Slimv Tutorial - Part Three

Contents

This is the third part of the Slimv Tutorial series. In the first two parts we introduced Lisp source code editing concepts, basic REPL operations and the SLIME debugger. In this part we follow Marco Barringer's SLIME tutorial movie to its end.

This part of the tutorial assumes that you have followed Part One and Part Two and your REPL is loaded with all the definitions added there. So if you haven't yet visited the previous parts, please do so before continuing with this page.

Tracing

So far we have installed package **split-sequence**, so we can now write **morse-to-string**.
This is the initial version:

```
(defun morse-to-string (string)
  (with-output-to-string (character-stream)
    (loop
      for morse-char in (split-sequence:split-sequence #\Space string)
      do (write-char (morse-to-character morse-char) character-stream))))
```

Just for fun after compiling morse.lisp we test our new function with an extra space added at the end of the morse string:

```
MORSE> (string-to-morse "marco")
"-- .- .-. -.-. ---"
MORSE> (morse-to-string "-- .- .-. -.-. --- ")


Slimv.REPL.lisp                                                      75,0-1         Bot
The value NIL is not of type CHARACTER.
   [Condition of type TYPE-ERROR]

Restarts:
  0: [RETRY] Retry SLIME REPL evaluation request.
  1: [*ABORT] Return to SLIME's top level.
  2: [TERMINATE-THREAD] Terminate this thread (#<THREAD "repl-thread" RUNNING {B617D49}>)

Backtrace:
  0: (SB-IMPL::STRING-OUCH #<SB-IMPL::STRING-OUTPUT-STREAM {AE0A679}> NIL)
  1: (WRITE-CHAR NIL #<SB-IMPL::STRING-OUTPUT-STREAM {AE0A679}>)
  2: (MORSE-TO-STRING "-- .- .-. -.-. --- ")
  ...
Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^4,1^^^^^^^^^^^^Top
```

In the backtrace we can see that **nil** was passed to **write-char** (look at frame #1), but it's not immediately clear why. We can use the editor buffer for testing function calls "in place": let's quit the debugger, replace argument **string** in the **split-sequence:split-sequence** call and evaluate the containing form with **,e**:

```
(defun morse-to-string (string)
  (with-output-to-string (character-stream)
    (loop
      for morse-char in (split-sequence:split-sequence #\Space "-- .- .-. -.-. --- ")
      do (write-char (morse-to-character morse-char) character-stream))))
```

```
MORSE> (split-sequence:split-sequence #\Space "-- .- .-. -.-. --- ")
("--" ".-" ".-." "-.-." "---" "")
```

From the result it is now clear that the problem is caused by the last empty string. Let's fix it by removing empty subsequences in **split-sequence:split-sequence** (and don't forget to restore the original **string** argument):

```
(defun morse-to-string (string)
  (with-output-to-string (character-stream)
    (loop
      for morse-char in (split-sequence:split-sequence #\Space string
                                                       :remove-empty-subseqs t)
      do (write-char (morse-to-character morse-char) character-stream))))
```

After compiling and testing, our new function seems to work as intended (the characters are converted to upper case because our morse mapping contains capital letters only):

```
MORSE> (morse-to-string (string-to-morse "marco"))
"MARCO"
```

But how can we make sure that this is not just a strange coincidence? In order to answer this, we're going to trace the functions involved. Slimv toggles tracing by placing the cursor on a function name and pressing **,t** (or by selecting Debugging/Toggle-Trace from the Slimv menu). So let's switch on tracing for **string-to-morse** and **morse-to-string**.

With tracing enabled we re-evaluate our last test form:

```
MORSE>
;; Pressing ,t on symbol string-to-morse
STRING-TO-MORSE is now traced.
MORSE>
;; Pressing ,t on symbol morse-to-string
MORSE-TO-STRING is now traced.
MORSE> (morse-to-string (string-to-morse "marco"))
  0: (STRING-TO-MORSE "marco")
  0: STRING-TO-MORSE returned "-- .- .-. -.-. ---"
  0: (MORSE-TO-STRING "-- .- .-. -.-. ---")
  0: MORSE-TO-STRING returned "MARCO"
"MARCO"
MORSE>
;; Pressing ,T to untrace all
Untracing:
  morse::string-to-morse
  morse::morse-to-string
```

Now we also got the tracing output, showing what functions were called, the arguments passed to them, and their return values. We can be fairly confident that our functions work as we want them to.

Inspecting objects

The SLIME Inspector has been introduced in section Inspecting a package. This section describes some more advanced uses of the Inspector. Let's begin with evaulating the defective form **(morse-to-string (string-to-morse 42))**, dropping us into the debugger. If we open the frame(s) we can see the local variable bindings and the source location of the frame. In the example below frame #5 has one argument, presented as **SB-DEBUG::ARG-0**.

```
MORSE> (morse-to-string (string-to-morse 42))


Slimv.REPL.lisp                                                     145,0-1         Bot
The value 42 is not of type ARRAY.
   [Condition of type TYPE-ERROR]

Restarts:
  0: [RETRY] Retry SLIME REPL evaluation request.
  1: [*ABORT] Return to SLIME's top level.
  2: [TERMINATE-THREAD] Terminate this thread (#<THREAD "repl-thread" RUNNING {B618699}>)

Backtrace:
  0: (STRING-TO-MORSE 42)
  1: (SB-INT:SIMPLE-EVAL-IN-LEXENV (STRING-TO-MORSE 42) #<NULL-LEXENV>)
  2: (SB-INT:SIMPLE-EVAL-IN-LEXENV (MORSE-TO-STRING (STRING-TO-MORSE 42)) #<NULL-LEXENV>)
  3: (SWANK::EVAL-REGION "(morse-to-string (string-to-morse 42))\n")
  4: ((LAMBDA ()))
  5: (SWANK::TRACK-PACKAGE #<CLOSURE (LAMBDA #) {B1272AD}>)
      in "~/.vim/slime/swank.lisp" line 2258
    Locals:
      SB-DEBUG::ARG-0 = #<CLOSURE (LAMBDA ()) {AD4BE6D}>

  6: (SWANK::CALL-WITH-RETRY-RESTART "Retry SLIME REPL evaluation request." #<CLOSURE ...
  7: (SWANK::CALL-WITH-BUFFER-SYNTAX NIL #<CLOSURE (LAMBDA #) {B127225}>)
Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^15,1^^^^^^^^^^^^Top
```

If we press **Enter** on **in "~/.vim/slime/swank.lisp" line 2258** then Slimv opens the associated source file in a buffer and locates line 2258.

If we select Debugging/Inspect while the cursor is on the frame line, then a special "Inspect in Frame" function is executed, which inspects the result of evaluating an expression in the given frame. Choose it for examining the value of the function argument **SB-DEBUG::ARG-0** that has a kind of useless representation in the frame box (**#<CLOSURE (LAMBDA ()) {AD4BE6D}>**).

```
  4: ((LAMBDA ()))
  5: (SWANK::TRACK-PACKAGE #<CLOSURE (LAMBDA #) {B1272AD}>)
      in "~/.vim/slime/swank.lisp" line 2258
    Locals:
      SB-DEBUG::ARG-0 = #<CLOSURE (LAMBDA ()) {AD4BE6D}>

  6: (SWANK::CALL-WITH-RETRY-RESTART "Retry SLIME REPL evaluation request." #<CLOSURE ...
Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^15,1^^^^^^^^^^^^Top
Inspect in frame 5: sb-debug::arg-0
```

```
Inspecting #<FUNCTION {AD4BE6D}>
--------------------

[1] FUNCTION: #<FUNCTION (LAMBDA ()) {A20DB3D}>
Closed over values:
[2] 0: "(morse-to-string (string-to-morse 42)) ..


[<<]
Slimv.INSPECT [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^1,1^^^^^^^^^^^^Top
```

The Inspector output is much more descriptive, isn't it?

We can also use the "Inspect in Frame" facility for examining other variable bindings. Take for instance ***package***...:

```
16: ((FLET SWANK-BACKEND:CALL-WITH-DEBUGGER-HOOK) #<FUNCTION SWANK:SWANK-DEBUGGER-HOOK..
      in "~/.vim/slime/swank-sbcl.lisp" line 1014
    Locals:
      *DEBUGGER-HOOK* = :<NOT-AVAILABLE>
      SB-KERNEL:*HANDLER-CLUSTERS* = :<NOT-AVAILABLE>
      SWANK-BACKEND::FUN = #<CLOSURE (LAMBDA ()) {BCC93ED}>
      SWANK-BACKEND::HOOK = #<FUNCTION SWANK:SWANK-DEBUGGER-HOOK>

Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^30,1^^^^^^^^^^^^Top
Inspect in frame 16: *package*
```

```
Inspecting #<PACKAGE {B65E011}>
--------------------

[1] Name: "MORSE"
Nick names:
[2] Use list: COMMON-LISP
Used by list:
[3] 26 present symbols.
0 external symbols.
[4] 26 internal symbols.
[5] 978 inherited symbols.
0 shadowed symbols.

[<<]
Slimv.INSPECT [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^1,1^^^^^^^^^^^^Top
```

... or ***standard-output*** as another example:

```
16: ((FLET SWANK-BACKEND:CALL-WITH-DEBUGGER-HOOK) #<FUNCTION SWANK:SWANK-DEBUGGER-HOOK..
      in "~/.vim/slime/swank-sbcl.lisp" line 1014
    Locals:
      *DEBUGGER-HOOK* = :<NOT-AVAILABLE>

Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^30,1^^^^^^^^^^^^Top
Inspect in frame 16: *standard-output*
```

```
Inspecting #<SWANK-BACKEND::SLIME-OUTPUT-STREAM {B60A699}>
--------------------

[1] Class: #<STANDARD-CLASS SWANK-BACKEND::SLIME-OUTPUT-STREAM>
--------------------
<0>  Group slots by inheritance [ ]
<1>  Sort slots alphabetically  [X]

All Slots:
<2> [ ]
[2] BUFFER       =

Slimv.INSPECT [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^1,1^^^^^^^^^^^^Top
```

One can even inspect the result of a compound expression, like in the following example:

```
  1: (SB-INT:SIMPLE-EVAL-IN-LEXENV (STRING-TO-MORSE 42) #<NULL-LEXENV>)
      No source line information
    Locals:
      SB-DEBUG::ARG-0 = (STRING-TO-MORSE 42)
      SB-DEBUG::ARG-1 = #<NULL-LEXENV>

Slimv.SLDB [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^11,1^^^^^^^^^^^^Top
Inspect in frame 1: (make-hash-table)
```

```
Inspecting #<HASH-TABLE {AED4BD9}>
--------------------

[1] Count: 0
[2] Size: 16
[3] Test: EQL
[4] Rehash size: 1.5
[5] Rehash threshold: 1.0


[<<]
Slimv.INSPECT [RO]^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^1,1^^^^^^^^^^^^Top
```

Cross reference

The last thing Marco was talking about was the Cross Reference (XRef) facility. Let's assume we're required to add a new parameter to function **morse-to-character**. In this case we need to find out and update everyone who calls that function. We can use XRef/List Callers from the Slimv menu or press **,xl**. It tells us the list of function and file names (if resolved) where **morse-to-character** is called:

```
(defun morse-to-character (morse-string)
  (first (find morse-string *morse-mapping* :test #'string= :key #'cdr)))

morse.lisp                                                           52,8           53%
List callers: morse-to-character
```

```
MORSE>
;; Here follows the result of 'List callers: morse-to-character'
MORSE-TO-STRING - no source information

Slimv.REPL.lisp                                                      75,0-1         Bot
```

At the same time we can ask **morse-to-character**: who do you call?
Use XRef/List callees from the Slimv menu or press **,xe**:

```
(defun morse-to-character (morse-string)
  (first (find morse-string *morse-mapping* :test #'string= :key #'cdr)))

morse.lisp                                                           52,8           53%
List callees: morse-to-character
```

```
MORSE>
;; Here follows the result of 'List callees: morse-to-character'
FIND - no source information
SECOND - no source information
STRING= - no source information

Slimv.REPL.lisp                                                      75,0-1         Bot
```

If we list callees for **morse-to-string** then we get a slightly longer list, and - as expected - it contains our **morse-to-character** and the recently installed **split-sequence:split-sequence** functions:

```
(defun morse-to-string (string)
  (with-output-to-string (character-stream)
    (loop
      for morse-char in (split-sequence:split-sequence #\Space string)
      do (write-char (morse-to-character morse-char) character-stream))))

morse.lisp                                                           63,8           92%
List callees: morse-to-string
```

```
MORSE>
;; Here follows the result of 'List callees: morse-to-string'
(LAMBDA
    (SB-PCL::.ARG0. SB-INT:&MORE SB-PCL::.MORE-CONTEXT. SB-PCL::.MORE-COUNT.)) - no sourc
WRITE-CHAR - no source information
MORSE-TO-CHARACTER - no source information
GET-OUTPUT-STREAM-STRING - no source information
SPLIT-SEQUENCE:SPLIT-SEQUENCE - /home/kovisoft/.sbcl/site/split-sequence/split-sequence.l
MAKE-STRING-OUTPUT-STREAM - no source information

Slimv.REPL.lisp                                                      75,0-1         Bot
```

At that point the Slime introductory movie ends.

Once again, let me reuse Marco's words: *I hope you think* ***Slimv*** *is cool.* :)

Previous: Part Two

Written by Tamas Kovacs

Last updated on Aug 28, 2020