Ildar Sagdejev

Adventures in software engineering

Website

December 7, 2016 · CLOJURESCRIPT BOOT

Starting a Node.js app with ClojureScript and Boot

I set out to write a Node.js application in ClojureScript, using Boot as my build tool. Because these tools are still young and rapidly evolving, I had to sort through an abundance of incomplete, misleading or out-of-date information to figure it out. For reference, here is an excellent article on how to do the same thing using Leiningen.

tl;dr - here's the quickstarter kit.

Let's start off by creating a file `src/app/core.cljs`:

```
1  (ns app.core
2    (:require [cljs.nodejs :as nodej
3  s]))
4  (nodejs/enable-util-print!)
5
6  (defn main [& args]
7    (println "Abracadabra!"))
8
9  (set! *main-cli-fn* main)
```

The namespace `app.core` reflects the source file `core.cljs` placed in the `app/` directory. We can compile this directly using the ClojureScript compiler:

Ildar Sagdejev

Adventures in software engineering

Website


```
$ cljsc src/app/core.cljs '{:target
```

There are a number of ways to install the ClojureScript compiler, but **building it from source** worked best for me. If all went well, you can run your app with:

```
$ node out/main.js
```

Using the ClojureScript compiler manually can be tricky. If you can't quite get it to work, don't worry. We're going to use the flexible Boot build tool to automate this process.

Let's create a `build.boot` project file:

```
(set-env!
  :source-paths #{"src"}
  :dependencies '[[adzerk/boot-cljs
"1.7.228-2" :scope "test"]])
(require
  '[adzerk.boot-cljs :refer [cljs]])
```

You can adjust the `boot-cljs` version to reflect the **latest** on Clojars. The point of Boot is to be a flexible build automation system that lets you assemble your own pipeline using a rich assortment of independent composable **tasks**.

Let's define a file watcher task that responds to changes by compiling our Node.js app with source mapping enabled:

```
(deftask dev
  "Watch/compile files in developmen
t"[]
  (comp
    (watch)
```

Ildar Sagdejev

Adventures in software engineering

Website


```
6       (cljs :source-map true
7             :optimizations :none
8             :compiler-options {:target
9   :nodejs}))
```

Running `boot dev` should now look something like this:

```
1   $ boot dev
2
3   Starting file watcher (CTRL-C to qui
4   t)...
5   Writing main.cljs.edn...
6   Compiling ClojureScript...
7   • main.js
8   Writing target dir(s)...
9   Elapsed time: 9,342 sec
```

The app should now appear in the `target/` directory, which is created by the `target` task, introduced as a standalone operation in Boot 2.5 (See: **Boot 2.5: Slow is smooth, smooth is fast**). You must `cd` into the `target` directory to run the compiled app because at this point it relies on the relative paths of the dependencies generated by the Google Closure Compiler in the `target/main.out/` directory.

```
1   $ cd target
2   $ node main.js
```

For building the final product, let's add a `prod` task:

```
1   (deftask prod
2     "Compile for production"
3     []
```

Ildar Sagdejev

Adventures in software engineering

Website


```
4     (comp
5       (cljs :optimizations :advanced
6             :compiler-options {:target
7   :nodejs})))
```

You are now ready to start writing your Node.js application in ClojureScript. This guide has not covered installing JavaScript dependencies or the various other options you can pass to the compiler. Note that we could have accomplished the same build process with the command line:

```
1  # dev build
2  $ boot watch cljs -c '{:target :nod
3  ejs}' target
4  # prod build
5  $ boot cljs -c '{:target :nodejs}'
```

However, a `build.boot` file will allow you to **expand** your build configuration as your project evolves. More information about `boot-cljs` is available on the **wiki**, though as usual some of it is geared specifically toward JavaScript in the browser.

Ildar Sagdejev

Adventures in software engineering

Website

Comments Community 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

Pavlos Vinieratos • 3 months ago

that was nice. thank you!

Reply • Share ›

ryancole • 5 months ago

Thanks for the blog post. It's very concise and clearly explains the process. I've got one question, though. What does it mean to "write your Node.js application in ClojureScript"? Does ClojureScript, by default, only target browser (because of Google Closure Compiler), and this process opens ClojureScript up to running on Node.js? Would this application still benefit from any of the Clojure (Java) features, or is it now constrained to Node.js run-time only? Thanks! I'm new to Clojure, trying to figure out if I'd like to try writing a website on ClojureScript. (still trying to figure out how'd I'd achieve client-side URL routing, and server-side rendering)

Reply • Share ›