literate lisp

a literate programming tool to write common lisp codes in org mode.

Jingtao Xu

December 8, 2018

Contents

1 Introduction

This is a common lisp project to show a way how to use literate programming in common lisp.

literate programming examples show the reason why use org mode, and there are also another lisp project papyrus to do the similiar thing but it use markdown file format.

By using common lisp package literate-lisp , emacs org mode and elisp library polymode, literate programming can be done perfectly

in one org file containing both documentation and source codes, and this org file works well with SLIME.

2 How to do it?

In org mode, the comment line start with character # (see org manual), and the lisp codes are surrounded by lines between #+begin_src lisp and #+end_src (see org manual).

```
#+BEGIN_SRC lisp :tangle no
(format t "this is a test.~%")
#+END_SRC
```

So to let lisp can read an org file directlly, all lines out of surrounding by #+begin_src lisp and #+end_src should be ignored, and even codes surrounding by them should be ignored if the options in a code block request such behaviour.

Here is a trick, a new lisp reader syntax for "# "(Sharpsign Whitespace) is defined to make lisp reader enter into org mode syntax, then ignore all lines after that until it meet #+BEGIN_SRC lisp.

When #+begign_src lisp is met, all org options for this code block will be read and it give us a chance to switch back to normal lisp reader or not.

And if it switch back to normal lisp reader, the end line #+END_SRC should be catched, so the lisp reader syntax for "#+"(Sharpsign Plus) is redefined to determine if it is #+END_SRC, if it is, then lisp reader will switch back to org mode syntax, if it is not, lisp reader will continue to read subsequent stream as like the original lisp reader.

This workflow restricts the org file starting with a comment character and a space character("# "), but it should not be a problem but indeed a convenient way for us to specify some local variables, for example I often put them in the first line of an org file:

```
# -*- encoding:utf-8 Mode: POLY-ORG;  -*- ---
```

Which make emacs open file with utf-8 encoding and poly-org-mode.

3 Implementation

3.1 Preparation

Firstly a new lisp package for this library is defined. Package named-readtables is used here to simplify the definition of reader syntax.

```
(in-package :common-lisp-user)
(defpackage :literate-lisp
  (:use :cl :named-readtables)
  (:documentation "a literate programming tool to write common lisp codes in org file."
      ↪ ))
(pushnew :literate-lisp *features*)
(in-package :literate-lisp)
```

a debug variable is used to switch on/off the log messages for this library

```
(defvar debug-literate-lisp-p nil)
(declaim (type boolean debug-literate-lisp-p))
```

3.2 function to handle reader syntax for "# "(# + Space)

There are many different lisp codes are written in one org file, some for function implementation, some for demo, and some for test, so an org code block option is defined to decide to read them or not. For example, if one lisp code block is used for demo, then it should be ignored when loading this org file. And if one lisp code block is used for test, it should only be loaded in a test environment.

a new org code block option `tangle` is defined after `#+BEGIN_SRC lisp`,and it has three meanings:

- yes
  It means that current code block should be read normally, it is the default mode when the option `tangle` is not provided.
- no
  It means that current code block should be ignored by lisp reader.
- test
  It means that current code block should be read only when feature `test` exist.

```
(defun tangle-p (feature)
  (case feature
    ((nil :yes) t)
    (:no nil)
    (:test (find :test *features* :test #'eq))))
```

Let's implement a function to read options after `#+BEGIN_SRC`, and convert every key and value to a lisp keyword(Test in here: 4.2.1).

```
(defun read-org-code-block-options (string begin-position-of-options)
  (with-input-from-string (stream string :start begin-position-of-options)
    (let ((*readtable* (copy-readtable nil))
          (*package* (find-package :keyword))
```

```
          (*read-suppress* nil))
        (loop for elem = (read stream nil)
                      while elem
                      collect elem))))
```

Now it's time to implement the new reader function for syntax "#"(# + Space).

```
(defun tangle-number-sign+space (stream a b)
  (declare (ignore a b))
  (loop for line = (read-line stream nil nil) then (read-line stream nil nil)
        until (null line)
        for start1 = (loop for c of-type character across line
                           for i of-type fixnum from 0
                           until (not (find c '(#\Tab #\Space)))
                           finally (return i))
        do (when debug-literate-lisp-p
             (format t "ignore line ~a~%" line))
        until (when (equalp start1 (search #1="#+BEGIN_SRC lisp" line))
                    (let* ((options (read-org-code-block-options line (+ start1 (length
                        ↪ #1#)))))
                      (tangle-p (getf options :tangle :yes)))))
  (values))
```

3.3 an implementation of original feature test.

This code block is referenced from the sbcl source codes with some minor modifications.

```
;;; If X is a symbol, see whether it is present in *FEATURES*. Also
;;; handle arbitrary combinations of atoms using NOT, AND, OR.
(defun featurep (x)
  (typecase x
    (cons
     (case (car x)
       ((:not not)
        (cond
          ((cddr x)
           (error "too many subexpressions in feature expression: ~S" x))
          ((null (cdr x))
           (error "too few subexpressions in feature expression: ~S" x))
          (t (not (featurep (cadr x))))))
       ((:and and) (every #'featurep (cdr x)))
       ((:or or) (some #'featurep (cdr x)))
       (t
        (error "unknown operator in feature expression: ~S." x))))
    (symbol (not (null (member x *features* :test #'eq))))
    (t
      (error "invalid feature expression: ~S" x))))
```

3.4 function to handle reader syntax for "#+"

The mechanism to handle normal lisp syntax "#+" is also referenced from sbcl source codes.

```
(defun tangle-sharp-plus-minus (stream sub-char numarg)
  ;; 1. read into the feature as an keyword.
  (let ((feature (let ((*package* (find-package :keyword))
                       ;;(*reader-package* nil)
                       (*read-suppress* nil))
                   (read stream t nil t))))
    ;;       2.1 if the feature is '#+END_SRC', then switch back to org syntax.
    (when debug-literate-lisp-p
      (format t "found feature ~s,start read org part...~%" feature))
    (cond ((eq :END_SRC feature)
           (when debug-literate-lisp-p
             (format t "found #+END_SRC,start read org part...~%"))
           (funcall #'tangle-number-sign+space stream sub-char numarg))
          ;; 2.2 otherwise test the feature.
          ;;   2.2.1 If the feature exist, read the following object recursively
              ↪ normally.
          ((featurep feature)
           (read stream t nil t))
          ;;   2.2.1 if the feature doesn't exist, read the following object
              ↪ recursively and ignore it.
          (t
           (let ((*read-suppress* t))
             (read stream t nil t)
             (values))))))
```

3.5 Install the new reader syntax.

```
(defreadtable :org
  (:merge :standard)
  (:dispatch-macro-char #\# #\Space #'tangle-number-sign+space)
  (:dispatch-macro-char #\# #\+ #'tangle-sharp-plus-minus))
```

3.6 tangle an org file

A function is provided to build lisp file from an org file.
Argument `org-file` is the source org file.
Argument `keep-test-codes` is a boolean value to indicate whether test codes should be kept.

```
(defun tangle-org-file (org-file &key
                        (keep-test-codes nil)
                        (output-file (make-pathname :defaults org-file
                                                    :type "lisp")))
  (let ((*readtable* (ensure-readtable ':org))
        (*features* (if keep-test-codes
                        *features*
                        (remove :test *features* :test 'eq)))
        (*read-eval* nil)
        (*print-pretty* t))
    (with-open-file (input org-file)
        (with-open-file (output output-file :direction :output
                                :if-does-not-exist :create
                                :if-exists :supersede)
          (format output
```

```
                ";;; This file is automatically generated from file `~a.~a'.~%"
                (pathname-name org-file) (pathname-type org-file))
        (loop for object = (read input nil nil nil)
              until (null object)
              do (when debug-literate-lisp-p
                   (format t "read object ~s~%" object))
                 (write object :stream output)
                 (write-char #\Newline output))))))
```

So when a new version of ./tangle.lisp can be released from this file, the following code should be executed.

```
(tangle-org-file
 (format nil "~a/tangle.org"
         (asdf:component-pathname (asdf:find-system :literate-lisp))))
```

Listing 1: a demo code to tangle current org file.

This function is only an experimental implementaion yet.

3.7 make asdf handle org file correctly

Firstly a new source file class for org files is defined in asdf.

```
(in-package :asdf)
(defclass org (cl-source-file)
  ((type :initform "org")))
(eval-when (:compile-toplevel :load-toplevel :execute)
  (export '(org) :asdf))
```

So a new asdf source file type :org can be used to define an org file like this

```
(asdf:defsystem literate-demo
  :components ((:module demo :pathname "./"
                        :components ((:org "readme"))))
  :depends-on (:literate-lisp))
```

Listing 2: a demo code to show how to include org file in asdf.

And file readme.org will be treated as a lisp source file in asdf.

Then the new reader syntax for org file is intalled when asdf actions are performed to org files.

```
(in-package :literate-lisp)
(defmethod asdf:perform :around (o (c asdf:org))
  (let ((*readtable* (ensure-readtable ':org)))
    (when (find-package :swank)
      (editor-hints.named-readtables::%frob-swank-readtable-alist
       *package* *readtable*))
    (call-next-method)))
```

Then after this package is loaded, one org file can be supported to be loaded by asdf automatically.

4 Test cases

4.1 Preparation

Now it's time to validate some functions. The nst test framework is used.

```
(eval-when (:compile-toplevel :load-toplevel :execute)
  (unless (find-package :nst)
    (ql:quickload :nst)))
```

4.2 test groups

4.2.1 test for reading org code block options

```
(nst:def-test-group read-org-code-block-options ()
  (nst:def-test t1
      (:equal nil) (read-org-code-block-options "" 0))
  (nst:def-test t2
    (:equal '(:tangle :no)) (read-org-code-block-options " :tangle no  " 0))
  (nst:def-test t3
    (:equal '(:tangle :no)) (read-org-code-block-options " :tangle no" 0)))
```

4.3 run all tests in this library

this function is the entry point to run all tests and return true if all test cases pass.

```
(defun run-test ()
  (nst::run-package :literate-lisp)
  (multiple-value-bind (status checks passed error fail warn)
      (nst::result-summary (nst::all-tests-report))
    (format t "~&nst test status for literate-lisp:~a, checks:~d, passed:~d, error:~D,
      ↪ faile:~D,warn:~D"
          status checks passed error fail warn)
    (and (= fail 0) (= 0 error))))
```

4.4 run all tests in demo project

To run all tests in demo project `literate-demo`, please load it by yourself, it'll be loaded by travis ci which will load config file `./.travis.yml` automatically every time there is a new git change.