Elixir紹介

2014/05/08
SICP Club LT

Livesense Inc.
HORINOUCHI Masato

Elixirとは

Elixir is a functional, meta-programming aware language built on top of the Erlang VM. It is a dynamic language that focuses on tooling to leverage Erlang's abilities to build concurrent, distributed and fault-tolerant applications with hot code upgrades.

見た目は

Ruby

中身は

Lisp

特徴

- 近代的なシンタックス
- 全てが式
- 強力なメタプログラミング機能
- 第一級オブジェクトとしてのドキュメント
- Erlangランタイムとの相互運用

Fibonacci (Ruby)

```ruby
class Fib
  def self.fib n
    case n
    when 0 then 0
    when 1 then 1
    else fib(n - 1) + fib(n - 2)
    end
  end
end

puts Fib.fib 10
```

Fibonacci (Elixir)

```elixir
defmodule Fib do
  def fib(0), do: 1
  def fib(1), do: 1
  def fib(n), do: fib(n-1) + fib(n-2)
end

IO.puts Fib.fib 6
```

Install

- OS X (homebrew)
  - $ brew install erlang –devel
  - $ brew install elixir
- Windows
  - なんか大変らしい
- Ubuntu
  - PPA あるよ

REPL

```
$ iex
iex> defmodule Hello do
...>   def world do
...>     IO.puts "Hello, world"
...>   end
...> end
iex> Hello.world
Hello, world
:ok
iex>
```

無名関数

```
iex> f = fn(x) -> x * 2 end
iex> f.(4)
8
iex> (fn(x) -> x * 3 end).(3)
9
```

map reduce

```
iex> Enum.map(
...>   [1,2,3],
...>   fn(x) -> x * 2 end)
[2, 4, 6]
iex> Enum.reduce(
...>   [1,2,3],
...>   fn(x, acc) -> x + acc end)
6
```

キーワード引数と括弧の省略

```
iex> if true do
...>   1
...> else
...>   2
...> end
1
iex> if true, do: 1, else: 2
1
iex> if true, [do: 1, else: 2]
1
iex> if(true, [do: 1, else: 2])
1
```

Homoiconicity

In a homoiconic language the primary representation of programs is also a data structure in a primitive type of the language itself. This makes metaprogramming easier than in a language without this property, since code can be treated as data: reflection in the language (examining the program's entities at runtime) depends on a single, homogeneous structure, and it does not have to handle several different structures that would appear in a complex syntax. To put that another way, homoiconicity is where a program's source code is written as a basic data structure that the programming language knows how to access.

quote

```
iex> length [1,2,3]
3
iex> quote do: length [1,2,3]
{:length, [context: Elixir, import: Kernel], [[1, 2, 3]]}
iex> 1 + 2
3
iex> quote do: 1 + 2
{:+, [context: Elixir, import: Kernel], [1, 2]}
```

macro

```
iex> defmodule MyUnless do
...>   defmacro unless(clause, options) do
...>     quote do: if !unquote(clause),
...>                  unquote(options)
...>   end
...> end
iex> require MyUnless
nil
Iex> MyUnless.unless true, do: 1, else: 2
2
```

defmacro if

```
defmacro if(condition, clauses) do
  do_clause = Keyword.get(clauses, :do, nil)
  else_clause = Keyword.get(clauses, :else, nil)

  quote do
    case unquote(condition) do
      _ in [false, nil] -> unquote(else_clause)
      _                 -> unquote(do_clause)
    end
  end
end
```

if すらもマクロ

ご清聴ありがとうございました