Techmix Hokkaido 2014 Elixir

発表するひと

- ヽ（´・肉・｀）ノ
- @niku_name

コンサドーレ札幌アドベントカレンダー2014

http://www.adventar.org/calendars/406

Elixriいつできたの？

- https://github.com/elixir-lang/elixir/
- 最初のコミットは **３ 年前 (2011-01-09)**

誰が作りはじめたの？

- **Jose Valim** さん
- Ruby On Rails のコミッタ
- RubyKaigi2013 のキーノート
  - http://gihyo.jp/news/report/01/rubykaigi2013/0002

どんな言語なの？

- ErlangVM の上で動作する動的な関数型言語
- Elixirの特徴
  - http://qiita.com/niku/items/7c61d6a6af38896ac603

いいところ

	一般的	+ α がある
特徴的		Lisp, Haskell, Erlang…
親しみやすそう	普段お使いの言語	Elixir, Scala

Elixir

- (僕が) **したしみやすそう**
- 便利
  - 道具が揃ってる :: **ErlangVM** や **Erlang** のライブラリを使える
  - 構文を拡張しやすい :: **Lisp** みたいなマクロが書ける

したしみ

- 構文
- ツール

構文

```elixir
1 + 1               # => 2
"abc" <> "def"      # => "abcdef"
Enum.map([1, 2, 3], fn(x) -> x * 2 end) # => [2, 4, 6]

defmodule Greeting do
  def hello(name) do
    "hello " <> name
  end
end

Greeting.hello("ニク") # => "hello ニク"
```

ツール

- ユニットテストツール **ExUnit**
- インタラクティブシェル **IEx**
- ビルドツール **Mix**
- パッケージ管理ツール **Hex**

ためしてみよう

プロジェクト作成

```
/Users/niku/projects% mix new exhomunculus
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/exhomunculus.ex
* creating test
* creating test/test_helper.exs
* creating test/exhomunculus_test.exs

Your mix project was created successfully.
You can use mix to compile it, test it, and more:

    cd exhomunculus
    mix test

Run `mix help` for more commands.
```

テスト実行

```
/Users/niku/projects% cd exhomunculus
/Users/niku/projects/exhomunculus% mix test
Compiled lib/exhomunculus.ex
Generated exhomunculus.app
.

Finished in 0.1 seconds (0.1s on load, 0.01s on tests)
1 tests, 0 failures

Randomized with seed 273204
```

挨拶を返す

挨拶(greet)を命令すると
“graaaa…”と返すようにしたい

```
# test/exhomunculus_test.exs
defmodule ExhomunculusTest do
  use ExUnit.Case

  test "the truth" do
    assert 1 + 1 == 2
  end

  test "greet" do
    assert Exhomunculus.greet == "graaaa..."
  end
end
```

テスト実行

まだ作ってないので**失敗**する

```
/Users/niku/projects/exhomunculus% mix test


  1) test greet (ExhomunculusTest)
     test/exhomunculus_test.exs:8
     ** (UndefinedFunctionError) undefined function: Exhomunculus.greet/0
     stacktrace:
       (exhomunculus) Exhomunculus.greet()
       test/exhomunculus_test.exs:9

.

Finished in 0.07 seconds (0.06s on load, 0.01s on tests)
2 tests, 1 failures

Randomized with seed 848607
```

プロダクションコード実装

```
# lib/exhomunculus.ex
defmodule Exhomunculus do
  def greet do
    "graaaa..."
  end
end
```

テスト実行

```
/Users/niku/projects/exhomunculus% mix test
..

Finished in 0.05 seconds (0.05s on load, 0.00s on tests)
2 tests, 0 failures

Randomized with seed 150260
```

通った.

いつも通りに開発できる.

コマンドラインから使う

escript を使えばできる

```
/Users/niku/projects/exhomunculus% mix escript.build
Consolidated Access
Consolidated Collectable
Consolidated Enumerable
Consolidated Inspect
Consolidated List.Chars
Consolidated Range.Iterator
Consolidated String.Chars
Consolidated protocols written to _build/dev/consolidated
** (Mix) Could not generate escript, please set :main_module \
in your project configuration (under `:escript` option) to a module that implements main/1
```

エラーが!

エラーメッセージをよくみる

(Mix) Could not generate escript,

please set **:main_module** in your project configuration

(under `:escript` option) to a module

that **implements main/1**

必要なものを実装

エラーメッセージから

- **main** 関数がどのモジュールにあるかを **config で指定**する
- main/1 **関数名がmain**で**引数が1つ**の関数の用意をする

が必要だとわかる

configで指定

```
# mix.exs
defmodule Exhomunculus.Mixfile do
  use Mix.Project

  def project do
    [app: :exhomunculus,
     version: "0.0.1",
     elixir: "~> 1.0",
     deps: deps,
     escript: escript] # ここと
  end

  def application do
    [applications: [:logger]]
  end

  defp deps do
    []
  end

  def escript do
    [main_module: Exhomunculus] # ここを追加
  end
end
```

main/1 関数の用意

```
# lib/exhomunculus.ex
defmodule Exhomunculus do
  def main(args) do #ここを追加
    IO.puts greet   #
  end               #

  def greet do
    "graaaa..."
  end
end
```

実行

```
/Users/niku/projects/exhomunculus% mix escript.build
lib/exhomunculus.ex:2: warning: variable args is unused
Compiled lib/exhomunculus.ex
Generated exhomunculus.app
Consolidated Access
Consolidated Collectable
Consolidated Enumerable
Consolidated Inspect
Consolidated List.Chars
Consolidated Range.Iterator
Consolidated String.Chars
Consolidated protocols written to _build/dev/consolidated
Generated escript exhomunculus with MIX_ENV=dev
/Users/niku/projects/exhomunculus% ./exhomunculus
graaaa...
```

天気情報を取得する

- HTTP経由でAPIから天気を取得
  - http://weather.livedoor.com/weather_hacks/webservice

外部ライブラリの利用

- **HTTPアクセス** するのに **httpoison**
  - https://github.com/edgurgel/httpoison
- **JSON解析** するのに **exjsx**
  - https://github.com/talentdeficit/exjsx

ライブラリの利用宣言

```
# mix.exs
defmodule Exhomunculus.Mixfile do
  use Mix.Project

  def project do
    [app: :exhomunculus,
     version: "0.0.1",
     elixir: "~> 1.0",
     deps: deps,
     escript: escript]
  end

  def application do
    [applications: [:logger, :httpoison]] #ここ
  end

  defp deps do
    [httpoison: "~> 0.5", #ここ
     exjsx: "~> 3.0"]     #
  end

  def escript do
    [main_module: Exhomunculus]
  end
end
```

ライブラリのダウンロード

```
/Users/niku/projects/exhomunculus% mix deps.get
```

インタラクティブシェル IEX

動作を確認するのに使う

```
/Users/niku/projects/exhomunculus% iex -S mix
(...snip...)
iex(1)> "こんにちは"
"こんにちは"
iex(2)> [1, 2, 3]
[1, 2, 3]
iex(3)> "hi" <> "ho"
"hiho"
iex(4)> 1 + 2
3
```

ライブラリも試せる

```
/Users/niku/projects/exhomunculus% iex -S mix
iex(1)> url = "http://weather.livedoor.com/forecast/webservice/json/v1?city=016010"
"http://weather.livedoor.com/forecast/webservice/json/v1?city=016010"
iex(2)> {:ok, %HTTPoison.Response{status_code: 200, body: body}} =
...(2)> HTTPoison.get(url)
(...snip...)
iex(3)> json = JSX.decode! body
(...snip...)
iex(4)> json["description"]["text"]
" 北海道の西海上に低気圧があって、24日夜に北海道付近を通過し（略）"
```

ここまで

ふつうです ^q^

ここから

ふつうではありません ^q^

ErlangVM由来のふつうではないこと

- **プロセスをたくさん**つくれる
- **分散**させやすい
- **耐障害性**が高い

プロセスをたくさんつくれる

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

プロセスをたくさんつくれる

```
defmodule Chain do
  def counter(next_pid) do
    receive do
      n ->
        send next_pid, n + 1
    end
  end

  def create_processes(n) do
    last = Enum.reduce 1..n, self,
             fn (_,send_to) ->
               spawn(Chain, :counter, [send_to])
             end

    send last, 0

    receive do
      final_answer when is_integer(final_answer) ->
        "Result is #{inspect(final_answer)}"
    end
  end

  def run(n) do
    IO.puts inspect :timer.tc(Chain, :create_processes, [n])
  end
end
```

同時に100万個生成

```
$ elixir --erl "+P 1000000" -r chain.exs -e "Chain.run(1_000_000)"
{9482041, "Result is 1000000"}
```

Macbook Proで9秒

分散させやすい

```
/Users/niku% mkdir a b
/Users/niku% cd a
/Users/niku/a% touch a.txt a.csv
/Users/niku/a% ls
a.csv  a.txt
/Users/niku/a% iex --sname a
iex(a@niku-MacBook-Pro)1> node
:"a@niku-MacBook-Pro"
iex(a@niku-MacBook-Pro)2> File.ls
{:ok, ["a.csv", "a.txt"]}

--------------------------------

/Users/niku% cd b
/Users/niku/b% touch b.txt
/Users/niku/b% ls
b.txt
/Users/niku/b% iex --sname b
iex(b@niku-MacBook-Pro)1> File.ls
{:ok, ["b.txt"]}
iex(b@niku-MacBook-Pro)2> Node.connect(:"a@niku-MacBook-Pro")
true
iex(b@niku-MacBook-Pro)3> current = self
#PID<0.59.0>
iex(b@niku-MacBook-Pro)4> Node.spawn(:"a@niku-MacBook-Pro",
                           fn -> send(current, File.ls) end)
#PID<9172.65.0>
iex(b@niku-MacBook-Pro)5> flush
{:ok, ["a.csv", "a.txt"]}
```

耐障害性が高い

(省略)

力不足により簡単なサンプルを用意できませんでした

Elixir独自のふつうではないこと

- マクロ！
- Elixirでメタプログラミングするのに使う

マクロ？

書いてあるコードを実行し始める前に書き換える

どんなことができるの？

たとえば IF 式が作れます

myif を作ってみる

```
if(評価) {
    // 評価がtrueのときにやること
} else {
    // 評価がfalseのときにやること
}
```

myif を作ってみる

- 評価対象(trueかfalseになる)
- true のときにやること
- false のときにやること

JavaScriptでmyif

```
function myif(condition, iftrue, iffalse) {
  return condition ? iftrue : iffalse;
}

console.log(myif(true, "_true", "_false"));  // => _true
console.log(myif(false, "_true", "_false")); // => _false
```

よさそう

Javascriptでmyif

```
function myif(condition, iftrue, iffalse) {
  return condition ? iftrue : iffalse;
}

myif(true, console.log("_true"), console.log("_false"));
```

Javascriptでmyif

```
function myif(condition, iftrue, iffalse) {
  return condition ? iftrue : iffalse;
}

myif(true, console.log("_true"), console.log("_false"));
// => _true
// => _false
```

あれ……

Javascriptでmyif

```
function myif(condition, iftrue, iffalse) {
  return condition ? iftrue : iffalse; //←ここで評価してほしい
}

myif(true, console.log("_true"), console.log("_false")); //ここで評価してしまう
```

myifの結果を返すときではなく，
myifに引数を渡すときに評価されてしまう

Javascriptでmyif

```
function myif(condition, iftrue, iffalse) {
  return condition ? iftrue.call() : iffalse.call();
}

myif(true,
     function() { return console.log("_true"); },
     function() { return console.log("_false"); }); // => _true
myif(false,
     function() { return console.log("_true"); },
     function() { return console.log("_false"); }); // => _false
```

渡すときにfunctionにしておけば，結果を返すときに評価してくれる

JavascriptでmyIf

- 引数に渡すときのfunctionを無くすことはできないのか？
- =>引数に渡ってくる内容をfunctionとして，引数を受けとるときには評価しないようにできないの？
- JavaScriptでのやり方を知らない……

奥がふかいif

いつも使っているやつなのに，思ったよりたくさんの要素がある

Elixirでmyif

- 関数でやると，JavaScriptと同じ悩みにあたる
- マクロ！

Elixirでmyif

```
defmodule My do
  defmacro my_if(clause, iftrue, iffalse) do
    IO.inspect(clause)
    IO.inspect(iftrue)
    IO.inspect(iffalse)
  end
end

defmodule MyTest do
  def run do
    import My
    My.my_if(true) do
      IO.inspect("_true")
    else
      IO.inspect("_false")
    end
  end
end

MyTest.run
```

Elixirでmyif

実行結果

```
true # => IO.inspect(clause) 分
{{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
 [line: 12], ["_true"]} # => IO.inspect(iftrue) 分
{{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
 [line: 12], ["_false"]} # => IO.inspect(iffalse) 分
"_false"
```

なんだこれは……

Elixirでmyif

http://elixir-lang.org/getting_started/meta/1.html

Elixir の文献によると

```
{ tuple か atom, list, list か atom }
```

3要素のくみあわせになっている

Elixirでmyif

- 一番目の要素はアトムかタプル．タプルの場合はこの構造と同じになっている
- 二番目の要素はメタデータ．行数とか文脈が書いてある．
- 三番目の要素はアトムか関数呼び出しの引数．アトムの場合，このひとかたまりタプルは変数である．

Elixirでmyif

IO.inspect(clause)のなかみ

```
{{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
 [line: 12], ["_true"]}

{{tuple},
 [list], [list]}
```

Elixirでmyif

1 番目のタプルのなかみ

```
{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
{:atom, [list], [list]}
```

Elixirでmyif

- 以下略.　たしかにそのようになっているようだ.
- これなんなの？ => AST(抽象構文木)です
- ASTってなんなの？プログラムの構造をプログラムで表したものです

Elixirでmyif

とりあえず忘れてください.
IO.inspect の**出力が最後に出てくる**ことに注目

```
true
{{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
 [line: 12], ["_true"]}
{{:., [line: 12], [{:__aliases__, [counter: 0, line: 12], [:IO]}, :inspect]},
 [line: 12], ["_false"]}
"_false" # <= **ココ**
```

Elixirでmyif

- IO.inspect の出力が最後に出てくる
- つまり引数で受けたときにはまだ評価されていない

Elxiirでmyif

```
defmodule My do
  defmacro my_if(clause, iftrue, iffalse) do
    quote do
      case unquote(clause) do
        false -> unquote(iffalse)
        nil -> unquote(iffalse)
        _ -> unquote(iftrue)
      end
    end
  end
end

defmodule MyTest do
  def run do
    import My
    My.my_if(true, IO.inspect("_true"), IO.inspect("_false"))
    My.my_if(false, IO.inspect("_true"), IO.inspect("_false"))
    My.my_if(nil, IO.inspect("_true"), IO.inspect("_false"))
  end
end

MyTest.run
"_true"
"_false"
"_false"
```

Elxiirでmyif

- できた
- マクロは慣れていない人類には早すぎる感

Elixirでmyif

> *Never use a macro when you can use a function*
>
> *– Programming Elixir*

まとめ

- ライブラリを使う人には現代的でしたしみやすい
- ライブラリを作る人にはほとんどなんでもできる

宣伝

- 毎週だいたい木曜日にErlangVMに関すること(Elixir含む)や，そうではないことをワイワイやるSapporo.beamをやってます
- 前回は高級トイレットペーパーのはなしやこの資料の作成をしてました