elixir

Gustavo C., Victor F.

Elixir

- Linguagem de propósito geral, com aspectos funcionais
- Foco no desenvolvimento de sistemas com alta escalabilidade, concorrência e tolerância a falhas
- Sintaxe altamente influenciada pela linguagem Ruby
- Executa na VM da Erlang, e compartilha diversas ideias e práticas com esse ecossistema

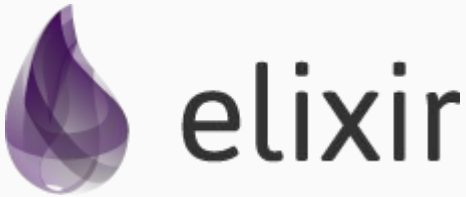
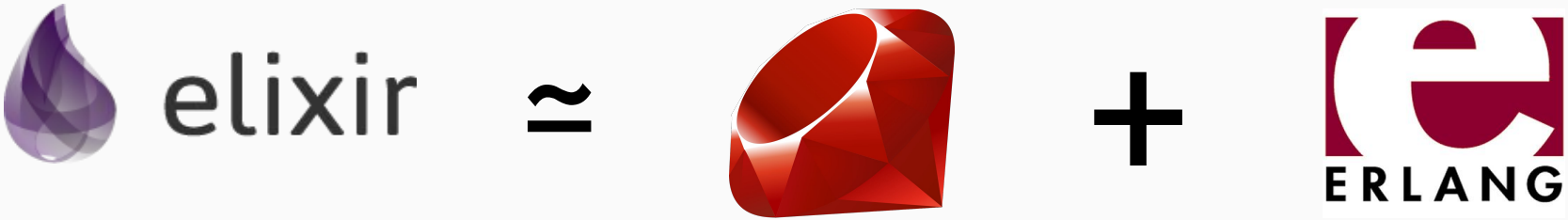


Imagem: https://elixir-lang.org/

Elixir

Simplificando...



Imagens:
Elixir: https://elixir-lang.org/
Ruby: https://commons.wikimedia.org/wiki/File:Ruby_logo.svg
Erlang: https://commons.wikimedia.org/wiki/File:Erlang_logo.png

Ruby

Ruby is...

A dynamic, open source programming language with a focus on simplicity and productivity. *It has an elegant syntax that is natural to read and easy to write.*

(https://www.ruby-lang.org/)

```ruby
# The Greeter class
class Greeter
  def initialize(name)
    @name = name.capitalize
  end

  def salute
    puts "Hello #{@name}!"
  end
end

# Create a new object
g = Greeter.new("world")

# Output "Hello World!"
g.salute
```

Código retirado de https://www.ruby-lang.org/

Erlang

What is Erlang?

Erlang is a programming language used to build massively scalable soft real-time systems with requirements on high availability. Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and fault tolerance.

(https://www.erlang.org/)

```
-module(fibonacci).    % This is the file 'fibonacci.erl', the module and the filename must match
-export([fib/1]). % This exports the function 'fib' of arity 1

fib(0) -> 0; % If 0, then return 0, otherwise (note the semicolon ; meaning 'else')
fib(1) -> 1; % If 1, then return 1, otherwise
fib(N) when N > 1 -> fib(N - 1) + fib(N - 2).
```

Código retirado da página da Wikipedia sobre Erlang:
https://en.wikipedia.org/wiki/Erlang_(programming_language)

Elixir - Sintaxe

Erlang

```erlang
-module(fibonacci).
-export([fib/1]).

fib(0) -> 0;
fib(1) -> 1;
fib(N) when N > 1 -> fib(N - 1) + fib(N - 2).
```

Ruby

```ruby
def fib(n)
  return 0 if n == 0
  return 1 if n == 1
  return fib(n-1) + fib(n-2)
end
```

Elixir

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

Códigos:
Erlang: Wikipedia: https://en.wikipedia.org/wiki/Erlang_(programming_language)
Elixir:
Gist do usuário kyanny: https://gist.github.com/kyanny/2026028

Elixir - Sintaxe: Pipes

```
iex> Enum.sum(Enum.filter(Enum.map(1..100_000, fn x -> 3*x end, odd?)))
7500000000
```

Usando pipes:

```
iex> 1..100_000 |> Stream.map(fn x -> 3*x end) |> Stream.filter(odd?) |> Enum.sum
7500000000
```

Elixir - Sintaxe: Pattern Matching

```
iex> {:ok, result} = {:ok, 13}
{:ok, 13}
iex> result
13

iex> {:ok, result} = {:error, :oops}
** (MatchError) no match of right hand side value: {:error, :oops}
```

Elixir - um pouco de história

- Criada pelo brasileiro José Valim (Destaque na comunidade Ruby e Rails) em um projeto da empresa Plataformatec
- Desenvolvida com o objetivo de simplificar o desenvolvimento na plataforma Erlang
- Open-source (Licença Apache), contando atualmente com centenas de contribuidores
- Elogiada pelo criador da Erlang

plataformatec

Imagens:
José Valim: GitHub (https://github.com/josevalim)
Plataformatec: http://plataformatec.com.br/

Voltando à Erlang...

- Desenvolvida dentro da Ericsson nos anos 80, para aplicações em telecomunicações
- Convertida em Open-source no final dos anos 90
- Foco em concorrência e alta disponibilidade:
  - Switch da Ericsson com disponibilidade de 99.9999999% (9 noves) (31 ms. em um ano!)
  - Whatsapp: 2 milhões de conexões TCP em uma única máquina
- Veremos como alguns conceitos do ambiente Erlang funcionam em Elixir

Imagens:
Ericsson: https://commons.wikimedia.org/wiki/File:Ericsson_logo.svg
Whatsapp: https://www.whatsappbrand.com/

Conceitos de Erlang/Elixir: Processos

- Em elixir, todo código roda dentro de um processo (Não confundir com os processos do SO!)
- Muito mais leve que um processo do SO, ou mesmo que threads em outras linguagens
- Não é incomum possuir dezenas ou centenas de milhares de processos rodando concorrentemente
- São completamente isolados, e se comunicam através de mensagens
- GC por processo (nada de pausas globais!)

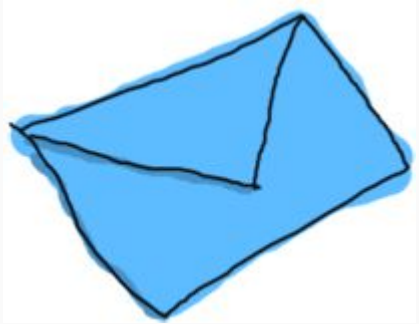


Imagem: Learn you some Erlang for great good!
(http://learnyousomeerlang.com/introduction#what-is-erlang)

Filosofia para processos: Let it Crash!

- Como os processos são leves e isolados, faz mais sentido deixar eles falharem (crash) do que programar preventivamente (tratando erros e exceções)
- Reiniciar um processo resolve bugs causados em função do estado do processo
- Bugs que não dependem do estado (ou seja, são facilmente reprodutíveis) devem ser prevenidos em tempo de desenvolvimento, usando-se testes de unidade, por exemplo

Imagem: Learn you some Erlang for great good!
(http://learnyousomeerlang.com/introduction#what-is-erlang)

Criando um processo em Elixir

spawn/1:

```
iex(1)> pid = spawn(fn -> IO.puts "Hello, Gold!" end)
Hello, Gold!
#PID<0.87.0>
iex(2)> Process.alive? pid
false
```

Processos: troca de mensagens

O próprio iex é um processo:

```
iex(8)> self()
#PID<0.84.0>
```

send/2 e receive/1:

```
iex(1)> parent = self()
#PID<0.84.0>
iex(2)> spawn fn -> send(parent, {:hello, self()}) end
#PID<0.87.0>
iex(3)> receive do
...(3)> {:hello, pid} -> "Got hello from #{inspect pid}"
...(3)> end
"Got hello from #PID<0.87.0>"
```

Processos: armazenando estado

Legal… mas e se eu precisar de estado?

Bom, que tal um processo para isso?

```
defmodule KV do
  def start_link do
    Task.start_link(fn -> loop(%{}) end)
  end

  defp loop(map) do
    receive do
      {:get, key, caller} ->
        send caller, Map.get(map, key)
        loop(map)
      {:put, key, value} ->
        loop(Map.put(map, key, value))
    end
  end
end
```

Processos: armazenando estado

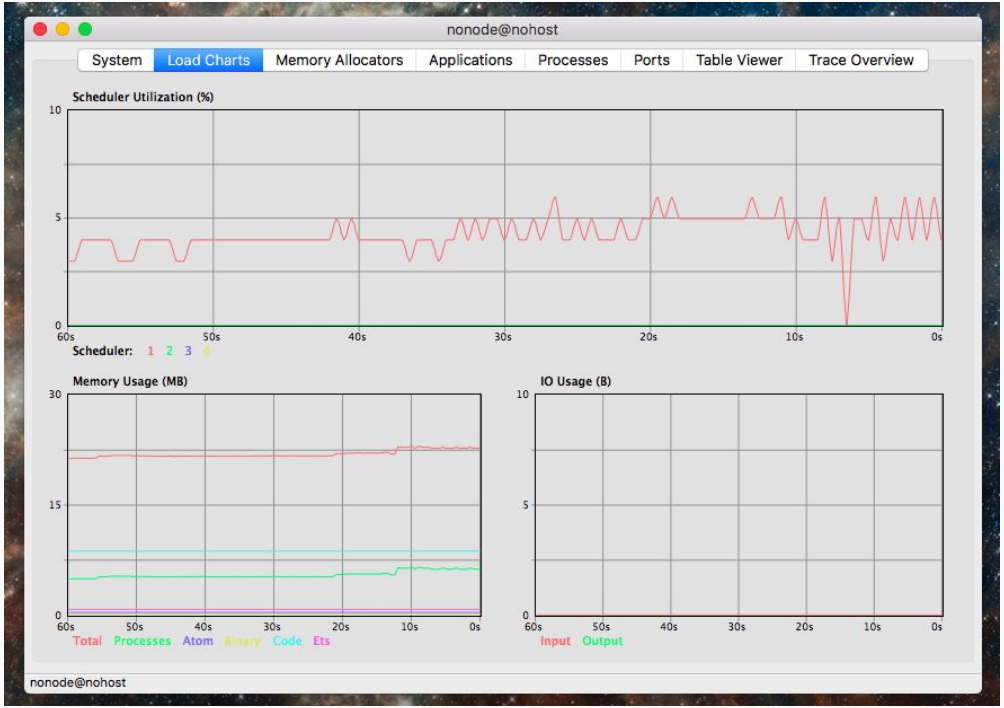
```
iex(2)> c "kv.exs"
[KV]
iex(3)> {:ok, pid} = KV.start_link
{:ok, #PID<0.94.0>}
iex(4)> send pid, {:get, :hello, self()}
{:get, :hello, #PID<0.84.0>}
iex(5)> flush()
nil
:ok
iex(7)> send pid, {:put, :hello, "Gold!"}
{:put, :hello, "Gold!"}
iex(8)> send pid, {:get, :hello, self()}
{:get, :hello, #PID<0.84.0>}
iex(9)> flush()
"Gold!"
:ok
```

Elixir e OTP

Para resolver problemas comuns como o gerenciamento de processos ou o armazenamento de estado, é possível usar o as bibliotecas do OTP (Open Telecom Platform), disponíveis no ambiente Erlang:

- Agents: Abstração de estados
- Tasks: Ferramentas para lidar com sincronização de processos
- GenServer: Interface para implementação do servidor em arquiteturas cliente/servidor
- Supervisor: Gerenciamento de uma árvore de processos

Observer (Erlang): Monitoramento da Erlang VM:



Casos de uso: Framework Phoenix

Phoenix: Principal projeto que utiliza Elixir

- Framework para aplicações web MVC server-side, na tradição do Rails

```
defmodule HelloPhoenix.PageController do
    use HelloPhoenix.Web, :controller

    def index(conn, _params) do
        render conn, "index.html"
    end
end
```

Imagem: Phoenix Framework (http://www.phoenixframework.org/)

Casos de uso: Pinterest

- API de anúncios: tempo de resposta (90%) abaixo de 800µs (microsegundos!)
- Sistema de notificações:
  - Java (10k LOC)
  - Elixir (< 1k LOC)

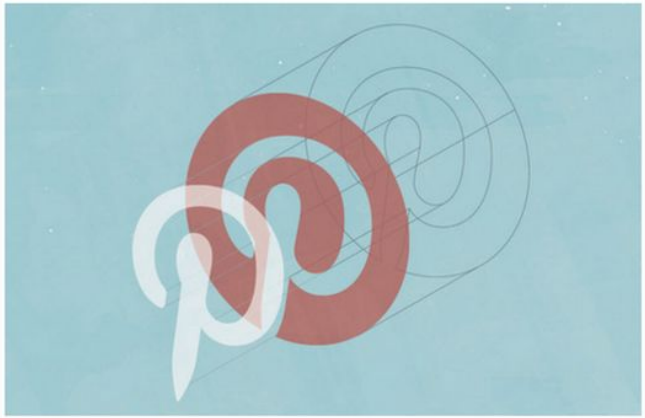


Imagem: Pinterest Engineering (https://medium.com/@Pinterest_Engineering/introducing-new-open-source-tools-for-the-elixir-community-2f7bb0bb7d8c)

Casos de uso: Rose Point

Elixir e Erlang em dispositivos embutidos, conversão de ethernet para protocolos de rede do meio náutico

Imagens:
Plataformatec (http://blog.plataformatec.com.br/2015/06/elixir-in-production-interview-garth-hitches/)
Rose Point Navigation Systems (http://wordpress.rosepoint.com/company-info/)

elixir

Gustavo C., Victor F.