(((ClojureScript)))

Simple & Powerful

Jean Boudet & Stéphane Labruyère
June 30th, 2018
Fullstack devs @ BeOpinion


BRACE YOURSELVES
PARENTHESES ARE COMING

What are Clojure and ClojureScript ?

- LISP (1958)
- Functional, Immutable
- REPL-orientated programming (leiningen)
- Clojure => JVM / ClojureScript => JavaScript
- Rich Hickey power!


SCHIBSTED MEDIA GROUP

Tools needed

ClojureScript

JavaScript

BABEL

Gulp

GRUNT
The JavaScript
Task Runner

BOWER

IMMUTABLE

A matter of parentheses :)

>ClojureScript

```clojure
; comment

(+ 1 1)

(+ 1 1 1 1 1)

(def hello "Hello World!")

(defn add
  [x y]
  ; last statement is returned
  (+ x y))

(add 1 2)

(map inc [1 2 3])

(-> 1 baz bar foo) ;;=> (foo (bar (baz 1)))
```

>JavaScript

```javascript
// comment

1 + 1;

1 + 1 + 1 + 1 + 1;

var hello = "Hello World!"

function add(x, y) {
    return x + y;
}


add(1,2);

[1 2 3].map(x => x + 1);

foo(bar(baz(1)));
```

A matter of parentheses :)

>ClojureScript

```clojure
(def ned
  {:name "Ned"
   :home "Stark"
   :meta {:height 188 :age 34})
(:name ned) ;;=> "Ned"
(get-in ned [:meta :age]) ;;=> 34

(def ned-updated
  (assoc-in ned [:meta :age] 65))
```

>JavaScript

```javascript
var ned = {
  name: "Ned",
  home: "Stark",
  meta: {height: 188, age: 34}
};
ned["name"] //=> "Ned"
ned["meta"]["age"] //=> 34

var nedUpdated = Object.assign(
  {},
  ned,
  {meta: {...ned.meta, age: 65}});
```

A matter of parentheses :)

>ClojureScript

```
(-> event .-target .-value)
(.getElementById js/document "id")
```

>JavaScript

```
event.target.value;
document.getElementById("id");
```

```
(function arg1 arg2)
```


I ALSO
LIKE TO LIVE DANGEROUSLY
quickmeme.com

Demo time

Building a small todo list

- Reagent: ClojureScript React wrapper
- Figwheel: hot reloading
- Emacs: just… because!

Code available here.

All that for just that?

Data manipulation: power of the core lib!

```clojure
(def is-array-sorted? (> 1 2 3 4 5 6)) ; true

(defn array-sorted?
  [a]
  (or (apply < a)
      (apply > a))

(def data
  [{:name "Tom" :age 13}
   {:name "Paul" :age 15}
   {:name "Bryan" :age 16}])

(array-sorted? (map :age data)) ; true

(zipmap [:a :b :c :d] [1 2 3 4])
; {:a 1, :b 2, :c 3, :d 4}

(merge-with + {:a 1} {:a 2} {:a 3 :b 1})
; {:a 6 :b 1}
```

Data manipulation: power of the core lib!

>ClojureScript

```
(def cersei {:name "Cersei"})
(def peoples #{{:name "Ned"} cersei})
(peoples cersei) ;;=> true
(peoples {:name "Ned"}) ;;=> true
```

>JavaScript

```
var cersei = {name: "Cersei"};
var peoples = new Set([{name: "Ned"}, cersei]);
peoples.has({name: "Ned"}) //=> false, no reference
peoples.has(cersei) //=> true
```

Data manipulation: power of the core lib!

```clojure
(def data
  [{:name "Ned" :gender :man :home :stark}
   {:name "Cersei" :gender :woman :home :lanister}
   {:name "John" :gender :man :home :stark}
   {:name "Deneris" :gender :woman :home :targaryen}])

;; Uppercase all names
(map #(update % :name clojure.string/upper-case) data)
;;=> [{:name "NED" :home :stark}...]
```

Data manipulation: power of the core lib!

```clojure
(def data
  [{:name "Ned" :gender :man :home :stark}
   {:name "Cersei" :gender :woman :home :lanister}
   {:name "John" :gender :man :home :stark}
   {:name "Deneris" :gender :woman :home :targaryen}])

;; Group by home
(group-by :home data)
;;=> {:stark [{:name "Ned" :home :stark}...]
;;     :lanister [...]}
```

Data manipulation: power of the core lib!

```clojure
(def data
  [{:name "Ned" :gender :man :home :stark}
   {:name "Cersei" :gender :woman :home :lanister}
   {:name "John" :gender :man :home :stark}
   {:name "Deneris" :gender :woman :home :targaryen}])

(def peoples-by-home (group-by :home data)
(defn select-women [items] (filter #(= (:gender %) :woman) items))

;; counting the women by home
(into {}
  (map #(vector (first %) (-> % second select-women count)))
  peoples-by-home)
;;{:stark 0 :lanister 1 :targaryen 1}
```

```clojure
(def data
  [{:name "John Snow"
    :home :stark}
   {:name "Cersei"
    :home :lanister}
   {:name "Aria"
    :home :stark}])

;;Add field age to John Snow
(assoc (nth data 2) :age 15)
(-> data
    (nth 2)
    (assoc :age 15))

;;Upper-case all name
(map
 (fn [item] (update item :name #(clojure.string/upper-case %)))
 data)

;;Only the lanister
(filter #(= :lanister (:home %)) data)

;;Get the number of persons by home
(->> data
     (group-by :home) ;; {:stark [...] :lanister [...]}
     (map (fn [[key items]] (vector key (count items))))
     (into {}))
```

Ressources

- ClojureScript Koans
- Official ClojureScript site
- Online REPL
- Interactive cheat sheet
- Clojure for the Brave and True

Thanks!

Any questions?