Clojure Basic Training - Module 4

Vars, bindings and namespaces

Clojure is dynamic

- You can connect and alter running programs
- But Clojure is not interpreted!
- Clojure is compiled (to JVM bytecode)
- “lein repl :connect host:port” lets you connect to any running nREPL out there!
- **Vars** is what enable Clojure dynamicity

What are Vars?

```
user=> (def x)
#'user/x
user=> (type (var x))
clojure.lang.Var
user=> (type user/x)
clojure.lang.Var$Unbound
```

- An indirection mechanism to refer to a value
- Var = name + 1 or more values + logic to return the value
- A Var is “bound” to a value or “unbound”
- A Var can be “re-bound” to another value
- Can be created with (def)

Where are Vars?

- (def x) creates a var
- (defn f []) also creates a var
- Many other variants: definline defmacro defmethod defmulti defonce defstruct
- When reloading a namespace, all vars contained in it are re-bound

Root binding

```
user=> (def x 1)
user=> (type (var x))
clojure.lang.Var
user=> (type x)
java.lang.Long
user=> x
1
```

- A var can be given a root value at creation time
- The root value can be changed anytime
- The root value is what you usually get and use

Namespaces

- Once a Var is created it goes into “storage”
- Everything in Clojure must be defined in storage
- This storage mechanism is called a **namespace**
- The REPL puts you automatically in “user>”
- Namespaces can be created and searched

Namespaces and files

Folder nesting gives ns name

```
$> foo/bar.clj
(ns foo.bar)
(def baz)
#'foo.bar/baz
```

Form declaration ends up in corresponding ns

- Clojure programs can be split across different files
- Forms in a file are attached to a default ns
- Default ns depends on file name and nested folders
- But (ns) declaration at the top is mandatory!

Require recap for the lab

```
1. (ns my-ns (:require [clojure.string :refer [blank?]]))
2. (ns my-ns (:require [clojure.string :as s]))
3. (ns my-ns (:require [clojure.string :refer :all]))
4. (ns my-ns (:refer-clojure :exclude [print]))
```

1. (blank?) is the only available without ns
2. clojure.string now available as “s/”
3. all definitions in clojure.string are available (danger!)
4. I want to define my own print

References

- Daniel Solano Gomez, “How Clojure Works”, Clojure/West 2014
- “Clojure Programming”, O’Reilly