Clojure and the Web

Glenn Vanderburg
InfoEther
glenn@infoether.com
@glv

Clojure

Clojure

- Modern dialect of Lisp
- Runs on the JVM
- Gives priority to performance and concurrency

Surface

- Extra literals (maps, vectors, sets, regexps)
- Metadata
- Renamed / simplified traditional functions
- Sequences
- Destructuring

Underneath

- Concurrent data structures
- STM
- Agents
- Atoms and dynamic vars
- Lazy evaluation

Java Integration

- Java methods appear to be single-dispatch generic functions.
- I'd rather write Java in Clojure than in Java.

```
(.size props)
(.put props "key" value)

(let [conn (doto (HttpUrlConnection. url)
               (.setRequestMethod "POST")
               (.setDoOutput true)
               (.setInstanceFollowRedirects true))]
  ; ...
)
```

Philosophy

- Pragmatic
- Encourages a functional style
- Allows for compromise

Clojure Web Development

- Cascade
- Clothesline
- Compojure
- Conjure
- Ring
- Twister
- Webjure

HTML Generation

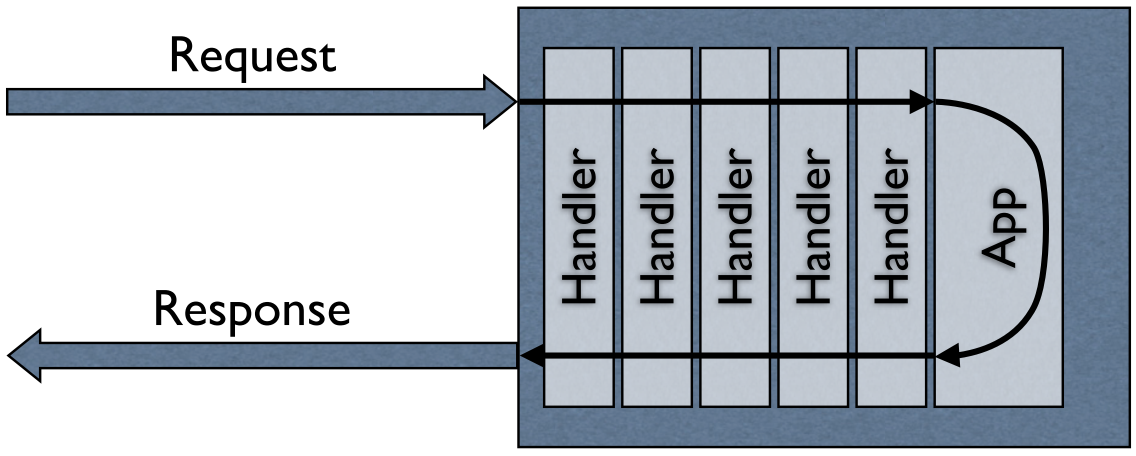
- clj-html
- Enlive
- Fleet
- Hiccup

Persistence, etc.

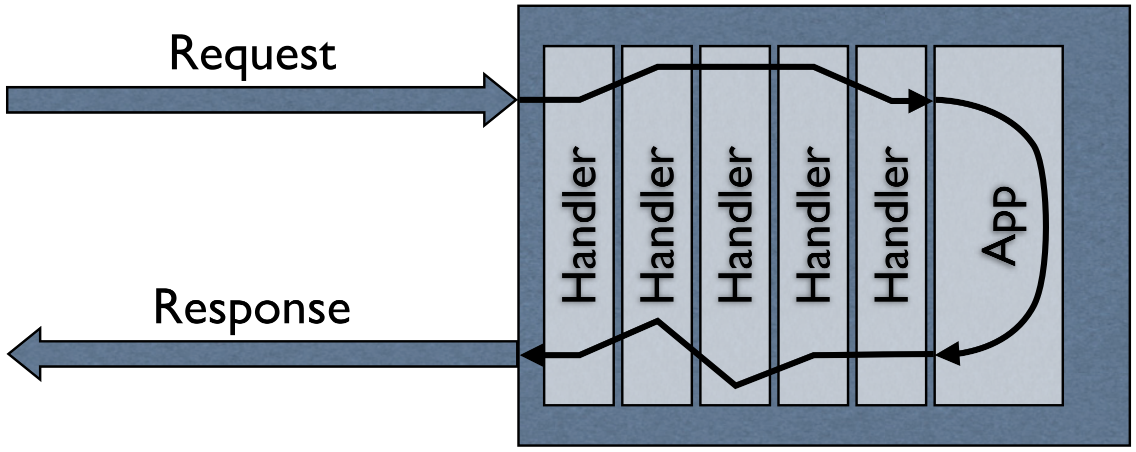
- Plenty of options
- Obligatory “plus Java libraries” plug

Ring

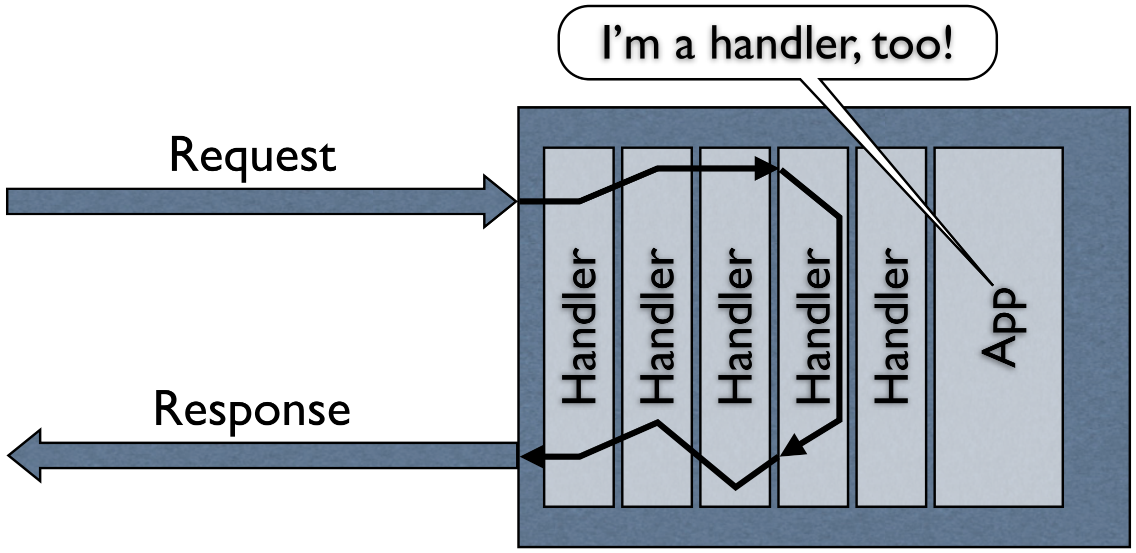
Architecture



Architecture



Architecture



Request

```
{ :server-port        80
  :server-name        "example.com"
  :remote-addr        "127.0.0.1"
  :uri                "/help"
  :query-string       "search=ring"
  :scheme             "http"
  :request-method     :get
  :headers            {"accepts" "..."}
  :body               nil ; an InputStream
  :content-type       nil
  :content-length     nil
  :character-encoding nil }
```

Response

```
{ :status  200
  :body    "hi" ; String, seq, File, or InputStream
  :headers {"content-type" "..."}  }
```

Apps

```
(defn app [request]
  { :status 200
    :headers {"Content-Type" "text/html"}
    :body "<h1>OMG HI!</h1>" })
```

Apps

```
(defn hello [request]
  (if (= "/hello" (:uri request))
    { :status 200
      :headers {"Content-Type" "text/html"}
      :body "<h1>OMG HI!</h1>" }

    (-> (response "<h1>NOT FOUND</h1>")
        (content-type "text/html")
        (status 404))))
```

Apps

```
(defn hello [request]
  (if (= "/hello" (request :uri))
    { :status 200
      :headers {"Content-Type" "text/html"}
      :body "<h1>OMG HI!</h1>" }

    (-> (response "<h1>NOT FOUND</h1>")
        (content-type "text/html")
        (status 404))))
```

Middleware Handlers

```
(defn auth-handler [request]
  (if (authorized? request)
    (handler request)
    (-> (response "Access Denied")
        (status 403))))
```

Middleware

```
(defn wrap-auth [handler]
  (fn [request]
    (if (authorized? request)
      (handler request)
      (-> (response "Access Denied")
          (status 403)))))
```

The Gauntlet

```
(defn handler [request]
  (-> (response "<h1>OMG HI!</h1>")
      (content-type "text/html")
      (status 200)))

(def app
  (-> handler
      wrap-auth
      (wrap-log :body)
      wrap-params))
```

The Request (Again)

```
{ :server-port        80
  :server-name        "example.com"
  :remote-addr        "127.0.0.1"
  :uri                "/help"
  :query-string       "search=ring"
  :scheme             "http"
  :request-method     :get
  :headers            {"accepts" "..."}
  :body               nil ; an InputStream
  :content-type       nil
  :content-length     nil
  :character-encoding nil }
```

Params Middleware

- Parses query string, body parameters
- Adds three keys to the request

```
{ :query-params {"search" "ring"}
  :form-params  {}
  :params       {"search" "ring"} }
```

Standard Middleware

- file
- static
- content-type
- file-info
- cookies
- session
- flash
- params
- keyword-params
- multipart-params
- nested params
- lint
- reload
- stacktrace

3rd-Party Middleware

- partial-content
- permacookie
- session-expiry
- session stores: mongodb, redis, riak
- basic-auth
- gzip
- json-params
- etag
- hoptoad
- upload-progress

Adapters

- Apache
- Jetty
- Plenty of others available …

Routing

- For simple apps, build a routing table with regexps and handlers.
- Add-on libraries provide routing configuration macros.

Processing Requests

```clojure
(defn login-post [request]
  (let [user (validate-user (:userid request)
                            (:password request))]
    (if user
      (render-template "login_successful" request user)
      (render-template "login_failed" request))))
```

Processing Requests

```clojure
(defn login-post [request]
  (let [user (validate-user (:userid request)
                            (:password request))]
    (if-not user (error :unauthorized)
    (render-template "login_successful" request user))))
```

Templating

Templating: clj-html

```clojure
(defn login-box
  []
  (if (is-logged-in)
    (do [:span {:class "login-text"}
      (get-user) " - "
          [:a {:href (get-logout-url "/")}
              "sign out"]])
    [:span {:class "login-text"}
      [:a {:href (get-login-url "/")} "sign in"]]))
```

```clojure
(defn render
  "The base layout for all pages"
  [body]
  (html
    (doctype :html4)
    [:head (include-css "/stylesheets/style.css")]
    [:body
      [:div {:class "container"}
        [:div {:id "login"} (login-box)]
        [:div {:id "content"} body]]]))
```

```clojure
(defn index
  [request]
  (render "Howdy!"))
```

Templating: Enlive

```
<!-- file index.html -->
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <link rel="stylesheet"
          type="text/css"
          href="/stylesheets/style.css"/>
  </head>
  <body>
    <div class="container">
      <div id="content">body text</div>
    </div>
  </body>
</html>
```

```
(deftemplate index "templates/index.html" [body-text]
  [:div.container] (prepend (login-box))
  [:div#content] (content body-text))
```

```html
<!-- file snippets.html -->
<div id="login">
  <span class="login-text">Login form or logout link</span>
</div>
```

```clojure
(defsnippet login-box "templates/snippets.html" [:#login] []
  [:div#login :span.login-text]
      (content
        (html-snippet
          (if (is-logged-in)
              (str (get-user)
                   " - "
                   (link-to "sign out" (get-logout-url "/")))
              (link-to "sign in" (get-login-url "/"))))))
```

```
(index "Howdy!")
```

Templating: Fleet

```
<!-- file templates/index.html.fleet -->
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <link type="text/css"
          href="/stylesheets/style.css"
          rel="stylesheet"/>
  </head>
  <body>
    <div class="container">
      <(login-box)>
      <div id="content">
        <(str data)>
      </div>
    </div>
  </body>
</html>
```

```
<!-- file templates/login-box.html.fleet -->
<div id="login">
  <span class="login-text">
    <(if (is-logged-in) ">
       <(get-user)> - <(link-to "sign out"
                                (get-logout-url "/"))>
       <" (link-to "sign in" (get-login-url "/")))>
  </span>
</div>
```

```
(fleet-ns view "templates")

(view/index "Howdy!")
```