Lisp to Ruby to Rubinius

ネットワーク応用通信研究所
楽天 技術研究所
Rubyアソシエーション
@yukihiro_matz

Yukihiro "Matz" Matsumoto
Lisp
Lisp

- one of the oldest
- O-Parts
  - out of place artifact
O-Parts of the language

oldest but newest

- symbolic computation
- garbage collection
- objects
- exceptions
Lisp

- S-expression
- macros
- everything object
- meta-programming
Dark side of Lisp

- Parentheses
- dangling language
- there's no lisp language
- CLOS
  - powerful but complex
Ruby

Lispy, but

• no S-expression
• no macros
• no CLOS
Ruby

- Algol-ish syntax
- Smalltalk-ish OO
- Language for ordinary programmers
(defun fact (n)
    (if (= n 1)
        1
        (* n (fact (1- n))))
  (print (format "6!=~D" (fact 6)))
; => 6!=720
```lisp
;; move parens
(defun fact (n)
  (if (= n 1)
      1
      (* n (fact (1- n))))
)
(print (format "6!=~D" (fact 6)))
```
;; operator syntax
(defun fact (n)
  (if (n == 1)
    1
    (n * (fact (n - 1)))))
(print (format "6!==D" (fact 6)))
;;; move argument parens
(defun fact (n)
  (if (n == 1)
      1
      (n * fact(n - 1))
  )
)
print(format("6!=~D", fact(6)))
;;; move argument parens
(defun fact (n)
    (if (n == 1)
        1
        (n * fact(n - 1))
    )
)
print(format("6!=~D", fact(6)))
Lisp

# syntax structures
defun fact (n)
  if (n == 1)
    1
  else
    (n * fact(n - 1))
  end
end
end
print(\"6!=\"D\", fact(6)))
# reduce parens

defun fact (n)
    if n == 1
        1
    else
        n * fact(n - 1)
    end
end

print(format("6!=~D", fact(6)))
```ruby
def fact(n)
  if n == 0
    1
  else
    n * fact(n - 1)
  end
end
printf "6!=%d", fact(6), "\n"
# => 6!=720
```
Lisp vs Ruby

Syntax

• Less parentheses

• Many 'end's

• special forms vs syntax structures
Lisp vs Ruby

Semantics

- Quite similar
- nearly one to one translation
- auto conversion from fixnums to bignums
  - fact 200
MatzLisp
MatzLisp

- not MacLisp
- not FranzLisp
Ruby

- Lisp without S-expression
- sprinkled with syntax sugar
- with OO from Smalltalk
- operators from C
- strings/regexp from Perl
Lisp without S-expr

Remember M-expression
Ruby = Weak Lisp?

- No S-expression
- No Macro
- Who cares.
Language Power ≠ Programming Power
Personal History

BASIC to Lisp
BASIC
BASIC

1424 STEPS 178 MEMORIES
Got tired of BASIC

- No user defined functions
- No user defined data types
Aristocracy

Language Designers
Implementers

Programmers
I met Lisp

- in an AI book
- Lisp made my eyes open
- users can do everything
Users can do Everything

- define functions
- define data types
- enhance the language
Democracy

- no discrimination
- users can be language implementers
But, wait

Unlike politics (or like politics)

- freedom comes with responsibility
- ordinary people hate (too much) responsibility
- or too much power
Too much power

• smart people love power
• smart people underestimate ordinariness of ordinary people
中庸
Happy Medium
Happy Medium

There should be somewhere in between language aristocracy and democracy, where ordinary people can live happily, without feeling fear.
Balance

It's quite easy to pursue extreme, but seeking 'something in-between' is far more difficult.
Ruby

My answer to the ultimate question.
## Result

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>=</td>
<td>Java</td>
<td>17.915%</td>
<td>-1.47%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>=</td>
<td>C</td>
<td>17.147%</td>
<td>+0.29%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>↑</td>
<td>C++</td>
<td>9.812%</td>
<td>-0.18%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>↓</td>
<td>PHP</td>
<td>8.370%</td>
<td>-1.79%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>=</td>
<td>(Visual) Basic</td>
<td>5.797%</td>
<td>-3.40%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>↑</td>
<td>C#</td>
<td>5.016%</td>
<td>+0.83%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>↑</td>
<td>Python</td>
<td>4.583%</td>
<td>+0.65%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>↑↑↑↑↑</td>
<td>Objective-C</td>
<td>3.368%</td>
<td>+2.76%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>↓↓↓</td>
<td>Perl</td>
<td>2.447%</td>
<td>-2.08%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>=</td>
<td>Ruby</td>
<td>1.907%</td>
<td>-0.47%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>↓↓</td>
<td>JavaScript</td>
<td>1.665%</td>
<td>-1.33%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>↓</td>
<td>Delphi</td>
<td>1.585%</td>
<td>-0.39%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>=</td>
<td>Lisp</td>
<td>1.084%</td>
<td>+0.24%</td>
<td>A--</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>↓↓</td>
<td>Pascal</td>
<td>0.790%</td>
<td>-0.17%</td>
<td>A--</td>
</tr>
<tr>
<td>15</td>
<td>27</td>
<td>↑↑↑↑↑</td>
<td>Transact-SQL</td>
<td>0.771%</td>
<td>+0.40%</td>
<td>A--</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>↑↑↑↑↑</td>
<td>Go</td>
<td>0.728%</td>
<td>+0.73%</td>
<td>A--</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>↑↑↑↑</td>
<td>RPG (OS/400)</td>
<td>0.715%</td>
<td>+0.26%</td>
<td>A--</td>
</tr>
<tr>
<td>18</td>
<td>30</td>
<td>↑↑↑</td>
<td>PowerShell</td>
<td>0.686%</td>
<td>+0.42%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>24</td>
<td>↑↑↑↑↑↑</td>
<td>Ada</td>
<td>0.676%</td>
<td>+0.29%</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>↓↓↓↓↓</td>
<td>PL/SQL</td>
<td>0.637%</td>
<td>-0.18%</td>
<td>A--</td>
</tr>
</tbody>
</table>
Self-sustaining languages

- C
- Lisp
- MFTL
- Ruby
My Favorite Toy Language

n. Describes a language about which the developers are passionate but no one else cares about.

--Jargon File
My Favorite Toy Language

The first great goal in the mind of the designer of an MFTL is usually to write a compiler for it

--Jargon File
I am not a fan of meta-circular implementation
Switching the brain

- C → Core mode
- Ruby → App mode
Self-sustaining systems

C

- Compiler written in C
- Compiled code does not rely on the language
Self-sustaining systems

Squeak

- Bootstrap
- Core written in Slang
  - subset of Smalltalk
  - compiles to C
- libraries in Smalltalk
Rubinius

Ruby implementation influenced by Smalltalk
Rubinius

- small VM in C++
- libraries written in Ruby
Bootstrap

1. VM
2. alpha
3. bootstrap
4. platform
5. common
6. delta
The virtual machine is able to load and execute bytecode, send messages (i.e. look up and execute methods), and all primitive functions are available, but not yet hooked up as Ruby methods.
VM

At this point there is enough defined behavior to begin to load up the rest of the runtime kernel which is all defined in ruby. This has to be done in several passes as the language grows.
alpha

This starts the loading of Ruby code. The ability to open classes and modules and define methods exists. Also, it is possible to raise exceptions and cause the running process to exit. This stage lays the foundation for the next two stages.
This stage continues to add the minimum functionality to support loading platform and common. The primitive functions are added for most of the kernel classes.
The FFI system is implemented and Ruby method interfaces to platform-specific functions are created. Once this is set up, platform specific things such as file access, math, and POSIX commands are attached.
The vast majority of the Ruby core library classes are implemented. The Ruby core classes are kept as implementation-neutral as possible.
delta

Implementation-specific versions of methods that override the versions provided in common are added.
Problems

- performance
- open class
(Possible) Solution

- JIT (LLVM)
- selector namespace / classbox
Thank you!