Arithmetic on Hash Maps

The Team of Fu

September 19, 2013

Contents

1 Introduction

Remark This is a literate program. [1] Source code *and* PDF documentation spring from the same, plain-text source files.

[1] See `http://en.wikipedia.org/wiki/Literate_programming`.

2 Tangle to Leiningen

```
(defproject ex1 "0.1.0-SNAPSHOT"
  :description "Project Fortune's Excel Processor"
  :url "http://example.com/TODO"
  :license {:name "TODO"
            :url "TODO"}
  :dependencies [[org.clojure/clojure  "1.5.1"]
                 [org.clojure/data.zip "0.1.1"]
                 [dk.ative/docjure     "1.6.0"]
                ]
  :repl-options {:init-ns ex1.core})
```

2.0.1 The Namespace

```
(ns ex1.core
  (:use [clojure.data.zip.xml :only (attr text xml->)]
        [dk.ative.docjure.spreadsheet] )
  (:require [clojure.xml :as xml]
            [clojure.zip :as zip]))
```

2.0.2 Data Instances

```
(def xml (xml/parse "myfile.xml"))
(def zippered (zip/xml-zip xml))
```

2.0.3 A Test Excel Spreadsheet

```
(let [wb (create-workbook "Price List"
                          [["Name"       "Price"]
                           ["Foo Widget" 100]
                           ["Bar Widget" 200]])
      sheet (select-sheet "Price List" wb)
      header-row (first (row-seq sheet))]
  (do
    (set-row-style!
      header-row
      (create-cell-style! wb
        {:background :yellow,
         :font       {:bold true}}))
    (save-workbook! "spreadsheet.xlsx" wb)))
```

2.1 Core Unit-Test File

Unit-testing files go in a subdirectory named `./ex1/test/ex1`. Again, the directory-naming convention enables valuable shortcuts from Leiningen.

As with the core source files, we include the built-in and downloaded libraries, but also the `test framework` and the `core` namespace, itself, so we can test the functions in the core.

```
(ns ex1.core-test
  (:use [clojure.data.zip.xml :only (attr text xml->)]
        [dk.ative.docjure.spreadsheet]
  )
  (:require [clojure.xml :as xml]
            [clojure.zip :as zip]
            [clojure.test :refer :all]
            [ex1.core :refer :all]))
```

We now test that the zippered XML file can be accessed by the *zipper* operators. The main operator of interest is `xml->`, which acts a lot like Clojure's *fluent-style* [2] *threading* operator `->`.[3] It takes its first argument, a

[2] `http://en.wikipedia.org/wiki/Fluent_interface`

[3] `http://clojuredocs.org/clojure_core/clojure.core/-%3E`

zippered XML file in this case, and then a sequence of functions to apply. For instance, the following XML file, when subjected to the functions `:track`, `:name`, and `text`, should produce `'("Track one" "Track two")`

```
<songs>
  <track id="t1"><name>Track one</name></track>
  <ignore>pugh!</ignore>
  <track id="t2"><name>Track two</name></track>
</songs>
```

Likewise, we can dig into the attributes with natural accessor functions [4]#+name: docjure-test-namespace

```
(deftest xml-zipper-test
  (testing "xml and zip on a trivial file."
    (are [a b] (= a b)
      (xml-> zippered :track :name text) '("Track one" "Track two")
      (xml-> zippered :track (attr :id)) '("t1" "t2"))))
```

Next, we ensure that we can faithfully read back the workbook we created *via* `docjure`. Here, we use Clojure's `thread-last` macro to achieve fluent style:

```
(deftest docjure-test
  (testing "docjure read"
    (is (=

      (->> (load-workbook "spreadsheet.xlsx")
           (select-sheet "Price List")
           (select-columns {:A :name, :B :price}))

      [{:name "Name"      , :price "Price"}, ; don't forget header row
       {:name "Foo Widget", :price 100.0  },
       {:name "Bar Widget", :price 200.0  }]

      ))))
```

[4] Clojure treats colon-prefixed keywords as functions that fetch the corresponding values from hashmaps, rather like the dot operator in Java or JavaScript; Clojure also treats hashmaps as functions of their keywords: the result of the function call `({:a 1} :a)` is the same as the result of the function call `(:a {:a 1})`

3 A REPL-based Solution

To run the REPL for interactive programming and testing in org-mode, take the following steps:

1. Set up emacs and nRepl (TODO: explain; automate)
2. Edit your init.el file as follows (TODO: details)
3. Start nRepl while visiting the actual |project-clj| file.
4. Run code in the org-mode buffer with `C-c C-c`; results of evaluation are placed right in the buffer for inspection; they are not copied out to the PDF file.

```
[(xml-> zippered :track :name text)        ; ("Track one" "Track two")
 (xml-> zippered :track (attr :id))]       ; ("t1" "t2")
```

```
(->> (load-workbook "spreadsheet.xlsx")
     (select-sheet "Price List")
     (select-columns {:A :name, :B :price}))
```

```
(run-all-tests)
```

4 References

5 Conclusion

Fu is Fortune.