Elixir 触ってみた @ Ruby札幌28

ヽ（´・肉・｀）ノ

本人いわく

http://elixir-lang.org/
に書いてある Elixir による自己紹介

全部が式

```
iex(1)> defmodule Hello do
...(1)>   IO.puts "Defining the function world"
...(1)>
...(1)>   def world do
...(1)>     IO.puts "Hello World"
...(1)>   end
...(1)>
...(1)>   IO.puts "Function world defined"
...(1)> end
Defining the function world
Function world defined
iex(2)> Hello.world
Hello World
:ok
```

全部が式(2)

- module は沢山の式からなりたっている.
- module の内容をプログラムで書ける

メタプログラミングできる

メタプログラミングとDSL

DSL を簡単に作れる(ExUnitの例)

```
defmodule MathTest do
  use ExUnit.Case

  test "can add two numbers" do
    assert 1 + 1 == 2
  end
end
```

protocol によるポリモーフィズム

ファイルにも配列にも使えるEnumモジュール

```
Enum.map([1,2,3], fn(x) -> x * 2 end) #=> [2,4,6]

# ----

file  = File.stream!("README.md")
lines = Enum.map(file, fn(line) -> Regex.replace(%r/"/, line, "'") end)
File.write("README.md", lines)
```

protocol によるポリモーフィズム(2)

- 自作モジュールでも Enum を使いたい
- Enum は Enumerable という protocol があれば使える
- MyModule 向けに Enumerable を実装する

protocol によるポリモーフィズム(3)

```
defimpl Enumerable, for: MyModule do
  def reduce(collection, acc, fun), do
    # (....)
  end

  def member?(collection, value), do
    # (....)
  end

  def count(collection), do
    # (....)
  end
end
```

一級市民としてのドキュメント

- 言語レベルでドキュメント化をサポートしている.
  - 色んなツールで簡単にドキュメントを使える.
- マークアップ記法として Markdown を使える.

一級市民としてのドキュメント(2)

```
defmodule MyModule do
  @moduledoc """
  Documentation for my module. With **formatting**.
  """

  @doc "Hello"
  def world do
    "World"
  end
end
```

一級市民としてのドキュメント(3)

```
$ iex -r my_module.exs
iex(1)> h MyModule
# MyModule

Documentation for my module. With **formatting**.
iex(2)> h MyModule.world
* def world()

Hello
```

パターンマッチング

まとまっているものをバラバラにして扱いやすくする

```
iex(1)> u = { :user, "John Doe", 19 }
{:user, "John Doe", 19}
iex(2)> elem u, 1
"John Doe"
iex(3)> { type, name ,age } = { :user, "John Doe", 19 }
{:user, "John Doe", 19}
iex(4)> type
:user
iex(5)> name
"John Doe"
iex(6)> age
19
```

パターンマッチング(2)

ガード節 (when) と混ぜると意図が伝わりやすくなる

```
def serve_drinks({ User, name, age }) when age < 20 do
  raise "No way #{name}!"
end

def serve_drinks({ User, name, age }) do
  # Code that serves drinks!
end

# ----

serve_drinks User.get("John")
#=> Raises "No way John!" if John is under 20
```

隅から隅まで Erlang

```
:application.start(:crypto)
:crypto.md5("Using crypto from Erlang OTP")
#=> <<192,223,75,115,...>>
```

- バイトコードレベルで互換
- 変換が容易
- Elixir から Erlang の関数はコスト 0 で実行できる

本人いわく，のまとめ

- 全部が式
- メタプログラミングとDSL
- protocol によるポリモーフィズム
- 一級市民としてのドキュメント
- パターンマッチ
- 隅から隅まで Erlang

触ってみたくなった？

インストール方法は
http://elixir-lang.org/getting_started/1.html
の 1.1 Installation に書いてある.

- Erlang R16B 以降
- Elixir

が必要.

触ってみたくなった？(Mac)

```
brew install elixir
```

で両方インストールできる.

触ってみたくなった？ (Windows)

- Erlang: http://www.erlang.org/download.html
- Elixir: https://github.com/elixir-lang/elixir/releases/

それぞれのコンパイル済 zip をダウンロードして解凍して使うのが簡単でおすすめ (らしい)

モダンなプログラミング言語

最近のプログラミング言語が備えている特徴
Elixir も備えている

パッケージ管理

mix :: Ruby の Rake と Bundler を合わせたようなもの

- mix new: プロジェクトを作る
- mix test: テストを実行する
- mix compile: コンパイルする

mix –help で詳しくみられる

ライブラリ管理 (みあたらず)

- rubygems を操作する gem のようなコマンドはまだ見つけられない
- rubygems 相当のライブラリ置き場は http://expm.co というのがある

REPL

iex :: Ruby の irb のようなもの

```
iex(1)> "ほげほげ"
"ほげほげ"
iex(2)> 1 + 1
2
iex(3)> def foo do
...(3)> "foo"
...(3)> end
** (SyntaxError) iex:3: cannot invoke def outside module
    src/elixir_macros.erl:184: :elixir_macros.translate/2
    lists.erl:1339: :lists.mapfoldl/3
    src/elixir.erl:134: :elixir.eval_forms/3
iex(3)> defmodule Foo do
...(3)> def bar do
...(3)> "bar"
...(3)> end
...(3)> end
iex(4)> Foo.bar
"bar"
```

ユニットテスト

ExUnit :: Ruby の Test::Unit みたいなもの

```
defmodule MathTest do
  use ExUnit.Case

  test "can add two numbers" do
    assert 1 + 1 == 2
  end
end
```

モダンなプログラミング環境のまとめ

- パッケージ管理
- ライブラリ管理(みあたらず)
- REPL
- ユニットテスト

Elixir らしそうなところ

個人的におおっ！
となったところ

マクロ

Elixir の内容は全て 3 要素のタプルで表されている

- atom か，同じ形式のタプル
- メタデータのリスト．ノードの番号とか行番号などを保持する
- 呼び出す関数の引数のリストか atom

ほとんどの構文がマクロで作られている

マクロ(2)

```
iex(1)> 1 + 2
3
iex(2)> quote do: 1 + 2
{:+, [context: Elixir, import: Kernel], [1, 2]}
iex(3)> defmodule MyMacro do
...(3)> defmacro one_plus_two do
...(3)> {:+, [], [1,2]}
...(3)> end
...(3)> end
iex(4)> require MyMacro
nil
iex(5)> MyMacro.one_plus_two
3
```

マクロ(3)

```
iex(1)> defmodule MyMacro do
...(1)>   defmacro unless(clause, options) do
...(1)>     quote do: if(!unquote(clause), unquote(options))
...(1)>   end
...(1)> end
iex(2)>
nil
iex(3)> require MyMacro
nil
iex(4)> MyMacro.unless true, do: IO.puts "false"
nil
iex(5)> MyMacro.unless false, do: IO.puts "false"
false
:ok
```

並列

並列があたりまえ.
簡単に作れるようになっている.

- spawn : 違うプロセスを作る
- x <- y : プロセス x に y という内容を送る
- receive : 送られた内容を取得する

並列(2)

```
iex(1)> current_pid = self
#PID<0.26.0>
iex(2)> spawn fn ->
...(2)>   current_pid <- { :hello, self }
...(2)> end
#PID<0.40.0>
iex(3)> receive do
...(3)>   { :hello, pid } ->
...(3)>     IO.puts "Hello from #{inspect(pid)}"
...(3)> end
Hello from #PID<0.40.0>
:ok
```

並列(3)

https://gist.github.com/niku/7301933

普通の MacBook で 100 万プロセス生成 16 秒で動くんだぜー

```
$ elixir --erl "+P 1000000" -r chain.exs -e "Chain.run(1_000_000)"
{16961375, "Result is 1000000"}
```

OTP

OTPとは何か？

http://www.ymotongpoo.com/works/lyse-ja/ja/16_what_is_otp.html

- 大抵のプロセスでは，共通の処理がある
- パターンを見極めて，共通ライブラリにまとめたもの

OTP(2)

OTP の便利なところ(一部)

- ワーカープロセスの監視/再起動が **組み込まれている**
- ダウンタイム **ゼロ** のリリース, デプロイ

Elixir らしそうなところのまとめ

- マクロ (Elixirすごい)
- 並列 (Elixirが使っているErlangVMすごい)
- OTP (Elixirが使っているErlangのライブラリすごい)

思考の転換

プログラマの思考はプログラミング言語に影響される

http://gihyo.jp/news/report/01/rubykaigi2013/0001

“まつもとゆきひろさん，Rubyに影響を与えた言語とRuby開発初期を語る。 ～ RubyKaigi 2013 基調講演 1日目”

想像してみてほしい

- もし並列処理が簡単に書けるなら
- もし無制限に並列処理できるなら

どんな考え方をするだろう？
Elixir を使って試してみようぜ.

参考にしている本/サイト

- http://elixir-lang.org/getting_started/
- http://www.ymotongpoo.com/works/lyse-ja/index.html
- http://pragprog.com/book/elixir/programming-elixir