Common Lisp坑爹简介

Liutos
mat.liutos@gmail.com

Outline

1. Tool Chain
2. Program Structure
3. Data Types
4. Functions
5. Special Operators
6. Multiple Values Return
7. CLOS
8. Macro Facility
9. Readtable
10. Things Left

Development Tools

- Editor: GNU Emacs
- Emacs Plugins: SLIME, paredit-mode
- Common Lisp Implementations: SBCL, CCL, CLISP, ...
- Package Manager: Quicklisp
- Building System: ASDF

Form Categories

- Symbols as Forms

Example

```
boundp let* multiple-value-bind ++
```

Form Categories

- Symbols as Forms

Example

```
boundp let* multiple-value-bind ++
```

- Conses as Forms

Example

```
(setq x 1) (boundp 'x)
```

Form Categories

- Symbols as Forms

Example

```
boundp let* multiple-value-bind ++
```

- Conses as Forms

Example

```
(setq x 1) (boundp 'x)
```

- Self-Evaluating Objects

Example

```
3 #c(2/3 5/8) "Hello, world."
```

Date Types

- Number
- Character
- Symbol
- List and Cons
- Array
- Vector
- String
- Bit-Vector
- Hash-Table
- Readtable
- Package
- Pathname
- Stream
- Random-State
- Structure
- Function

Define Functions(Part 1) - Lambda and Defun

Lambda is used for defining a new anonymous function

Example

```
(lambda (x) (* x 2))
```

Defun is used for defining or redefining a named function

Example

```
(defun times2 (n) (* n 2))
```

Define Functions(Part 2) - Lambda List

Definition

Lambda list is a list for describing how the arguments are received when calling a function

lambda list keywords

&allow-other-keys	&key	&rest
&aux	&optional	

Example

```
(defun fn1 (a &optional b) (list a b))
(fn1 1) => (1 NIL)
(fn1 1 2) => (1 2)
(defun fn2 (a &key b c) (list a b c))
(fn2 1) => (1 NIL NIL)
(fn2 1 :c 2) => (1 NIL NIL)
(defun fn3 (a &rest args) (list a args))
(fn3 1 2 3 4 5) => (1 (2 3 4 5))
```

Special Operators

Definition

The operator treats its subexpressions with special syntax and evaluating rules

block	let*	return-from
catch	load-time-value	setq
eval-when	locally	symbol-macrolet
flet	macrolet	tagbody
function	multiple-value-call	the
go	multiple-value-prog1	throw
if	progn	unwind-protect
labels	progv	
let	quote	

Examples for Special Operators

1. GOTO in C(from binghe)

```
int fact(int n)
{
  int result = 1;
a:
  result *= n--;
  if (n > 0) goto a;
  return result;
}
```

Examples for Special Operators

1. GOTO in C(from binghe)

```c
int fact(int n)
{
  int result = 1;
a:
  result *= n--;
  if (n > 0) goto a;
  return result;
}
```

2. GOTO in Common Lisp(from binghe, too)

```lisp
(defun fact (n)
  (let ((result 1))
    (tagbody
     a
       (setq result (* result n))
       (decf n)
       (if (> n 0) (go a)))
    result))
```

What's multiple values return

Definition

One function call yields any number of values

Example

1. `(values 1 2 3) => 1, 2, 3`
2. `(truncate 10 3) => 3, 1`
3. `(ceiling pi) => 4, -0.85840734641020690d0`
4. `(intern "ABC") => ABC, NIL`

Advantages(Part 1) - Avoid Construction

Spliting a string from given position

Example

1. Version 1: Using construction.

```
(defun split-string (string position)
  (cons (subseq string 0 position)
        (subseq string position)))
```

Advantages(Part 1) - Avoid Construction

Spliting a string from given position

Example

1. Version 1: Using construction.

```
(defun split-string (string position)
  (cons (subseq string 0 position)
        (subseq string position)))
```

2. Version 2: Using multiple values return.

```
(defun split-string (string position)
  (values (subseq string 0 position)
          (subseq string position)))
```

Advantages(Part 2) - Avoid Different Meanings Of Return Value

Example

small

```
(defvar *ht* (make-hash-table))
(setf (gethash 1 *ht*) 'nil)
(gethash 1 *ht*) => NIL, T
(gethash 2 *ht*) => NIL, NIL
```

Common Lisp Object System

Definition

The facility for implementing the object-oriented programming paradigm in Common Lisp

Functionalities and Features

1. Class-based
2. Define new classes
3. Define new generic functions and method
4. Multiple inheritance
5. Metaobject Protocol
6. Method combinator and define new method combinators

Examples

1. Define a new class

```
(defclass person ()
  (name age))
```

Examples

1. Define a new class

```
(defclass person ()
  (name age))
```

2. Define the same class with more slot options

```
(defclass person ()
  ((name
    :initarg :name
    :accessor person-name
    :initform "default name")
   (age
    :initarg :age
    :accessor person-age
    :type integer)))
```

What's MACRO

Definition

Macro is a facility for transforming one expression to another expression and then evaluates it.

What's MACRO

Definition

Macro is a facility for transforming one expression to another expression and then evaluates it.

Usages of Macro

1. Transformation
2. Establish bindings
3. Controls the evaluation of each arguments
4. Access the context

Define New Control Structures

DO = TAGBODY + GO

Example

```
(defmacro my-do (varlist endlist &body body)
  (let ((exit-point (gensym)) (body-point (gensym)))
    `(block nil
       (let ,(mapcar #'(lambda (vl) (list (first vl) (second vl))) varlist)
         (tagbody
            (go ,exit-point)
            ,body-point
            (progn ,@body)
            (psetq ,@(mapcar #'(lambda (vl) (list (first vl) (third vl)))
                             varlist))
            ,exit-point
            (unless ,(car endlist) (go ,body-point))
            (return-from nil (progn ,@(cdr endlist))))))))
```

SETF: General Assignment

Regular assignment

Example

```
(defvar *ht* (make-hash-table))
(setf *ht* nil)
*ht* => NIL
```

Access places

Example

```
(defvar *a* (cons 1 2))
*a* => (1 . 2)
(setf (cdr *a*) 3)
*a* => (1 . 3)
```

Access slots

Example

```
(defclass foo () ((a :initarg :a)))
(defvar *i* (make-instance 'foo :a 123))
(slot-value *i* 'a) => 123
(setf (slot-value *i* 'a) 321)
(slot-value *i* 'a) => 321
```

What's readtable

The object contains associations between macro characters and their reader macro functions and controls the behavior of the reader.

What's readtable

The object contains associations between macro characters and their reader macro functions and controls the behavior of the reader.

Pre-Defined Macro Characters

( ) ' ; " ` , #

What's readtable

The object contains associations between macro characters and their reader macro functions and controls the behavior of the reader.

Pre-Defined Macro Characters

( ) ' ; " ` , #

Customizes the readtable for adding new lexical rules

Adds a syntactic sugar in Common Lisp for reading hash tables

Something Else Intresting

1. Loop Macro - DSL for iteration
2. Pretty Printing
3. MOP - Metaobject Protocol
4. Condition System - Signal, Handle and Restart.