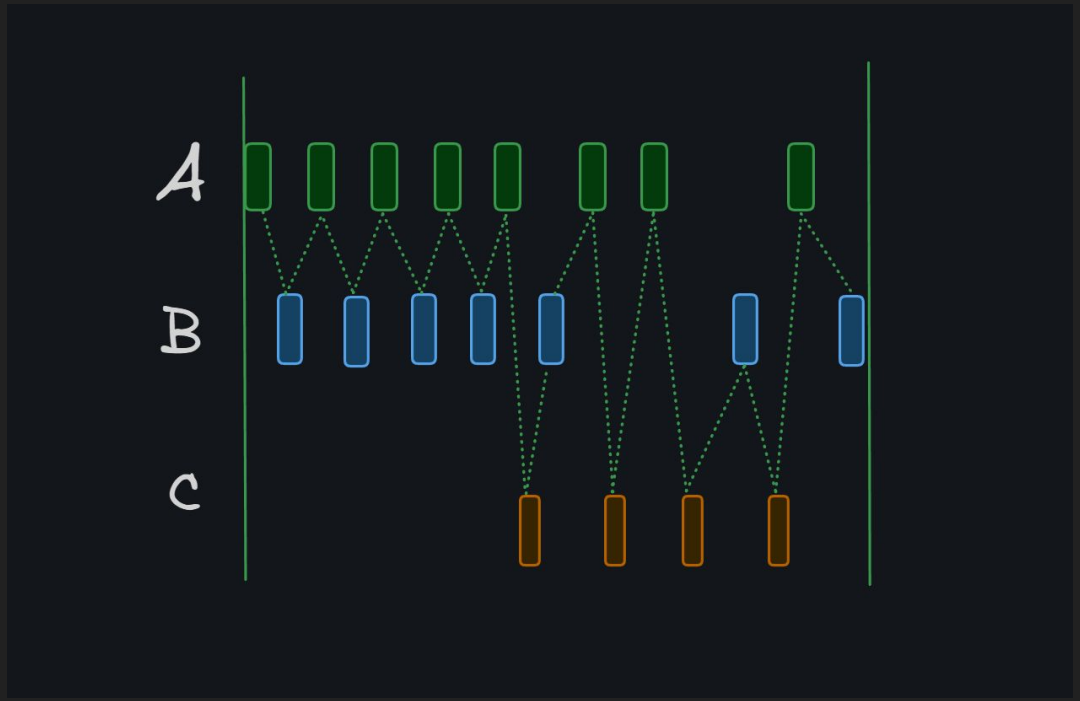
Exemplo síncrono



Exemplo assíncrono

Corrotinas

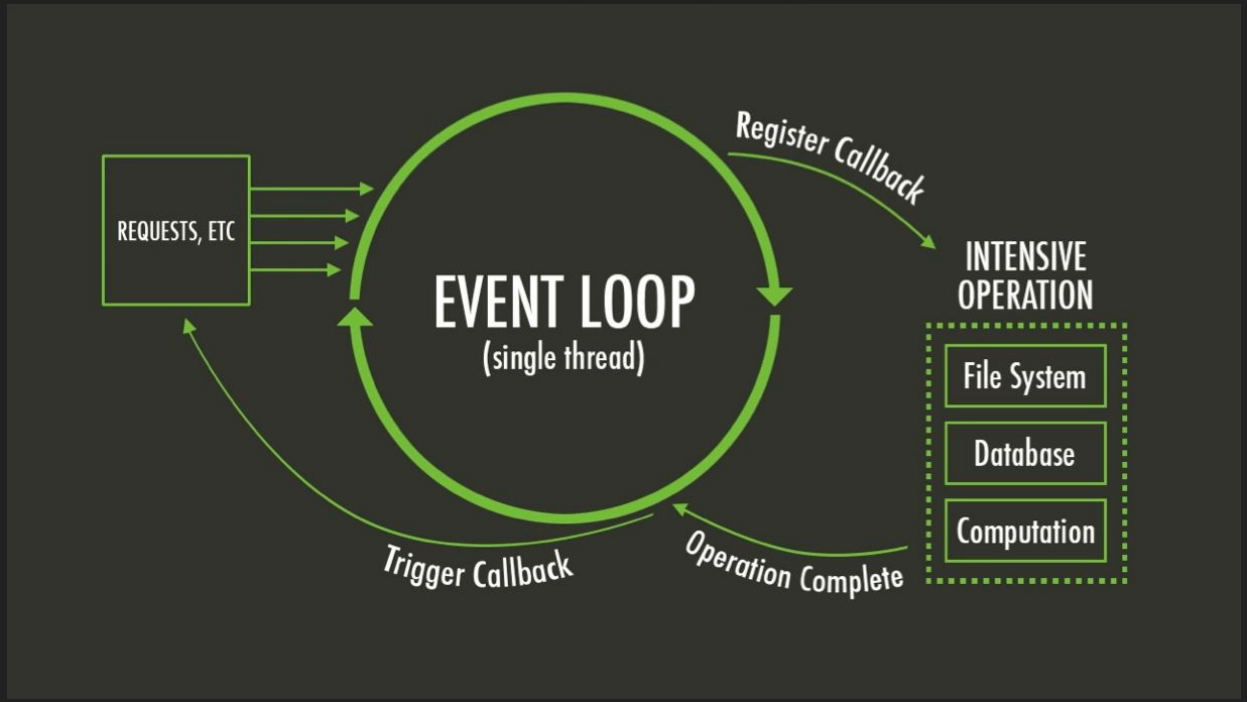
Exemplo assíncrono



Corrotinas fracionam as execuções de métodos para que “caibam” dentro de um espaço/tempo parecendo ser uma comunicação simultânea, mas na verdade a comunicação é sequencial.

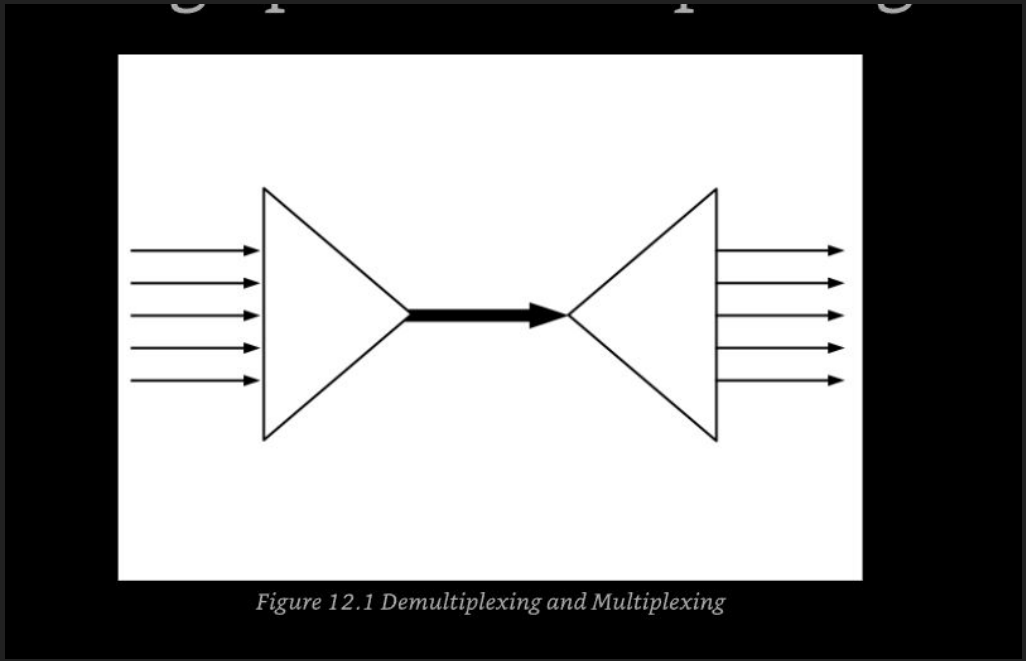
Como coordenar isso?

Event-loop



O event loop controla a execução das corrotinas, é possível implementar ações como um **Timer** e até mesmo execução com delay, como na setTimeout do Javascript.

Multiplexing e Demultiplexing

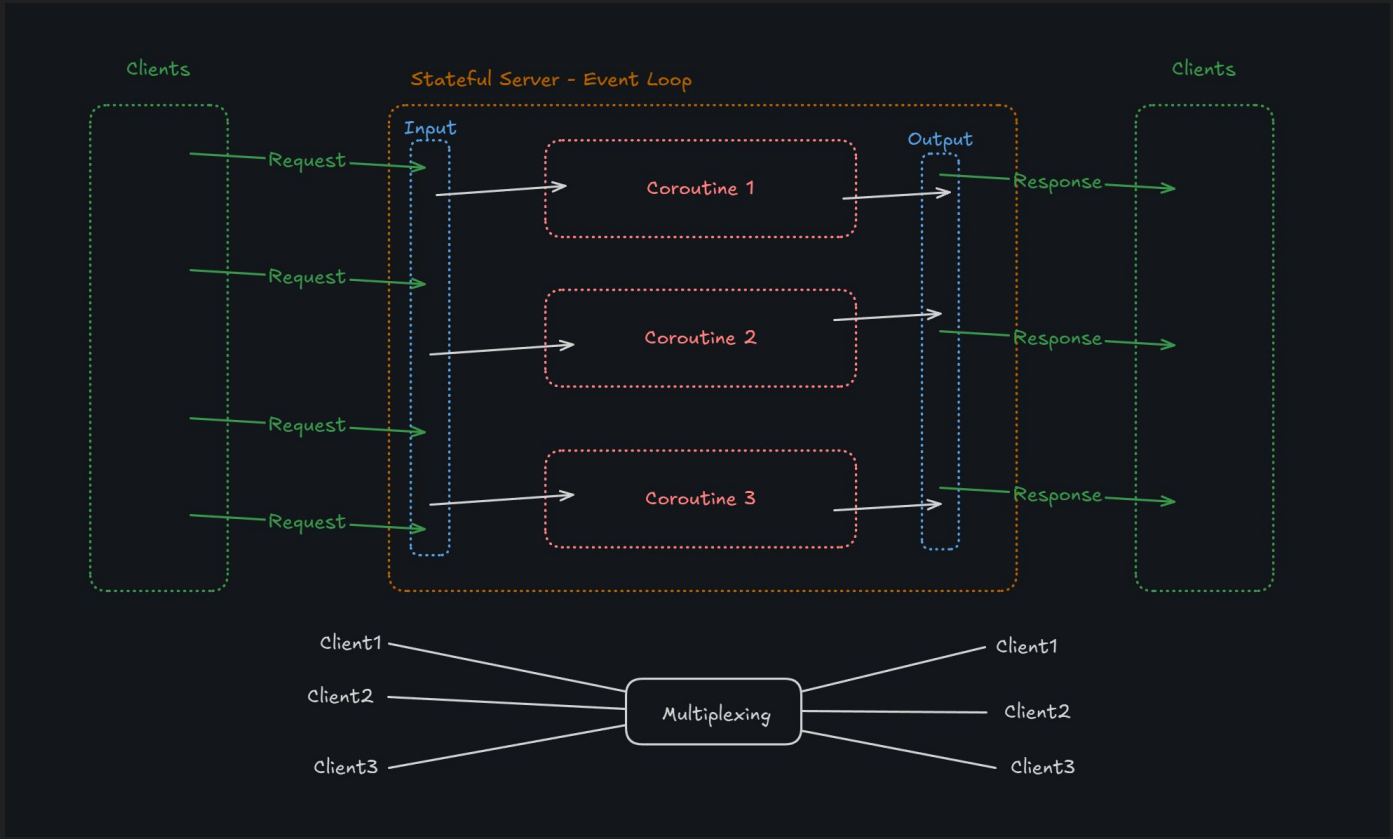


Consiste em achatar a comunicação em um único canal a fim de conseguir ter um único ponto de comunicação para as corrotinas, imagina que temos pedaços e eles precisam posteriormente de reorganização e despacho aos seus clients.

Source: E-Book Mastering Swoole PHP

Veja o exemplo:

Podem ocorrer em uma única thread



Primeiras limitações

Fazer um event loop na mão pode parecer divertido, mas trará problemas de comunicação, uma delas acontece que toda função nativa do PHP é **bloqueante** abaixo trago alguns pontos que geram bloqueio do event-loop.

1. Funções de leitura de arquivo(file_get_contents, file_put_contents e etc)
2. Métodos de comunicação em rede como curl, fopen entre outros precisam ser trabalhados de outra forma.
3. Input/Output do PHP(default) é bloqueante, por isso é preciso ter sua implementação de socket
4. Comunicação client socket/tcp com Redis / Mysql / Postgres / MongoDB
5. Comunicação com filas como SQS, RabbitMQ / AMQP, Kafka e etc extensões são bloqueantes
6. Tudo que for I/O tradicional, gerará problemas a solução será sempre procurar por implementações customizadas assíncronas

Web server com event-loop

O projeto ReactPHP é um projeto que apareceu logo que o PHP começou a trazer funcionalidades para sockets/streams, posteriormente com generators, esta lib consegue criar um servidor stateful em PHP sem precisar de uma extensão(mas pode ser usada com).

Porém é importante tudo que for acesso externo, será preciso usar clients desenvolvido em cima desta lib, esse é um grande desafio, deve-se não usar funções nativas do PHP que fazem comunicação com serviços externos, seu uso pode gerar o problemas de bloqueio do event loop.

Web server com event-loop

```php
require __DIR__ . '/vendor/autoload.php';

use Psr\Http\Message\ServerRequestInterface;
use React\Http\HttpServer;
use React\Http\Message\Response;

$http = new HttpServer(function (ServerRequestInterface $request) {
    return Response::plaintext( text: 'Hello World!');
});

$socket = new React\Socket\SocketServer( uri: '0.0.0.0:8080');
$http->listen($socket);

echo "Server running at http://0.0.0.0:8080" . PHP_EOL;
```

Bloqueio do event-loop

Vamos ver uma mão na massa para entender como isso acontece!

Referência de material: 3-WebServer

Outros projetos

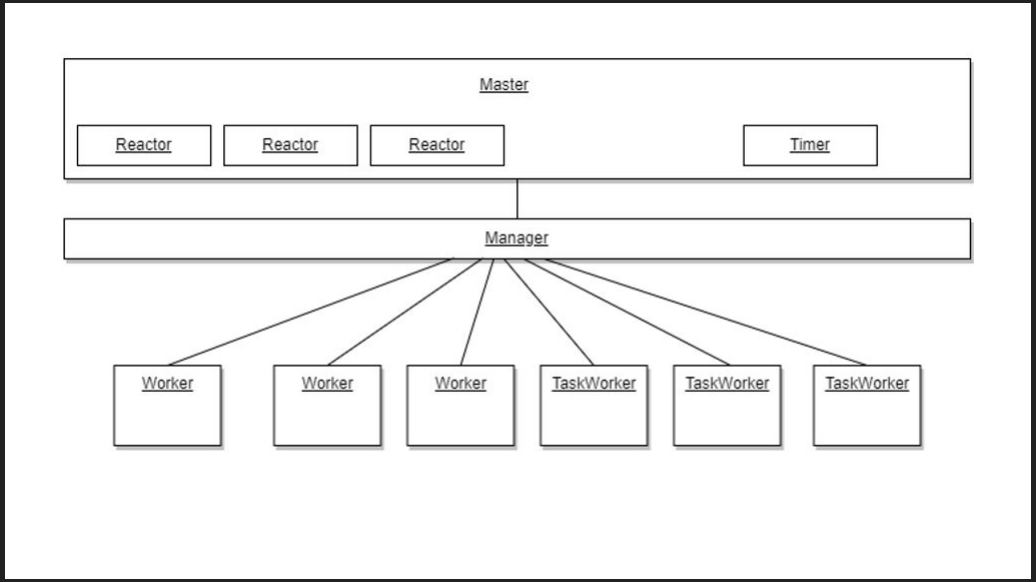
Amphp - https://amphp.org/

Revolt - https://revolt.run/

Swoole - https://swoole.com/

Swow - https://docs.toast.run/swow/en/

Sobre o Swoole



O Swoole é uma extensão em PHP que permite de forma mais abrangente habilitar o assincronismo na linguagem, um ponto que mais destaca das demais ferramentas é sua possibilidade de alterar funções nativas do PHP para que não travem o event-loop, também é possível usar seu webserver built-in, que chega a benchmarks impressionantes.

Vejamos a seguir os hooks possíveis do Swoole.

Hooks no Swoole

1. Hooks relacionados à rede:

- **SWOOLE_HOOK_TCP**: Habilita o suporte a corrotinas para sockets TCP e streams, incluindo os utilizados pelo Redis, PDO, MySQLi e Streams do PHP.
- **SWOOLE_HOOK_UNIX**: Habilita o suporte a corrotinas para sockets de domínio Unix.
- **SWOOLE_HOOK_UDP**: Habilita o suporte a corrotinas para sockets UDP.
- **SWOOLE_HOOK_UDG**: Habilita o suporte a corrotinas para funções relacionadas a UDP.
- **SWOOLE_HOOK_SSL**: Habilita o suporte a corrotinas para conexões SSL/TLS.
- **SWOOLE_HOOK_TLS**: Habilita o suporte a corrotinas para conexões TLS.
- **SWOOLE_HOOK_CURL**: Habilita o suporte a corrotinas para a extensão cURL, permitindo requisições HTTP/1 não bloqueantes.
- **SWOOLE_HOOK_SOCKETS**: Habilita o suporte a corrotinas para a extensão de sockets.

2. Hooks de gerenciamento de processos:

- **SWOOLE_HOOK_PROC**: Habilita o suporte a corrotinas para funções proc_* como proc_open, proc_close, proc_get_status e proc_terminate.

3. Hooks de arquivos e I/O:

- **SWOOLE_HOOK_FILE**: Habilita o suporte a corrotinas para operações com arquivos como fopen, fread, fwrite, file_get_contents, etc.
- **SWOOLE_HOOK_STREAM_FUNCTION**: Habilita o suporte a corrotinas para funções relacionadas a streams.
- **SWOOLE_HOOK_STDIO**: Habilita o suporte a corrotinas para operações de entrada/saída padrão.

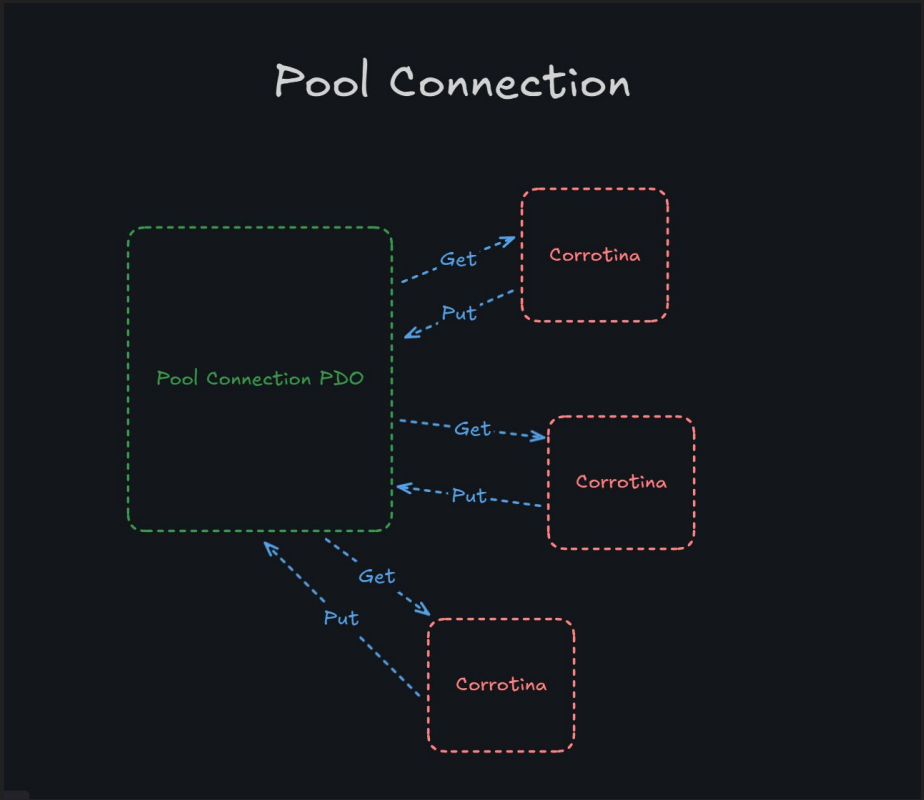
4. Hooks de tempo e atraso:

- **SWOOLE_HOOK_SLEEP**: Habilita o suporte a corrotinas para funções de bloqueio de tempo como sleep, usleep, time_nanosleep e time_sleep_until.

Swoole mão na massa

Vamos entender como o Swoole resolve isso!

Pool Connection



Trabalhando com um servidor como o Swoole, assíncrono, em um event-loop usando corrotinas, o mais indicado é que toda comunicação passe por uma pool de comunicação, pois alí é possível coordenar as operações sem que haja a violação de consumo de recursos, vamos fazer agora um desafio de ajustar 300 mil inserts no bando em PHP movendo para o formato assíncrono com Pool Connection no PDO.

Últimas considerações

1. O PHP tem várias formas de trabalhar paralelo e assíncrono!
2. Consumo de memória aumenta e precisa ser cuidado.
3. Para comunicações externas e massivas, use Pool Connection.
4. Use as corrotinas quando realmente tiver necessidade.
5. Monitore toda a execução, cuidado com o event-loop blocking.
6. Estude a configuração do servidor Swoole para entender melhor como executá-lo.

Obrigado pela presença!

Bons códigos assíncronos para vocês!

Dúvidas?

Conheça o programa de mentoria DifferDev

https://differdev.com/next-gen-4/