One () to Rule Them All

Aaron Bedra
Relevance, Inc.

I have a double agenda

But first let’s talk about Clojure(Script)

We have this great language with rich data structures

It can help us solve lots of problems

Web problems are included in the set of all problems...

A short aside...

Clojure on the Web

Ring

http endpoints are functions

```
{request} -> handler -> {response}
```

basic handler

```
(defn hello-world [request]
  (let [{:keys [request-method uri]}
        request]
    {:status  200
     :headers {}
     :body (str "hello, "
                request-method
                " "
                uri)}))
```

request
keys

response
keys

return nil to ignore inputs

```
(defn hello-world [request]
  (let [{:keys [request-method uri]}
        request]
    (when (and (= request-method :get)
               (= uri "/"))
      {:status  200
       :headers {}
       :body    "The index page"})))
```

test for whatever
you care about

Compojure

a little macro magic later...

```
(defroutes routes
  (GET "/" [] "The index page"))
```

running embedded

```
(ns training.web
  (:use [ring.adapter.jetty :only (run-jetty)]
        [compojure.core :only (defroutes GET)]))

(defroutes routes
  (GET "/" [] "<h2>Hello World</h2>"))

(run-jetty routes {:port 8080
                   :join? false})
```

Middleware

middleware

```
(defn wrap-cookies
  [handler]
  (fn [request]
    (let [request (if (request :cookies)
                    request
                    (assoc request :cookies
                           (parse-cookies
                            request)))]
      (-> (handler request)
          (set-cookies)
          (dissoc :cookies)))))
```

call original handler

modify the result

17

common ring middleware

with-params

with-keyword-params

with-cookies

with-multipart

with-session

So common that Compojure wraps them for you

Exposing an API

```
(defn api
  [routes]
  (-> routes
      wrap-keyword-params
      wrap-nested-params
      wrap-params))
```

Exposing a Site

```clojure
(defn site
  [routes & [opts]]
  (-> (api routes)
      (with-opts
        wrap-multipart-params
        (:multipart opts))
      (with-opts wrap-session (:session opts))))
```

html
(hiccup)

html elements

clojure
vector

```
(html [:h1 "hi"])
-> "<h1>hi</h1>"
```

html attributes

clojure
map

```
(html [:a
        {:href "http://clojure.org"}
        "Clojure"])
```

```
<a href="http://clojure.org">Clojure</a>
```

id, class shortcuts

id
follows #

class
follows .

```
(html [:h1#title.main "hi"])
```

```
<h1 class="main" id="title">hi</h1>
```

simple composition

```
(defn home []
  (layout/home
   [:ul
    (map
     (fn [lab] [:li (make-url lab)])
     all)]))

(defroutes lab-routes
  (GET "/" [] (home)))
```

mix clojure literals...

...with fncalls

and call them from routes

composable routing

```
(defroutes lab-routes
  (GET "/" [] (home))
  (GET "/labs/:name" [name] (render-lab name))
  (route/files "/")
  (route/not-found "<h1>Not Found</h1>"))

(def application (-> lab-routes
                     handlers/with-logging))
```

compose routes

simple function wrapping

implementation comparison

feature	clojure impl	oo impl
endpoint	function	interfaces, classes
request	map	interfaces, classes
response	map	interfaces, classes
cookies	map	interfaces, classes
session	map	interfaces, classes
routing	functions, macros	interfaces, classes, config, XML
middleware	functions, macros	interfaces, classes, config, XML, AOP

fns are easy to test!

```clojure
(deftest render-the-labs
  []
  (doseq [lab all]
    (let [url (lab-url lab)
          resp (application {:request-method :get
                                       :uri url})]
      (is
       (= {:status 200
           :headers
           {"Content-Type" "text/html;
                                  charset=utf-8"}}
          (select-keys resp
                       [:status :headers]))))))
```

It turns out there's actually a lot of ways to solve problems on the web

Except we are a little light in one area

We are all hopelessly polyglot except when it comes to client side browser code

No matter what we use for our backends we all unify on JavaScript*

Why?

Clojure rocks, JavaScript reaches

So we took Clojure on the road

Yep, CoffeeScript already did it

But there’s so much more

ClojureScript has a full Clojure reader

Clojure data is much more powerful than JSON or XML

With ClojureScript you can adopt Clojure data as your wire protocol

And there’s a hidden gem

Closure

????

ClojureScript works with Closure's advanced compiler

But wait, there's more!

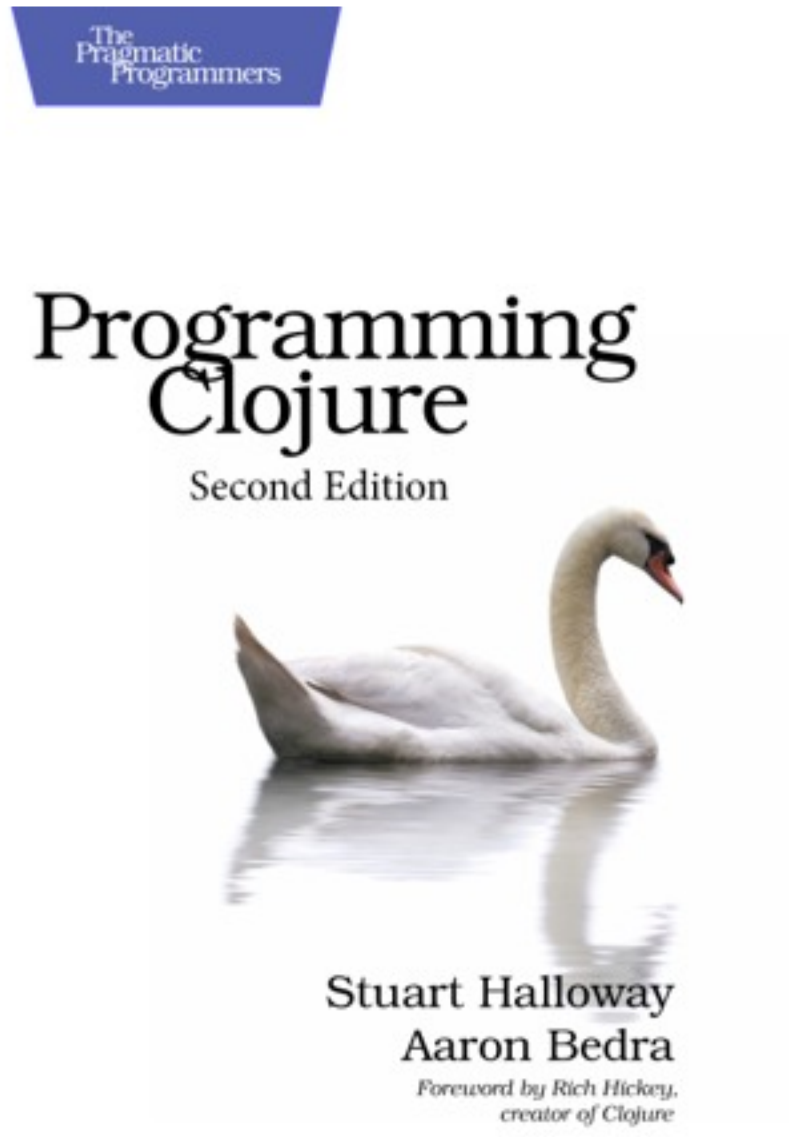
Browser connected REPL

DEMO

Questions?

Additional information

- aaron@clojure.com
- @abedra
- thinkrelevance.com


The Pragmatic Programmers
Programming Clojure
Second Edition
Stuart Halloway
Aaron Bedra
Foreword by Rich Hickey, creator of Clojure