AboutFishbowl

January 21, 2015

1 Fishbowl: an enhanced interactive Shell for Common Lisp

This document is a short presentation of *Fishbowl* which is, technically-speaking, a Common Lisp implementation of an IPython kernel. So what does this means in practice ? Let's see.

1.1 A (somewhat poor man's) distributed Lisp REPL

Basically IPython is a better shell (understand REPL) for the Python programming language. So what the h.ck is the relationship with Lisp ? Well, the IPython architecture allows distributed (unsecure !) interactions between :

- clients – called *frontends* – that manage the user-part of the interactions : reading expressions, printing results and such, and

- servers – called *kernels* – that actually perform the computations.

So in an IPython REPL, the frontends play the R and P parts, while the kernels play the E part. Fishbowl is thus the Eval part while the existing IPython frontends can be used for Reading and Printing. Basic interactions would look like the following :

In [1]: (+ 2 12)

Out[1]: 14

Things happen like these :

- first, the frontend reads the Lisp expression (+ 2 12)

- this expression is sent, through the network (ZMQ sockets to be precise), to the Fishbowl kernel, which performs the corresponding computation (thanks to the `eval` function of course).

- the resuling value 14 is then sent back to the frontend for printing, and that's it.

In case the expression yields side effects, such as writing a file, these are actually performed by Fishbowl so one should take care servicing kernels only on private networks ! The standard output and error streams are captured and thus visible from the frontend side.

In [2]: (format t "I compute (+ 2 12), yielding ~A" (+ 2 12))

I compute (+ 2 12), yielding 14

Out[2]: NIL

The second expression, namely `In [2]`, yields the `NIL` value, namely `Out [2]`, but the formatted string written on the `*standard-output*` stream is also printed.

Errors are also printed,as the following (counter-)example shows :

```
In [3]: (/ 2 0)
```

```
DIVISION-BY-ZERO:
  #<DIVISION-BY-ZERO {1007988493}>
```

```
Out[3]: NIL
```

We remark that, at least in the current version, Fishbowl does *not* support interactions with the Lisp debugger.

This is a good moment to stay something important :

Fishbowl is **not** intended as a replacement for SlimE or SlimV.

Indeed, this would imply replacing Emacs of Vim for starting with . . .

At the very best, Fishbowl (together with the IPython frontend) could be seen as a replacement for the basic REPL of most Common Lisp implementations (as for now without the debugger). An interface with Swank (the "kernel" part of SlimE/V) would be a strong improvement here, but that is only for now a wish (contributors welcome !).

Some features might already prove useful. First and foremost, the distributed architecture of IPython/Fishbowl allows to connect multiple frontends to multiple kernels, as well as exchanging Data. Since IPython kernels exist for a growing number of programming languages (Python, Ruby, Julia, Ocaml, Haskell, etc.), this opens many possibilities such as writing interactive distributed applications developped in a multi-languages environment (which appeals much more to me than the *everything-in-javascript* trend of the moment).

But a question remains: is *Fishbowl* bringing something to the Lisp table ? Obviously, a not-really-better REPL does not. The main reason why *Fishbowl* was written in the first place is for supporting IPython *notebooks*.

1.2 A Lisp environment for interactive documents

So what is a notebook ? Well, that's easy, this is what you are reading now !

It is a document mixing :

- *Markdown*-formatted **text** (optionally including *Latex/Mathjax* formulas)

- **Computations** described in various language, Common Lisp as far as Fishbowl is concerned.

For example, *Leonardo* may write a fantastically clever algorithm for an important computation :

```
In [4]: (defun fibonacci (n)
          (if (<= n 1)
              1
              (+ (fibonacci (- n 2)) (fibonacci (- n 1)))))
```

```
Out[4]: FIBONACCI
```

And then, Leonardo can answer one of its most desired question :

```
In [5]: (fibonacci 10)
```

```
Out[5]: 89
```

```
In [5]: (loop for k below 10 collect (fibonacci k))
```

```
Out[5]: (1 1 2 3 5 8 13 21 34 55)
```

and then, comment about such computations, such that making conjectures or wonderings. . .

What about $\lim_{k \to +\infty} \frac{F_{k+1}}{F_k}$ with F_k the k-th term of the Fibonacci serie ?

Well, let's check this . . .

```
In [7]: (loop for k from 2 below 20 collect (/ (fibonacci (1+ k)) (fibonacci k)))
```

```
Out[7]: (3/2 5/3 8/5 13/8 21/13 34/21 55/34 89/55 144/89 233/144 377/233 610/377 987/610 1597/987 2584/
```

Oops ... don't want the exact rational values but approximations ...

```
In [6]: (loop
          for k from 2 below 20
          collect (float (/ (fibonacci (1+ k)) (fibonacci k))))
```

```
Out[6]: (1.5 1.6666666 1.6 1.625 1.6153846 1.6190476 1.617647 1.6181818 1.6179775
          1.6180556 1.6180258 1.6180371 1.6180328 1.6180345 1.6180338 1.618034 1.618034
          1.618034)
```

What is this value 1.618034 ? ... seems not far from ... $\frac{1+\sqrt{5}}{2}$

```
In [2]: (/ (+ 1 (sqrt 5))
            2)
```

```
Out[2]: 1.618034
```

Don't you think this number looks shiny ?

This (probably unsuccessful) half-joke at least summarizes the way I am using notebooks in my own teaching and research work.

At the technical level, the IPython notebook server is simply a frontend developped as a complete web application that can connect to any kernel, among which of course Fishbowl.

1.3 Rich display

The notebook provide rich display for textual and graphical data. The package `fishbowl-user` provides a few functions for producing displayable data in Lisp.

By default, the Lisp backend works in the `fishbowl-user` package, which we can check right away:

```
In [3]: *package*
```

```
Out[3]: #<PACKAGE "FISHBOWL-USER">
```

1.3.1 Customized plain text

By default, the data produced by the Lisp kernel is formatted as in (`format t "~S" <data>`), i.e. roughly the same way the Lisp REPL works. The produced Lisp string is then encoded as a JSON string and sent back to the frontend(s).

Let's see what happens with a string.

```
In [10]: "this is a string with a
          new line"
```

```
Out[10]: "this is a string with a
          new line"
```

Suppose now we would like to define a particular kind of string where new lines are replaced (in the shown output) by JSon new lines, i.e. the string `"\n"`.

The simplest way to do so is to create a CLOS class to encoded our special strings.

```
In [1]: (defclass custom-string ()
          ((content :initarg :content :reader cstr-content)))
```

```
Out[1]: #<STANDARD-CLASS CUSTOM-STRING>
```

```
In [2]: (defun custom-string (str)
          (make-instance 'custom-string :content str))
```

Out[2]: CUSTOM-STRING

Now, we have to specialize a generic method **render-plain** to produce our customized strings.

```
In [3]: (defmethod render-plain ((str-obj custom-string))
          (let ((ncontent (make-array (length (cstr-content str-obj))
                                       :element-type 'character :fill-pointer 0 :adjustable t)))
              (loop for char across (cstr-content str-obj)
                    for index from 0
                    do (cond ((char= char #\Newline)
                                (vector-push-extend #\\ ncontent)
                                (vector-push-extend #\n ncontent))
                             (t (vector-push-extend char ncontent))))
              (format nil "\"~A\"" ncontent)))
```

Out[3]: #<STANDARD-METHOD RENDER-PLAIN (CUSTOM-STRING) {1007368573}>

Let's try our customized strings ...

```
In [4]: (custom-string "this is a customized string with a
         new line")
```

Out[4]: "this is a customized string with a\n new line"

1.3.2 Programmatically-generated Latex

A nice feature of the notebook is that one may use Latex to describe mathematical formulas. This can be done in the markdown cells, by directly writing the formulas.

For example: `$\frac{17}{48}$` will give: $\frac{17}{48}$

It is also possible to generate latex on the kernel side using Lisp code. The **latex** function simplifies the process :

```
In [12]: (latex "$\\frac{17}{48}$")
```

Out[12]:

$$\frac{17}{48}$$

Note that the backslash must be escaped in the Lisp string. Thanks to this **latex** function, we can generate latex content programmatically, as in the following example :

```
In [13]: (defun show-fraction (num den)
           (latex (format nil "$\\frac{~A}{~A}$" num den)))
```

```
REDEFINITION-WITH-DEFUN:
  #<SB-KERNEL:REDEFINITION-WITH-DEFUN {100794EF63}>
```

Out[13]: NIL

```
In [15]: (show-fraction 17 48)
```

Out[15]:

$$\frac{17}{48}$$

4

1.3.3 Raw HTML content

It is of course also possible to generate raw HTML content directly from Lisp. This is not recommended however in the case the intent is to generate beautiful printouts of the notebook. But the HTML generation can be useful at times.

```
In [6]: (html "<h3 style=\"color:red\">Important</h3>
             <p style=\"color:blue\">Fishbowl is <strong style=\"color:green\">nice</strong>.</p>")

Out[6]: #<HTML-TEXT {100685FF43}>
```

You can *of course* use your favorite HTML framework to generate your content. Here is an example.

```
In [3]: (ql:quickload "cl-markup")

To load "cl-markup":
  Load 1 ASDF system:
    cl-markup

; Loading "cl-markup"

Out[3]: ("cl-markup")

In [7]: (html (markup:markup
              (:table :border 0 :cellpadding 4
                      (loop for i below 25 by 5
                            collect (markup:markup
                                        (:tr :align "right"
                                            (loop for j from i below (+ i 5)
                                                  collect (markup:markup
                                                              (:td :bgcolor
                                                                  (if (oddp j) "pink" "green")
                                                                  (format nil "~@R" (1+ j)))))))))))

Out[7]: #<HTML-TEXT {1006370D73}>
```

1.3.4 Vector images as SVG

```
In [7]: (svg-from-file "profile/fishbowl-small.svg")

Out[7]:
```


1.3.5 Bitmap images as PNG

```
In []:
```

1.4 To be continued . . .

In [ ]: