

PHP funcional além do array_map

Jean Carlo Machado

Sobre

CompuFácil

- php-src
 - git/git
- torvald/linux
 - vim/vim
 - Doctrine
- Zend Framework
 - phpunit



Haskell

1936: Imperativo

jumps: GOTO

1950: Estruturado

subrotinas: while, for, if

~~GOTO~~

Programação Funcional

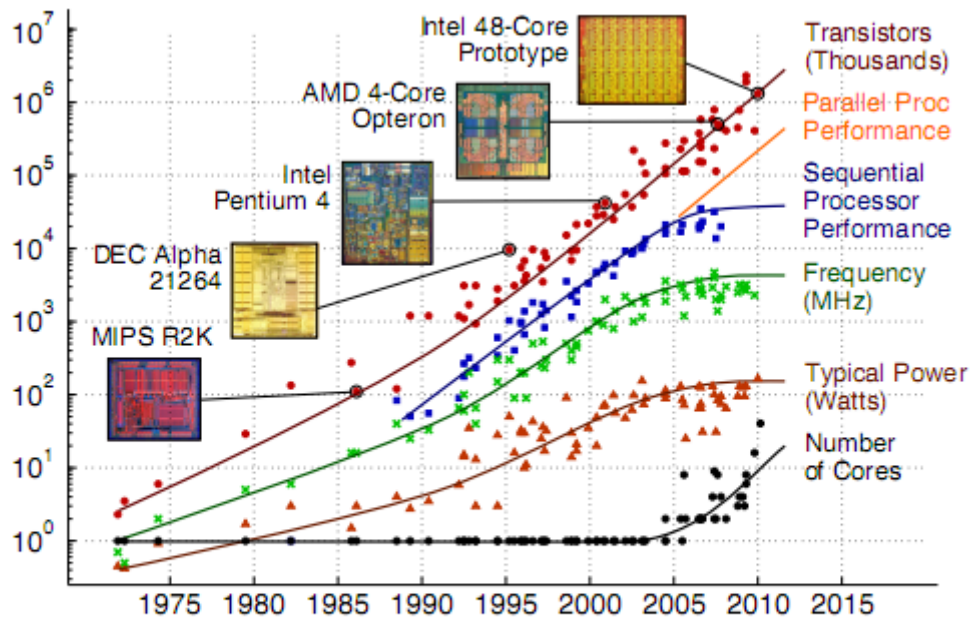
~~mudança de estado~~

Vantagens

Concorrência nativa

Facilita a modularização

Desacelera o apodrecimento



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

Prepared by C. Batten - School of Electrical and Computer Engineering - Cornell University - 2005 - retrieved Dec 12 2012 -

<http://www.ece.cornell.edu/courses/ece5950/handouts/ece5950-overview.pdf>

Prática

Estado === Impureza

```
$inpureIncrement = function (&$count) {  
    $count++;  
};  
for ($i = 0; $i < 3; $i++) {  
    echo "Next:". $inpureIncrement($i);  
}  
//Next: 0  
//Next: 2  
//Next: 4
```

Pureza (relativa)

```
function pureIncrement($count) {  
    $myCounter = $count;  
    $inpureIncrement = function(&$myCounter){  
        $myCounter++;  
    };  
    $inpureIncrement($myCounter);  
    return $myCounter;  
}  
//Next: 1  
//Next: 2  
//Next: 3
```

Impuros por natureza

IO

random

Funções de alta ordem

Retornáveis

```
$sumPartial = function(int $a) {  
    return function(int $b) use ($a) {  
        return $a+b;  
    }  
};  
$sumOne = $sumPartial(1);  
$sumOne(7);  
//8  
$sumOne(5);  
//6
```

Repassáveis

```
$incrementTwo = function ($n, $sumOne) {  
    return $sumOne($sumOne($n));  
};
```

Fold

Defina o básico

```
function sum($a, $b) {  
    return $a + $b;  
}  
//function product($a, $b);  
//function aORb($a, $b);  
//function append($a, $b);  
//function greater($a, $b);
```

Componha

```
$sumList = fold('sum', 0, [1,2,3]);  
//6  
$multiplyList = fold('product', 1, [2,2,2]);  
//8
```

```
function fold(  
    callable $f,  
    $init,  
    $list) {  
  
    if (empty($list)) {  
        return $init;  
    }  
    return $f(  
        fold($f, $init, allbutlast($list)),  
        last($list)  
    );  
};
```

Map

```
$double = function($a) {  
    return $a*2;  
}
```

```
$map($double, [1, 2, 4]));  
//[2,4,8]
```

```
$map = function($f, $list) {  
    $applyAndAppend=function($f,$list,$b){  
        $list[] = $f($b);  
        return $list;  
    };  
    return fold($applyAndAppend,null)($list);  
}
```

Árvores

```
$tree = [  
  1 => [  
    2,  
    3 => [  
      4,  
    ],  
  ],  
];  
foldTree($sum, $sum, 0, $tree)  
//10
```

Map Tree

```
$tree = [  
    3 => [1,4],  
    1 => [1,5 => [1,2,3]]  
];  
$result = mapTree($double, $tree);  
//[  
//     6 => [2,8],  
//     2 => [2,10 => [2,4,6]]  
//];
```

PHP functions

- array_map
- array_filter
- array_reduce
- array_sum
- array_unique

g (f input)

- começa f só quando g tenta ler algo
- suspende f
- g roda até precisar ler mais dados

Nth primes

```
getNthPrimes(10);  
//[1,2,3,5,7,11,13,17,19,23]
```

Nth primes

```
function primes() {  
    foreach (\f\infiniteSequence(1) as $i) {  
        if (isPrime($i)) {  
            yield $i;  
        }  
    }  
}
```

```
print_r(\f\takefrom(primes(), 10));  
//[1,2,3,5,7,11,13,17,19,23]
```

```
function isPrime($x) {
    $seq = \f\takeFrom(
        \f\infiniteSequence(),
        $x
    );
    $seq = \f\removeFirst($seq);
    $seq = \f\removeLast($seq);
    $multiple = function($a) use ($x) {
        return ($x % $a == 0);
    }
    return \f\noneTrue(
        \f\map($multiple, $seq)
    );
}
```

}

Aplicação parcial

```
$append3 = function($a, $b, $c) {  
    return [$a, $b, $c];  
};  
$append1And2 = \f\partial($append3)(1)(2);  
$append1And2(5)  
//[1,2,5]  
$append1And2(9)  
//[1,2,9]
```

```

function partial(
    callable $callable, ...$args){
    $arity = (new \ReflectionFunction(
        $callable
    ))
    ->getNumberOfRequiredParameters();
    return $args[$arity - 1] ?? false
    ? $callable(...$args)
    : function (...$passedArgs)use($callable,
        return partial($callable,
            ...array_merge($args, $passedArgs)
        );
    };
}

```

Real: get-profile

Requisito v1

```
curl http://localhost:8000/profile/777  
#{ "id": 777, "name": "Saruman" }
```



```
function getProfile(  
    callable $query,  
    $userId) {  
    return $query(  
        "Select * from User where id = %id ",  
        $userId  
    );  
}  
  
$realProfile = \f\partial  
    ('getProfile')  
    ([new Database, 'query']);
```

Requisito v2: cache

```
curl http://localhost:8000/profile/777  
# go to database  
#{ "id": 777, "name": "Saruman" }  
curl http://localhost:8000/profile/777  
# don't go to database  
#{ "id": 777, "name": "Saruman" }
```

```
$memoize = function($func) {  
    static $results = [];  
    return function ($a) use (  
        $func, &$results) {  
        $key = serialize($a);  
        if (empty($results[$key])){  
            $results[$key]=call_user_func(  
                $func, $a );  
        }  
        return $results[$key];  
    };  
};  
$memoizedProfile = $memoize($realProfile);
```

Requisito v3: log request

```
curl http://localhost:8000/profile/777  
#{"id":777,"name":"Saruman"}  
# also writes to file  
cat /tmp/log  
#getProfile called with params s:3:"777";
```

```
function fileLogger($str) {  
    file_put_contents(  
        '/tmp/log',  
        "FileLog: ".$str.PHP_EOL,  
        FILE_APPEND  
    );  
}
```

```
function logService(  
$logger, $serviceName,  
callable $service) {  
    return function($args) use (  
        $logger, $serviceName,  
        $service) {  
        $logger(  
            "Service called ". $serviceName.  
            " with params ".serialize($args));  
        return call_user_func($service, $args);  
    };  
};
```

```
$loggedMemoizedGetProfile =  
  \f\partial('logService')  
    ($logger)  
    ('getProfile')  
    ($memoizedProfile);
```

Modularização

Mais que módulos

Decompor os problemas em partes menores

Re-compor com avaliação tardia e funções de alta ordem

~~Apodrecimento~~

- Difícil adicionar efeitos sem quebrar as interfaces
 - Quão maior a interface mais feio o código
- Interfaces facilmente quebráveis com composição
 - Quebrar encoraja reuso
- Complexidade horizontal ao invés de vertical

Conclusão

Imperativo quando necessário
Funcional quando possível

Para novos tipos

- 1 - defina as operações fundamentais**
- 2 - junte funções**

Seu trabalho não é resolver problemas

**Definir problemas de uma forma que
eles se resolvam**

Ferramentas

- hlstrojny/functional-php
- functional-php/pattern-matching
 - jeanCarloMachado/f
 - pimple/pimple

Referências

→ Why functional programming matters

→ <http://archive.li/uTYWZ#selection-407.912-407.919>

→ <https://medium.com/@jugoncalves/functional-programming-should-be-your-1-priority-for-2015-47dd4641d6b9>

→ <https://blog.inf.ed.ac.uk/sapm/2014/03/06/enemy-of-the-state-or-why-oop-is-not-suited-to-large-scale-software/>

→ <https://www.youtube.com/watch?v=7Zlp9rKHGD4>

→ <https://www.youtube.com/watch?v=q0HRCEKAcas>

Dúvidas?

Link: goo.gl/cwJr8y

Github: <https://github.com/jeanCarloMachado>

Twitter: <https://twitter.com/JeanCarloMachad>

E-mail: contato@jeancarlomachado.com.br