FREE eBook

LEARNING clojurescript

Free unaffiliated eBook created from
Stack Overflow contributors.

#clojurescri
pt

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: clojurescript

It is an unofficial and free clojurescript ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official clojurescript.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with clojurescript

Remarks

ClojureScript is a version of Clojure that compiles to javascript code and runs in the browser.

While it is mostly the same as Clojure, there are a few differences (mainly, it lacks some api calls that invoke java functions that are not available in javascript)

Examples

Installation or Setup

Leiningen

Note: If you're going to use Leiningen, you first need to download and install JDK 6 or newer.

The easiest way to get started with Clojure is to download and install Leiningen, the de facto standard tool to manage Clojure projects.

Linux:

```
curl https://raw.githubusercontent.com/technomancy/leiningen/stable/bin/lein > ~/bin/lein
export PATH=$PATH:~/bin
chmod 755 ~/bin/lein
lein
```

OS X:

Follow Linux steps above or

Install with [Homebrew]:

```
brew install leiningen
```

Windows:

See https://github.com/technomancy/leiningen#installation.

Connecting to a REPL

Once you have `lein` installed, execute

```
lein repl
```

Start a new ClojureScript Project

```
lein new mies myproject
cd myproject
./scripts/build
```

Now open `index.html` in a Web Browser.

Press F12

Look at the Console, and observe the following:

```
Hello world!
```

Acknowledgements

- David Nolen's extraordinary work on ClojureScript.

Read Getting started with clojurescript online: https://riptutorial.com/clojurescript/topic/1198/getting-started-with-clojurescript

Chapter 2: Figwheel

Remarks

Figwheel automatically rebuilds your clojurescript code when source files change and reloads code in browser. Reload works without refreshing page and you can preserve some of app's state between reloads by using `defonce`.

It is alternative to REPL-based development (although it includes REPL too). Instead of re-evaling changed functions in REPL, it reloads all code, and you can use `println` to see result of expression evaluation in browser's js console.

Examples

Creating a New Project

Create a new project with the Leiningen `figwheel` template:

```
lein new figwheel hello-world
```

Run Figwheel:

```
cd hello-world
lein figwheel
```

After a moment it will start a development webserver and open the page in your browser.

It also opens a Clojurescript REPL connected to the browser. Try entering `(js/alert "Test")`: you should see an alert pop up in the browser.

Try editing `hello_world/core.cljs`. It will automatically refresh in your browser when you save it. You can use reload functionality similarly to the REPL by adding `print`s and seeing results in the Javascript console.

Read Figwheel online: https://riptutorial.com/clojurescript/topic/3963/figwheel

Chapter 3: Getting Started with Reagent

Introduction

Reagent is a library that implements elements of React.js into ClojureScript, like the creations of custom "tags", which is implemented in Reagent through functions.

Examples

Reagent Atoms

Reagent atoms are essentially the same as regular atoms in both Clojure and ClojureScript - they are essentially variables that can be altered. This is especially useful, as Clojure(Script)'s data types are mostly immutable - which means that to change the value of a variable, the variable has to be redeclared.

Normal atoms are incompatible with Reagent, and so Reagent has its own. They are declared like normal variables, except with an additional function wrapped around the value:

```
(:require [reagent.core :as r])

(def num (r/atom 1))
```

You can get the value of an atom in two ways:

```
(deref num) ; => 1
@num        ; => 1
```

To change the value of an atom, there are two commands, `swap!` and `reset!`.

- `swap!` is given commands, changes the atom's value based on the original value of the atom itself
- `reset!` is given a value, and changes the atom's value to the value given, regardless of what the atom was originally

```
(swap! num inc) ; => (inc num) => num = 2
(reset! num 5)  ; => num = 5
```

UI using Reagent and Hiccup

Reagent is an interface between ClojureScript and react. It allows you to define efficient React components using nothing but plain ClojureScript functions and data, that describe your UI using a Hiccup-like syntax.

Example:-

```
(defn sample-component []
  [:div
   [:p "I am a component!"]
   [:p.someclass
    "I have " [:strong "bold"] "text."]])
```

Output:

I am a component!

I have **bold** text.

Read Getting Started with Reagent online: https://riptutorial.com/clojurescript/topic/8939/getting-started-with-reagent

Chapter 4: JavaScript Events

Syntax

- (goog.events dom-element event-type event-handler-function) ;;Creates a Google Closure event listener
- (.addEventListener dom-element load-event) ;;Creates normal JavaScript event listener. Can be browser specific.

Remarks

All Closure event names can be found in their documentation on the EventType enum.

Examples

Adding Event to Button Using Closure Library

```
(ns so-doc.events
  (:require
   [goog.dom :as dom]
   [goog.events :as events]))


(defn handle-click [event] ;an event object is passed to all events
  (js/alert "button pressed"))

(events/listen
 (dom/getElement "button"); This is the dom element the event comes from
 (.-CLICK events/EventType); This is a string or array of strings with the event names.
;;All event names can be found in the EventType enum
 handle-click ;function that should handle the event
 )
```

Google Closure does not support page-load events, and considers them to be not idiomatic. They recommend inserting scripts inline as soon as the content they to access has loaded.

Using JavaScript Interop

```
(ns so-doc.events)

(enable-console-print!)

(defn click-event []
  (println "Button clicked"))

(defn load-event []
  (println "Page loaded!")
  (.addEventListener (.getElementById js/document "btn") "click" click-event false))
```

```
(.addEventListener js/window "load" load-event false)
```

Like normal Javascript, this method requires browser specific handling. This will not work in Internet Explorer for instance.

Unlike Google Closure, JavaScript easily supports page-load events.

Read JavaScript Events online: https://riptutorial.com/clojurescript/topic/6653/javascript-events

Chapter 5: lein-cljsbuild

Remarks

More details on parameters for lein-clsjbuild can be found in their example project.

Examples

ClojureScript dev and production build

Add a `:cljsbuild` node like the following to your project.clj file.

```
:cljsbuild {
          :builds {
                   ;;Different target goals should have different names.
                   ;;We have the dev build here
                   :dev {
                        ;;The ClojureScript code should reside in these directories
                         :source-paths ["src-cljs"]
                         :compiler {
                                    ;;This is the target output file
                                    ;;This will include none of the goog code.

                                    :output-to "resources/public/js/main.js"
                                    ;;This stores the compilation files, including goog files.
                                    ;;When using this dev profile, you should add "<script
src="/js/goog/base.js>" to your html files.
                                    ;;That will load the goog dependency.
                                    :output-dir "resources/public/js/out"
                                    ;;Having optimizations off is recommended for development
                                    ;;However this means you have to add the goog dependencies
yourself (see above)
                                    :optimizations :none
                                    ;;With no optimizations, marking this true, means it adds
a source map so you can debug in browser
                                    :source-map true
                                    }
                         }
                   :prod {
                          :source-paths ["src-cljs"]
                          :compiler {
                                     ;;Outputs the javascript file to this path.
                                     :output-to "resources/public/js/main.js"
                                     ;;Advanced optimizations make your code smaller.
                                     ;;They include the goog code in your main file.
                                     ;;No need to add an extra script tag.
                                     :optimizations :advanced
                                     :jar true
                                     }
                          }
                   }
          }
```

To run these build commands, use `lein cljsbuild once`. You can specify which profile to use by adding the profile name as the last argument, for example `lein cljsbuild once dev` to compile with dev parameters.

Using `lein cljsbuild auto` will cause cljsbuild to automatically update any changed files.

Read lein-cljsbuild online: https://riptutorial.com/clojurescript/topic/6093/lein-cljsbuild

Chapter 6: State Management with re-frame (https://github.com/Day8/re-frame)

Introduction

It starts to become difficult when we think clojurescript as a functionally pure language, that holds state for it UI components. It is simply, not possible.

However, it is possible to separate out individual components as well as their states. We can do it by storing data/state in reagent/atom. But when there are lots of states and lots of dependencies, things quickly become confusing and we start to wish for a out of the box solution for our state management. This is where re-frame comes in.

Examples

1. Simple Dispatch event

We will look at a simple dispatch event with the example usage.

```
(ns myapp.events

  (:require [re-frame.core :refer [reg-event-db]]))



(reg-event-db

 :enable-feature-toggle

 (fn [db [_ _]]

   (assoc-in db [:global :enable-feature-toggle] true)))
```

This is creating a event called `:enable-feature-toggle` which will create a entry in db `{:global {:enable-feature-toggle true}}`. A *db* is like a global store for preserving the event outcomes. These outcomes can then be subscribed to, and used to modify the state of the element as we will see in the **Simple Subscribe Event** example.

After creating the event we actually have to dispatch the event somewhere in our code/UI components to make some use of it. For example, if we want to do some action on click of a div, we can fire an on-click event.

```
(ns myapp.components.page-header

  (:require [myapp.events]
            [re-frame.core :refer [dispatch]]))
```

```
(defn cljs-header []
    [:div {:class "cljs-header"

           :on-click #(dispatch [:enable-feature-toggle])} "Click Me"])
```

As soon as we click on the div, the :enable-feature-toggle event will be fired and a new value will be set in the db.

Read State Management with re-frame (https://github.com/Day8/re-frame) online: https://riptutorial.com/clojurescript/topic/9932/state-management-with-re-frame--https---github-com-day8-re-frame-

Credits

S. No	Chapters	Contributors
1	Getting started with clojurescript	Community, hawkeye, Matt W-D, Zain Rizvi
2	Figwheel	kolen, porglezomp
3	Getting Started with Reagent	Mrinal Saurabh, Qwerp-Derp, Supriya Gole
4	JavaScript Events	DonyorM
5	lein-cljsbuild	DonyorM
6	State Management with re-frame (https://github.com/Day8/re-frame)	Mrinal Saurabh