Blog & Video

Curious about ClojureScript, but not sure how to use it


You task for Tuesday:
  Implement Tick-tack-toe from the video
ClojureScript

No
  Refs
  Agents

def - creates Javascript variable

Google Closure library - optimize

Numbers
  integer & floating point only
  Equality from Javascript
    (= 0.0 0) => true

:private - not enforced
:const - can not redefine

fn
  no runtime check for arity

Most but not all collection fns are implemented

Almost all Seq library functions are available in ClojureScript

Foo/bar always means that
Foo is a namespace

To access JS object properties
use a leading hyphen

.-target .-value
Some Examples
Set Up

In core

(ns firstreagent.core
  (:require [reagent.core :as reagent :refer [atom]]
    [reagent.session :as session]
    [secretary.core :as secretary :include-macros true]
    [goog.events :as events]
    [goog.history.EventType :as EventType]
    [firstreagent.events :as e])
  (:import goog.History))

(secretary/defroute "/events" []
  (session/put! :current-page #'e/main))
(ns firstreagent.events
  (:require [reagent.core :as r] ))

(defn atom-input [value]
  [:input {:type "text"
           :value @value
           :on-change (fn [event] (reset! value (-> event .-target .-value)))}])

(defn main []
  (let [val (r/atom "foo")]
    (fn []
      [:div
       [:p "The value is now: " @val]
       [:p "Change it here: " [atom-input val]]])))
[:input {:type "text"
    :value @value
    :on-change (fn [event] (reset! value (-> event .-target .-value))))]

<input on-change= "firstreagent.repl$eval13805$fn__13806@3c5b5bae"
    type="text"
    value="cat" />
All Three Run

[::input {:type "text"
    :value @value
    :on-change (fn [event] (reset! value (-> event .-target .-value)))}
]

[::input {:type "text"
    :value @value
    :on-change (fn [] (reset! value "Cat"))}
]

[::input {:type "text"
    :value @value
    :on-change (fn [event foo] (reset! value foo)))}]

Thursday, November 12, 15
The Correct handler is Called

(def value (r/atom "foo"))

(defn handler
  ([] (reset! value "None"))
  ([event] (reset! value "One"))
  ([event foo] (reset! value "Two"))
)

(defn atom-input [value]
  [:input {:type "text"
    :value @value
    :on-change handler}]
)

(defn main []
  (fn []
    [:div
      [:p "The value is now: " @value]
      [:p "Change it here: " [atom-input value]]]))
Some DOM Events

<table>
<thead>
<tr>
<th>Mouse Events</th>
<th>Keyboard Events</th>
<th>Form Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>onclick</td>
<td>onkeydown</td>
<td>onblur</td>
</tr>
<tr>
<td>oncontextmenu</td>
<td>onkeypress</td>
<td>onchange</td>
</tr>
<tr>
<td>ondblclick</td>
<td>onkeyup</td>
<td>onfocus</td>
</tr>
<tr>
<td>onmousedown</td>
<td></td>
<td>onfocusin</td>
</tr>
<tr>
<td>onmouseenter</td>
<td></td>
<td>onfocusout</td>
</tr>
<tr>
<td>onmouseleave</td>
<td></td>
<td>oninput</td>
</tr>
<tr>
<td>onmousemove</td>
<td></td>
<td>oninvalid</td>
</tr>
<tr>
<td>onmouseover</td>
<td></td>
<td>onreset</td>
</tr>
<tr>
<td>onmouseout</td>
<td></td>
<td>onsearch</td>
</tr>
<tr>
<td>onmouseup</td>
<td></td>
<td>onselect</td>
</tr>
<tr>
<td>onmouseout</td>
<td></td>
<td>onsubmit</td>
</tr>
</tbody>
</table>

Lot more at

http://www.w3schools.com/jsref/dom_obj_event.asp
DOM -> Reagent event names

onchange -> on-change

onmousemove -> on-mouse-move
DOM Event Objects

Properties

bubbles
cancelable
currentTarget
defaultPrevented
eventPhase
isTrusted
target
timeStamp
type
view
Bubbling

If an event occurs in d3

It is sent to the element d3

Then to element d2

Then to element d1

To stop bubbling

event.stopPropagation() ;; All modern browsers except IE

event.cancelBubble = true ;; IE
# MouseEvent & KeyEvent Objects

### MouseEvent Properties

- altKey
- button
- buttons
- clientX
- clientY
- ctrlKey
- detail
- metaKey
- relatedTarget
- screenX
- screenY
- shiftKey
- which

### KeyEvent Properties

- altKey
- ctrlKey
- charCode
- key
- keyCode
- metaKey
- shiftKey
- which
More Dom Events

http://www.w3schools.com/jsref/dom_obj_event.asp
List, tutorial

http://quirksmode.org/dom/events/index.html

Browser compatibility
Second Example

In core

(ns firstreagent.core
  (:require [reagent.core :as reagent :refer [atom]]
    [reagent.session :as session]
    [secretary.core :as secretary :include-macros true]
    [goog.events :as events]
    [goog.history.EventType :as EventType]
    [firstreagent.events :as e])
  (:import goog.History))

(secretary/defroute "/events" []
  (session/put! :current-page #'e/main))
(ns firstreagent.events
  (:require [reagent.core :as r]))

(defn main []
  (let [value (r/atom {:x "none" :y "none"})]
    (fn []
      [:div {:on-mouse-move #(reset! value {:x (.-clientX %) :y (.-clientY %)})}
       [:p "X: " (:x @value) " Y: " (:y @value)]
       [:p "Move the mouse between here"]
       (repeat 3 [:br])
       [:p "and here"]))))

X: none Y: none

Move the mouse between here

and here
Undo

Undo (0)

X: none Y: none

Move the mouse between here

and here

Thursday, November 12, 15
(def location (r/atom {:x "none" :y "none"}))

(def undo-list (r/atom nil))

(defn undo []
  (let [undos @undo-list]
    (when-let [old (first undos)]
      (reset! location old)
      (reset! undo-list (rest undos)))))

(defn undo-button []
  (let [n (count @undo-list)]
    [:input {:type "button" :on-click undo
             :disabled (zero? n)
             :value (str "Undo (" n ")")}])))
(defn track-mouse []
  [:div {:on-mouse-move #(reset! location {:x (-clientX %) :y (-clientY %)})}
   [:p "X: " (:x @location) " Y: " (:y @location)]
   [:p "Move the mouse between here"]
   (repeat 5 [:br])
   [:p "and here"]])

(defn main []
  (add-watch location ::undo-watcher
    (fn [__ old-state __]
      (swap! undo-list conj old-state))
    [:div
      [undo-button]
      [track-mouse]]
  )
)
In ClojureScript print output will appear in the browser’s JavaScript console.

(defn main []
  (let [value (r/atom {:x "none" :y "none"})]
    (fn []
      [:div {:on-mouse-move #(reset! value {:x (.-clientX %) :y (.-clientY %)})}
       [:p "X: " (:x @value) " Y: " (:y @value)]
       [:p "Move the mouse between here"]
       (repeat 3 [:br])
       (print "this is a test")
       [:p "and here"]))))
In Chrome
Some Details
Core of a Component

Render function

Input some data

Returns Hiccup vector that will be converted to HTML
Three Ways to Create a Component

Render function
   Form-1 component

Function that returns a render function
   Form-2 component

Map of functions, one of which is the render function
   Form-3 component
Render function - Form-1 Reagent Component

(defn greet
  [name] ;; data coming in is a string
  [:div "Hello " name])

(defn wrong-component
  [name]
  [:div "Hello"] [:div name])

(defn right-component
  [name]
  [:div
   [:div "Hello"
    [:div name]]])
Form-2 Reagent Component

Function that returns a render function

(defn timer-component []
  (let [seconds-elapsed (reagent/atom 0)] ;; setup, and local state
  (fn [] ;; inner, render function is returned
    (js/setTimeout #(swap! seconds-elapsed inc) 1000)
    [:div "Seconds Elapsed: " @seconds-elapsed])))

timer-component is called once per component instance

The render function it returns will potentially be called many, many times
Rookie mistake

(defn outer
  [a b c] ;; <--- parameters
  ;; ....
  (fn [a b c] ;; <--- forgetting to repeat them, is a rookie mistake
    [:div
      (str a b c)]))

Explain why
React Component
# React Component - Relevant Parts

<table>
<thead>
<tr>
<th>Data</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>props (properties)</td>
<td>render (required)</td>
</tr>
<tr>
<td>Arguments/parameters</td>
<td>getInitialState</td>
</tr>
<tr>
<td>state</td>
<td>getDefualtProps</td>
</tr>
</tbody>
</table>

We will not see these two

create-class
Constructor

render function called when props or state change
React Component Lifecycle Methods

componentWillMount
  Called once

cOMPONENT DidMount
  Called once

cOMPONENT WiliReceiveProps
  Called when receiving new props

shouldComponentUpdate
  Return false to cancel update

cOMPONENT WillUPDATE
  Called before update

componentDidUpdate
  Called after update

cOMPONENT WILL Unmount
Form-3 Reagent Component

Map of functions
  render function
  Some React component lifecycle methods

(defn my-component
  [x y z]
  (let [some (local but shared state) ;<-- closed over by lifecycle fns
can  (go here)]
    (reagent/create-class ;<-- expects a map of functions
      {:component-did-mount ;; the name of a lifecycle function
        #(println "component-did-mount") ;; your implementation

        :component-will-mount ;; the name of a lifecycle function
        #(println "component-will-mount") ;; your implementation

        ;; other lifecycle funcs can go in here

        :display-name "my-component" ;; for more helpful warnings & errors
When Do Components Update
Reagent Component are Reactive

Each Component has a render function

Render function turns input data into hiccup (HTML)

Render functions are rerun when their input data changes, producing new hiccup

New hiccup is "interpreted" by Reagent and ultimately results in new HTML
Two Types of Input

props
ratoms - Reagent atoms
Props

(defn greet [name] ;; name is a string
[:div "Hello " name])

Name is a prop (property)

greet will be called each time name changes
(defn greet
    [name]
    [:div "Hello " name])

(defn greet-family
    []
    [:div
        [greet "Dad"]
        [greet (str "Bro-" (rand-int 10))]]])

[greet "Dad"] - rendered once

[greet (str "Bro-" (rand-int 10))]
    9 times out of ten rerendered when parent is rerendered

Each time greet-family is rendered
Is subcomponents are checked
If there props have changed
rerender them
Ratoms

(def name (reagent.ratom/atom "Bear"))

(defn ask-for-forgiveness
  [] ;; <--- no props
  [:div "Please " @name " with me"])

ask-for-forgiveness
  rerendered when @name changes
parent-renderer is rerun

What happens when button is pressed?
greet-number’s prop has changed
so rerun

more-button is not rerun

(defn parent []
  (let [counter (reagent.ratom/atom 1)] ;; the render closes over this state
    (fn parent-renderer []
      [:div
        [more-button counter] ;; no @ on counter
        [greet-number @counter]]))
(defn greet-number
  "I say hello to an integer"
  [num] ;; an integer
  [:div (str "Hello #" num)])

(println "What happens when button is pressed?

(defn more-button
  [counter] ;; a ratom
  [:div {:class "button-class"
        :on-click #(swap! counter inc)} ;; increment the int value in counter
    "More"])

(defn parent
  []
  (let [counter (reagent.ratom/atom 1)] ;; the render closes over this state
    (fn parent-renderer
      []
      [:div
       [more-button counter] ;; no @ on counter
       [greet-number @counter]])))
When are things Equal?

(def x1 {:a 42 :b 45})
(def x2 {:a 42 :b 45})

(= x1 x2) ;; => true

(identical? x1 x2) ;; => false

= are values same
Java equals

identical?
point to the same structure
Java ==

use to compare props

identical?
Used to compare value inside ratoms
Lifecycle Functions

prop changes trigger all lifecycle methods

ratoms changes do not trigger lifecycle methods