Tape Framework

Less random stuff in random places

clyfe

December 21, 2020

Tape Modules

- (Ported to cljc from Duct-Framework)
- The configuration is initiated twice:
  - first, into an intermediate configuration,
  - which in turn: is initiated into the system.

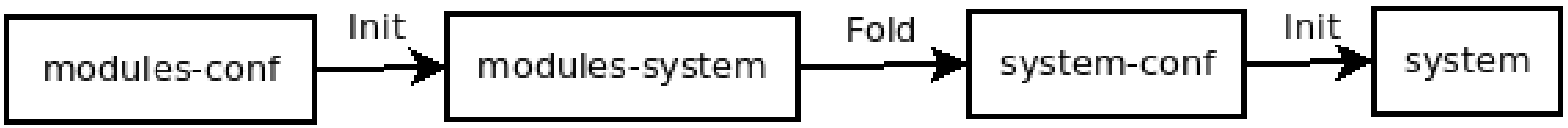

Figure: Modules

Components: two types

- Module components (type 1):
  - initialize into functions that
  - merge data into the system config map.
- Module components are mostly:
  - Controller modules: add Re-Frame function components.
  - View modules: add Reagent view function components.
- System components (type 2): whatever else, but mostly:
  - Re-Frame registerable functions and
  - Reagent view functions.

Init process

- Read module config edn.
- Prep module config into system config.
- Init system.

```
{:tape.mvc.controller/module nil
 :tape.mvc.view/module nil
 ...
 :tape.toasts.controller/module nil}
```

Figure: resources/myapp/config.edn

```
(def module-conf (m/read-config "myapp/config.edn"))
(def system-conf (m/prep-config module-conf))
(def system (ig/init system-conf))
```

Figure: src/myapp/core.clj

Re-Frame Globals

- Re-Frame chose globals for a simpler API.

```
(ns my.app.p.controller)

(rf/reg-event-db ::index
  (fn [db _]
    {::people (...)}))
```

Figure: Re-Frame plain

```
(ns my.app.people.view)

(defn index []
  (for [person people]
    [:p person]))
```

Figure: Reagent plain

Use Integrant

- Use Integrant to do away with globals.
- (Re-)**Frame** registration done by a separate Integrant component.

```
(ns my.app.p.controller)

(defmethod ig/init-key
  ::index [_ _]
  (fn [db _]
    {::people (...)
     ::filtered false}))
```

Figure: Re-Frame Integrant

```
(ns my.app.people.view)

(defmethod ig/init-key
  ::index [_ _]
  (fn []
    (for [person people]
      [:p person])))
```

Figure: Reagent Integrant

Automate use Integrant

- Start with plain functions.
- Leverage Integrant keys inheritance.

```
(ns my.app.p.controller)

(defn index [db _]
  {::people (...)
   ::filtered false})

(derive ::index
        :tape/const)
```

Figure: Re-Frame Integrant

```
(ns my.app.people.view)

(defn index []
  (for [person people]
    [:p person]))

(derive ::index
        :tape/const)
```

Figure: Reagent plain

Use Modules

- Modules are buckets of handlers, subscritptions, view functions.

```
(defmethod ig/init-key
  ::module [_ _]
  (fn [config]
    (merge config
      {::index index})))
```

Figure: Controller module

```
(defmethod ig/init-key
  ::module [_ _]
  (fn [config]
    (merge config
      {::index index})))
```

Figure: View module

Automate use Modules

- Have to annotate functions.
- **defmodule**:
  - inspects the namespace,
  - expands into modules above.

```
(c/defmodule)
```

Figure: Controller module

```
(v/defmodule my.app.p.controller)
```

Figure: View module

```
(defmethod ig/init-key :tape/const [_ v] v)
```

Figure: Note

Leverage Intuitions

- **MVC** is well understood, let's use it in naming.
  - Controller: Re-Frame stuff.
  - View: Reagent stuff.
  - Model: whatever.
- An **App** is is multiple MVC triples.

Leverage Metadata

- Annotate functions, conveys what they **are**.
- Automate module definition via **defmodule**.

```
(ns my.app.people.controller)

(defn ^::c/event-db show [db _]
  {::person (...)})

(defn ^::c/sub person [db _]
  (::person db))


(c/defmodule)
```

Figure: Controller module

```
(ns my.app.people.view)

(defn ^::v/view show []
  (let [person
        @(v/subscribe
           people.c/person)]
    [:p person]))

(v/defmodule
  my.app.people.controller)
```

Figure: View module

Module discovery

- Requiring each view and controller module is tedious.
- "Module discovery" finds them by name pattern.

```
(mvc/require-modules "src/myname/myapp/app")
;; (require '[myname.myapp.app.this.view])
;; (require '[myname.myapp.app.that.controller])

(let [{:keys [modules routes]}
      (mvc/modules-discovery "src/myname/myapp/app")]
  ...add modules and routes to config map...)
;; modules is a map:
;; {:myname.myapp.app.this.view/module nil
;;  :myname.myapp.app.that.controller/module nil}
;; routes is a vector:
;; - of routes in each controller module.
```

Consistent Naming

- Force: event names to be namespaced (by real, existing namespaces).
- Force: handlers names to match event names.
- If a view is rendered as a result of an event handler executing, their names should match.

```
:my.app.people.controller/index  ;;  event
my.app.people.controller/index   ;;  handler
my.app.people.view/index         ;;  view
```

Make calls navigable

- Event dispatches bear some analogy to function calls.
- They should be navigable as well ("jump to definition").
- v/subscribe & v/dispatch are macros:
  - use handler & subscription symbols,
  - macroexpand to normal Re-Frame.

```
(v/dispatch
  [people.c/index])

(rf/dispatch
  [::people.c/index])
```

Figure: Dispatch

```
(v/subscribe
  [people.c/person])

(rf/subscribe
  [::people.c/person])
```

Figure: Subscribe

Links

https://github.com/tape-framework