Tape Framework

Less random stuff in random places

clyfe

January 1, 2021

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.

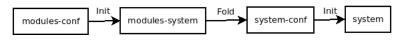


Figure: Modules

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/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.

Figure: Re-Frame plain

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)
                              (ns my app people view)
(defmethod ig/init-key
                              (defmethod ig/init-key
                                ::index [ ]
  ::index [ ]
  (fn [db ]
                                (fn []
    {::people (...)
                                  (for [person people]
     :: filtered false }))
                                    [:p person])))
```

Figure: Re-Frame Integrant

Figure: Reagent Integrant

Automate use Integrant

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

```
(ns my.app.p.controller)
                              (ns my app people view)
(defnindex [db]
                              (defnindex []
  {::people (...)
                                (for [person people]
   :: filtered false })
                                   [:p person]))
(derive ::index
                              (derive ::index
        :tape/const)
                                       :tape/const)
```

Figure: Reagent plain

Figure: Re-Frame Integrant

Use Modules

• Modules are buckets of handlers, subscriptions, view functions.

```
      (defmethod ig/init-key
      (defmethod ig/init-key

      :: module [__]
      :: module [__]

      (fn [config]
      (fn [config]

      (merge config
      (merge config

      {::index index})))
      {::index index})))
```

Figure: Controller module

Figure: View module

Automate use Modules

- Have to annotate functions with metadata.
- mvc/defm:
 - inspects the namespace,
 - expands into modules above.

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

Figure: Note

Gist: the right thing is sugar for the left thing

```
(ns my.app.p.controller)
(ns my.app.p.controller)
(defmethod ig/init-key
                             (defn index
  ::index [ ]
                               {::mvc/reg ::mvc/event—
 (fn [db ]
                               [db ]
    {::people (...)}))
                               {::people (...)})
(derive ::index
        ::mvc/event—db)
(defmethod ig/init-key
  (fn [config]
   (merge config
      {::index nil })))
                             (mvc/defm :: module)
```

Leverage Intuitions

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

Leverage Metadata

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

```
( ns my.app.people.controller)
(defn show
   {::mvc/reg ::mvc/event-db}
   [db _] {::person (...)})
(defn person
   {::mvc/reg ::mvc/sub}
   [db _] (::person db))
(mvc/defm ::module)
```

Figure: Controller module

Figure: View module

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

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

(def config ;; modules system config map
  (merge
    (mvc/modules - map "src/myapp/app")
    (module/read - config "myapp/config.edn")))
```

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").
- tools/subscribe & tools/dispatch are macros:
 - use handler & subscription symbols,
 - macroexpand to normal Re-Frame.

```
(mvc/dispatch
                               (mvc/subscribe
  [people.c/index])
                                 [people.c/person])
                                  becomes
   becomes
(rf/dispatch
                               (rf/subscribe
 [:: people.c/index])
                                [:: people.c/person])
```

Figure: Dispatch

Figure: Subscribe

Links

https://github.com/tape-framework

