EulerLisp Object System

Leon Rische

[2019-08-22 Thu 21:48]

Contents

EulerLisp comes with a simple object system based on `MEROON` as described in chapter 11 of "Lisp In Small Pieces". (TODO: proper reference / sci quoting)

Interesting side note: Smalltalk 72, the first object oriented language, was first implemented in Lisp.

Compared to the object systems e.g. in Ruby, Java or Python, object systems in Lisp are different in that they are based on **generic methods** and their multimethods, a technique known as **multiple dispatch**. (TODO, 1-1 quote) (TODO, obj. system has no multimethods)

There is no difference between sending a message to an object and calling a function.

The second difference is **reflection**, allowing the object system to speak about itself. The class of an object is itself an object of meta-class which is itself an object and so on.

The behavior of such meta-objects is known as the **Meta-Object Protocol** (c.f. "The Art of the Meta-Object Protocol").

Properties:

1. All "builtin" values can be represented by objects
2. Self describing
3. Generic functions (like CLOS), but without multimethods
4. Efficient Code

Objects are represented by vectors. The following indices are reserved:

1. Index 0, for the class of the object as a number indexing into a global list of classes

Classes are stored in a vector `*classes*`. Each class has to have a name, there are no anonymous classes. Classes are seen as **static** objects.

`(lookup-class name)` returns the class for the symbol `name`.

Generics are stored in a list `*generics*`, `(lookup-generic name)` returns the generic function for the symbol `name`.

There are two kinds of fields, **normal** and **indexed**.

Classes are defined using `(defclass name superclass-name fields)`. For each class, a number of functions are created:

1. A predicate for checking if an object belongs to the class
2. One allocator to create new objects of a class with **uninitialized** fields `(allocate-<classname> ...)`
3. One allocator to create new objects with initialized fields
4. Selectors for both normal and indexed fields
   - `(<classname>-<fieldname> o)`
   - `(set-<classname>-<fieldname>! o)`
5. Functions `(<classname>-<fieldname>-length o)` returning the length of each indexed field

For reflective operations, a new object of class `Class` named `<classname>-class` is generated.

At this point, "real" macros are needed.