Programmation fonctionnelle en JavaScript :  ou  ?

Speaker

Igor Laborie

Expert Web & Java @ilaborie igor@monkeypatch.io

Knowledge

#2

Langages pratiqués

1. Java
2. JavaScript (TypeScript, CoffeeScript)
3. Kotlin, Scala

Notions dans

Python, SML, Racket, Rust, Swift, Go, ...

Roadmap

I. Langages fonctionnels
II. Programmation fonctionnelle en JavaScript - Part I
III. Entracte
IV. Programmation fonctionnelle en JavaScript - Part II
V. Remaque sur la performance
VI. Conclusion

Langages fonctionnels

Fooling around with alternating current (AC) is just a waste of time. Nobody will use it, ever. -- Edison, 1889

*

"

There is no reason anyone would want a computer in their home. -- Ken Olson, 1977

- "

I predict the Internet will soon go spectacularly supernova and in

- C est un langage fonctionel
- JavaScript est un langage fonctionel
- Java est un langage OO
- JQuery est une monade

Paradigmes

- programmation impérative
- programmation orientée objet
- programmation fonctionnelle
- ...

> On peut adopter donc un style de programmation fonctionnelle avec la plupart des langages. Les carcactéristiques des langages peuvent rendre cela plus ou moins facile (voir obligatoire)

Only one

C'est quoi un langage fonctionnel ?

> Il n'y a qu'un langage fonctionnel : le λ-calcul

developpeur fonctionnel => Devel

(Haskell, normaux, humour, paresseux, lambda)

Static vs Dynamic

Programmation fonctionnelle <-> Typage statique

Dynamique

Lisp (1958)

Racket (1994), Clojure (2007), ...

Statique

ML (1973)

Haskell (1990), Scala (2004)

Programmation fonctionnelle en JavaScript - Part I

Function


```
function mult(a, b) {
    return a * b;
}

typeof mult; // "function"
```

Anonymous

```
var mult = function (a, b) {
    return a * b;
}
```

ES2015

Pure Function


```
var sum = 0;

[1, 2, 3, 4, 5]
    .forEach(elt ⇒ sum += elt);
console.log({sum});
```

⚠️ danger, c'est un nid à bugs.

=> Éviter les fonctions qui n'ont pas de paramètres, ou retournent `void`

```
const sum = [1, 2, 3, 4, 5]
    .reduce((acc, elt) ⇒ acc += elt);
```

referentially transparent

avec ImmutableJS

Comment fait-on ?

```
class List<T> {
    private array: T[];

    constructor(elements: T[] = []) {
        this.array = [ ... elements];
    }

    add(element: T) : List<T> {
```

Composition

High Order function

map, reduce, filter, sort

```
// f: X ⇒ Y
// g: Y ⇒ Z
// compose: (Y ⇒ Z, X ⇒ Y) ⇒ X ⇒ Z
const compose = (g, f) ⇒ x ⇒ g(f(x))
```

```
const factorial3 = (n, acc = 1) ⇒
    (n⩽1) ? acc : factorial3(n - 1, acc * );
```

Support de la tail-rec)

⚠️ Éviter les recusions entre plusieurs fonctions:

```
const a = n ⇒ {
    if (n > 42) {
        return b(n - 42);
    }
    return 2;
}
```

Lisibilité


```
speakers
    .filter(speaker ⇒ speaker.xp > 10 &&
        speaker.some(lang ⇒ lang === 'JavaScript'))


speakers
    .filter(speaker ⇒ speaker.xp > 10) // is experime
    .filter(speaker ⇒ speaker.some(lang ⇒ lang === '


const isExperimented = speaker ⇒ speaker.xp > 10;
const isLoveJS = speaker ⇒ speaker.loves.some(lang =

speakers
    .filter(isExperimented)
    .filter(isLoveJS)
```

Part I - bilan

- `function` first-class citizen
- immutable faisable

Avoid

- Loops: `while`, `do ... while`, `for`, `for ... of`, `for ... in`
- Variable declarations with var or let
- Void functions
- Object mutation (for example: o.x = 5;)
- Array mutator methods: `copyWithin`, `fill`, `pop`, `push`, `reverse`, `shift`,`sort`, `splice`, `unshift`
- Map mutator methods: `clear`, `delete`, `set`

Entracte

By Adrian Pingstone (Photographed by Adrian Pingstone) [Public domain], ➳ via Wikimedia Commons

Photo by Lance Anderson on Unsplash

Ailurus fulgens

Habitat: the Himalayan forests of Nepal and China.

Programmation fonctionnelle en JavaScript - Part II

Jargon

A functor is a collection of X that can apply a function f : X → Y over itself to create a collection of Y.

(Array with map)

Reduction

Function: (X, X) -> X

Combinators

Composition

Currification

into a chain of functions of one argument that will yield the same result when called in sequence with the same arguments.

```
f(x, y, z) = g(x)(y)(z)
```

```
const mult = (a: number, b: number) => a * b;

// idea: identity = mult(1, _)
const identity = (a: number) => mult(1, a);

// better
const multCurry = (a: number) => {
    return (b: number) => a * b;
}
```

Memoïsation

High Order Function

➳ https://medium.freecodecamp.org/understanding-memoize-in-javascript-51d07d19430e ➳ https://www.sitepoint.com/implementing-memoization-in-javascript/

```
const fibonacci = function(n) {
    switch (n) {
        case 1 : return 1;
        case 2 : return 1;
        default:
            return fibonacci(n-2) + fibonacci(n-1);
    }
}
```

Abstract Data Type

```
datatype rational = Whole of int
                  | Frac  of int*int
```

TypeScript

type union

Pattern Matching

```
let                    vector                    vector, {
    { x, y, z }: Math.sqrt(x ** 2 + y ** 2 + z ** 2),
    { x, y }:    Math.sqrt(x ** 2 + y ** 2),
    [ ... ]:        vector.length,
    else: {
        throw new Error("Unknown vector type");
    }
}
```

Pour implémenter du pattern-matching il faut de la déconstruction. On l'a déjà sur les {} et les [].

```
const myPoint = { x: 14, y: 3 };
const {x, y} = myPoint; // x === 14, y === 3

const tab = [1, 2, 3, 4];
```

Monads


```
Some(1)
    .map { it + 1 }
    .flatMap { Some("4" + it) }
    .map { it }
```

Part II - bilan

- langage souple permet pas mal de manipulation
- manque `flatMap`
- mais ➳ Ramda

Remaque sur la performance

Performance en quoi ?

- temps d'éxécution (mimimum, maximun, moyen, première exécution) ?
- consomation de mémoire ?
- consomation d'énergie ?

Rules

- on priviligie la lisibilité du code à une (hypothétique) optimisation de performance
- si besoin de meilleures performances (à définir), on définit le seuil désiré
- on effectue des mesures
- on suit l'évolution de ces performances dans toute la durée de vie du projet
- on isole la zone à optimiser, idéalement la plus petite possible
- on commente, pour expliquer pourquoi on n'a perdu de la lisibilité
- on effectue des mesures

- quel est l'objectif (mesurable) ?
- faire des mesures
- identifier le bottleneck
- amélioration

Conclusion

- les bases sont présentes.
- écosystème dans ce domaine, plutôt à la mode

Mais encore insuffisant à nom gout:

- `flatMap` 😭
- un lazy Array (comme les `Stream` Java)

Bibliothèque

> Program designers have a tendency to think of the users as idiots who need to be controlled. They should rather think of their program as a servant, whose master, the user, should be able to control it. If designers and programmers think about the apparent mental qualities that their programs will have, they'll create programs that are easier and pleasanter — more humane — to deal with.

-- John McCarthy "The Little Thoughts of Thinking Machines", Psychology Today, December 1983, pp. 46–49. Reprinted in Formalizing Common Sense: Papers By John McCarthy, 1990, ISBN 0893915351

Values

> La valeur principale d'un programme est dans ce qu'il réalise, pas dans son style.

Mais en temps qu'artisan développeur, j'accorde de l'importance au style.

Value of JS

expressivité souplesse ecosysteme évolu dans le bon sens

Why functional

- Plus facile à tester
- moins de bugs
- plus simple
- plus évolutif *

Liens

slides

-  Fantasy Land Specification (aka "Algebraic JavaScript Specification")

Questions ?