St. Louis Clojure

ClojureScript Reagent Tutorial

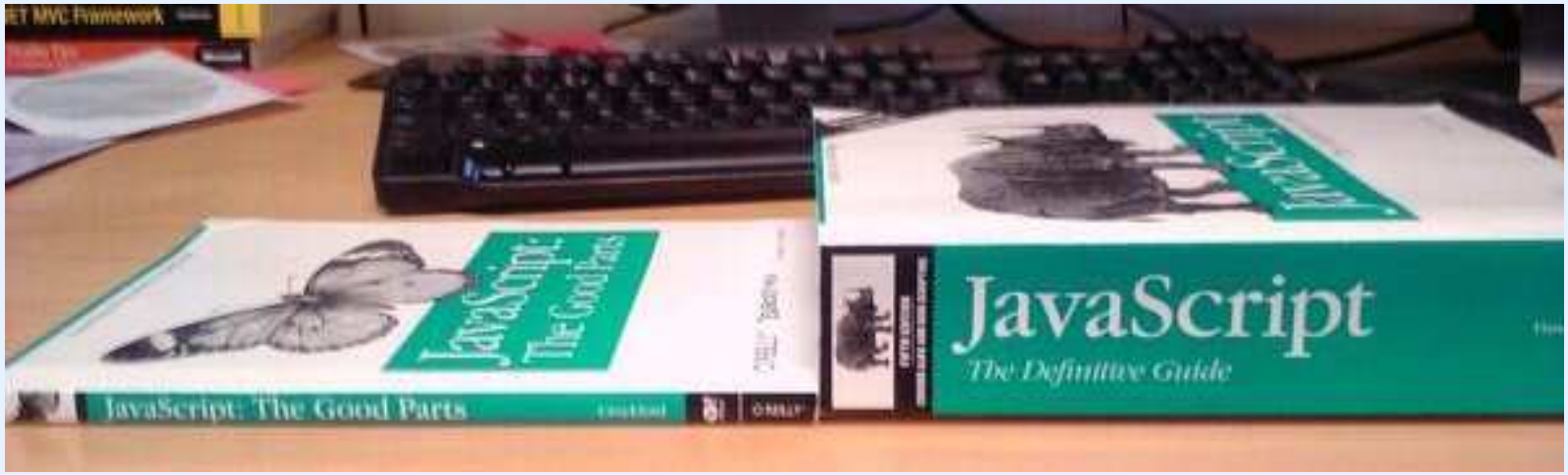
Christopher Mark Gore

cgore.com

Tuesday, May 17, AD 2016

We write Clojure at The Climate Corporation, and we're hiring! Come work with us!

Why ClojureScript? Isn't JavaScript good enough?



ClojureScript lets us use Clojure, a real lisp, in place of JavaScript.

ClojureScript versus JavaScript – Namespaces

JavaScript has no native namespacing.

ClojureScript's namespacing works the same as in Clojure.

One namespace:

```
1 (ns my.library)
2 ...
```

Including another namespace:

```
1 (ns my.library
2   (:require [other.library :as other]))
3 ...
```

ClojureScript versus JavaScript – no variable hoisting

This actually does something in JavaScript other than raise an error, which is probably not what you want:

```
1 function printName() {
2   console.log('Hello, ' + name);
3   var name = 'Bob';
4 }
```

ClojureScript versus JavaScript – destructuring binds

```
1  (def m {:first "Bob"
2          :middle "J"
3          :last "Smith"})
4
5  (let [{:keys [first middle last]} m]
6    ...)
7
8  (def color [255 255 100 0.5])
9
10 (let [[r g _ a] color]
11   ...)
```

ClojureScript versus JavaScript – arbitrary keys for hash maps

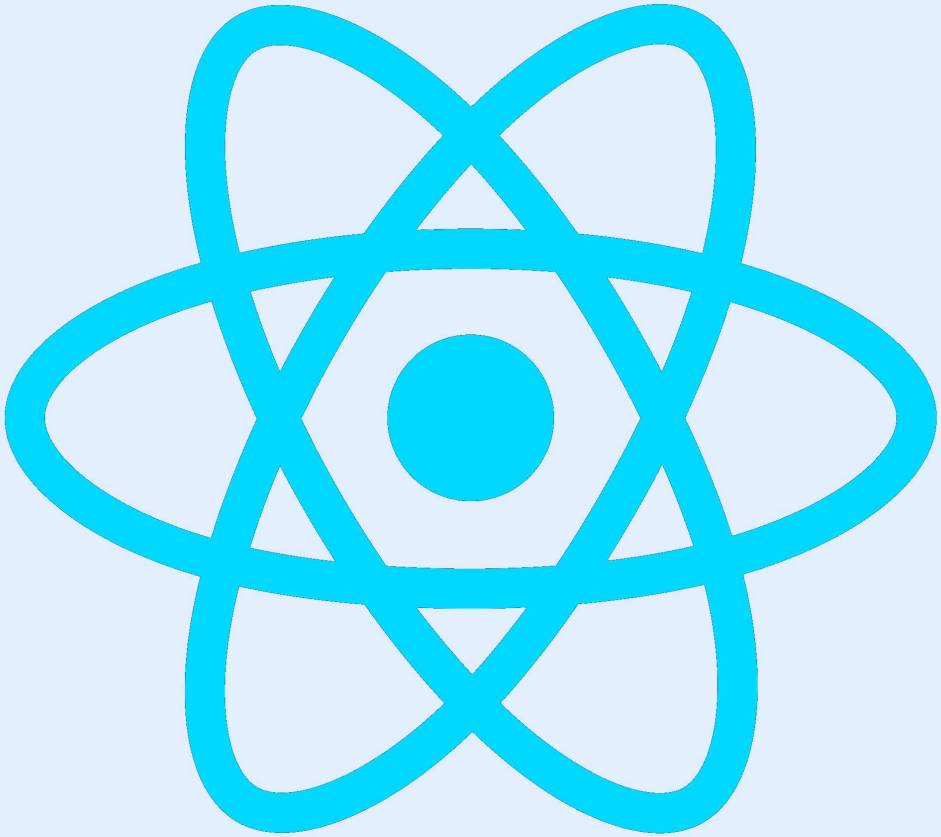
```
1 // JavaScript -- Only string keys allowed
2 var m = {
3   "foo": 1,
4   "bar": 2
5 };
```

```
1 ;; ClojureScript -- Arbitrary keys allowed
2 (def m { [1 2] 3
3          #{1 2} 3
4          '(1 2) 3 })
```

There's lots of other differences too, but they are mostly opinionated things.

- Immutable collections
- Simpler Boolean system
- Optional parameters and arity
- Lazy evaluation
- …

Reagent is a wrapper around React.js for ClojureScript.



Why do we want to use React.js?

- React.js comes out of Facebook originally, but is used everywhere now.
- React.js dates back to 2011.
- React.js is good at creating user interfaces.
- If you are used to MVC, React.js is the **V**.

Why Reagent instead of Om?

- They both wrap React.js in ClojureScript.
- I haven't used either except for tutorials.
- Om has a shorter name, which is better.
- But Reagent has a more awesome name.
- Reagent looks more logical to me.

So I went with Reagent to play with first.

01-initial-project

This is just a simple project stood up via

```
lein new reagent reagent-tutorial
```

You can then do

```
cd reagent-tutorial
lein figwheel
```

And view the page at `http://localhost:3449/`.

02-adding-a-simple-component

We add a simple component:

```
(defn simple-component []
  [:div
   [:p "I am a component!"]
   [:p.someclass
    "I have " [:strong "bold"]
    [:span {:style {:color "red"}} " and red "] "text."]])
```

And use it.

```
(defn home-page []
  [:div [:h2 "Welcome to reagent-tutorial"]
   [:p "Here's a simple component:"
    [simple-component]]
   [:div [:a {:href "#/about"} "go to about page"]]])
```

03-adding-an-atom

In Clojure, mutable state is typically done with an atom.

```
1  (def click-count (atom 0))
2
3  (defn counting-component []
4    [:div
5     "The␣atom␣"
6     [:code "click-count"]
7     "␣has␣value:␣"
8     @click-count ".␣"
9     [:input {:type "button" :value "Click␣me!"
10              :on-click #(swap! click-count inc)}]])
```

04-adding-a-timer

We can have local atoms within let blocks. And we can update those atoms at a regular interval from the clock.

```
1 (defn timer-component []
2   (let [seconds-elapsed (atom 0)]
3     (fn []
4       (js/setTimeout #(swap! seconds-elapsed inc) 1000)
5       [:div
6        "Seconds␣Elapsed:␣" @seconds-elapsed])))
```

05-shared-state-input-box

Often you'll want to update an atom in one place and display it in another.

```
(defn shared-state []
  (let [val (atom "foo")]
    (fn []
      [:div
       [:p "The value is now: " @val]
       [:p "Change it here: " [atom-input val]]])))
```

06-bmi-calculator

This is a bit more complicated. It consists of a function to calculate a basic BMI value:

```
1 (defn calc-bmi []
2   (let [{:keys [height weight bmi] :as data} @bmi-data
3         h (/ height 100)]
4     (if (nil? bmi)
5       (assoc data :bmi (/ weight (* h h)))
6       (assoc data :weight (* bmi h h)))))
```

06-bmi-calculator

A slider:

```
1  (defn slider [param value min max]
2    [:input {:type "range"
3             :value value
4             :min min
5             :max max
6             :style {:width "100%"}
7             :on-change
8               (fn [e]
9                 (swap! bmi-data assoc param
10                       (.-target.value e))
11                (when (not= param :bmi)
12                  (swap! bmi-data assoc :bmi nil)))}])
```

06-bmi-calculator

And finally the actual BMI component:

```
(defn bmi-component []
  (let [{:keys [weight height bmi]} (calc-bmi)
        [color diagnose] (cond (< bmi 18.5) ["orange" "underweight"]
                               (< bmi 25) ["inherit" "normal"]
                               (< bmi 30) ["orange" "overweight"]
                               :else ["red" "obese"])]
    [:div
     [:h3 "BMI calculator"]
     [:div
      "Height: " (int height) "cm"
      [slider :height height 100 220]]
     [:div
      "Weight: " (int weight) "kg"
      [slider :weight weight 30 150]]
     [:div
      "BMI: " (int bmi) " "
      [:span {:style {:color color}} diagnose]
      [slider :bmi bmi 10 50]]]))
```

Conclusion

I ended up going with JavaScript after all for the project I was originally looking into this for. But it's a bit of a special case and will need to generate JavaScript itself, sort of like ClojureScript. If you don't need that *(you probably don't)* then ClojureScript and Reagent might be a really good fit.

Questions?