

Elixir

Présentation cours PLM →

Sommaire

- Historique
- Justification du langage
- Avantages
- Erlang
- Utilisation
- Concepts
- Particularités
- Projet



elixir

Historique

- Créé par José Valim en 2012
- Acteur majeur de l'eco-système Ruby et Ruby on Rails
- Son souhait pour Elixir :
 - Aussi accessible que Ruby
 - Avec de meilleures performances et une résilience à toute épreuve



Justification du langage

« What kind of business could I build if what before took a hundred servers could today take two servers? I think that's enabling a lot of innovation. »

Chris McCord, créateur du framework Phoenix

Avantages

- **Concurrence** - gérer les tâches simultanées de manière efficace, grâce à son modèle de concurrence basé sur des processus légers.
- **Productivité** - syntaxe claire et concise, qui facilite la lecture et l'écriture du code (inspirée de Ruby et de Python notamment).
- **Robustesse** - Elixir est conçu pour être résilient et tolérant aux pannes. Les applications Elixir sont conçues pour continuer à fonctionner même en cas d'échec d'un processus ou d'une machine.
- **Performance** - conçu pour être hautement performant, grâce à sa capacité à gérer les tâches simultanées et à sa compilation en bytecode Erlang.

Machine virtuelle Erlang



- Elixir est créé sur la VM Erlang "BEAM"
- Développée dans les années 80 par Ericsson
- Afin de gérer les infrastructures téléphoniques grandissantes

- Besoins principaux:
 - Haute concurrence
 - Bonne scalabilité

Pourquoi ne pas utiliser Erlang directement ?

Erlang

```
-module(hello_module).  
-export([some_fun/0, some_fun/1]).  
  
% A "Hello world" function  
some_fun() ->  
    io:format('~s~n', ['Hello world!']).  
  
% This one works only with lists  
some_fun(List) when is_list(List) ->  
    io:format('~s~n', List).  
  
% Non-exported functions are private  
priv() ->  
    secret_info.
```

Influenced by Prolog and Smalltalk

Elixir

```
defmodule HelloModule do  
  # A "Hello world" function  
  def some_fun do  
    IO.puts "Hello world!"  
  end  
  
  # This one works only with lists  
  def some_fun(list) when is_list(list) do  
    IO.inspect list  
  end  
  
  # A private function  
  defp priv do  
    :secret_info  
  end  
end
```

Influenced by Ruby and Clojure

Source: elixirforum.com

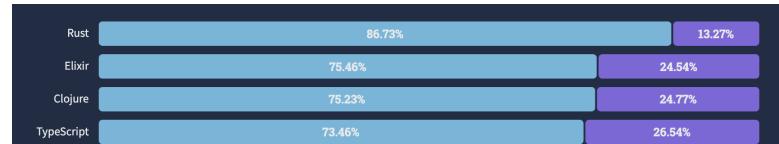
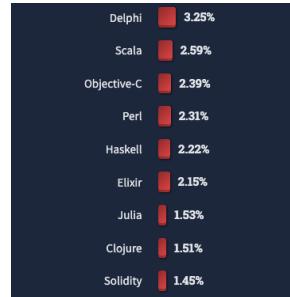
Utilisation

Statistiques basées sur les réponses du sondage de Stack Overflow 2022 (~72k réponses)

Encore peu connu et peu utilisé (**2.15%**)

Mais très apprécié par ceux qui en font.

Fait partie des langages les mieux payés.



Concepts (1)

Dynamiquement typé...

```
iex> x = 1
1
iex> i x
Term
  1
Data type
  Integer
Reference modules
  Integer
Implemented protocols
  IEx.Info, Inspect, List.C chars, String.C chars
```

mais fortement typé !

```
iex> 1 + "2"
(ArithmetricError) bad argument in arithmetic expression: 1 + "2"
```

Concepts (2)

Langage Fonctionnel

TOUTES les données sont immuables

```
iex> x = 1  
1  
iex> x = 2  
2  
iex> x  
2
```

Nous n'avons pas changé la valeur de la variable x, nous avons réassigné le label x à une nouvelle valeur.

Pas de "vraies" boucles

```
iex> Enum.each(1..3, fn x -> IO.puts(x) end)  
1  
2  
3  
:ok
```

Concepts (3)

Process

Bien plus légers que des threads.

```
defmodule Counter do
  use Agent

  def start_link(initial_value) do
    Agent.start_link(fn _ &gt; initial_value end, name: __MODULE__)
  end

  def value do
    Agent.get(__MODULE__, fn x &gt; x end)
  end

  def increment do
    Agent.update(__MODULE__, fn x &gt; x + 1 end)
  end
end
```

```
iex(1)> Counter.start_link(0)
{:ok, #PID<0.114.0>}

iex(2)> Counter.value()
0

iex(3)> Counter.increment()
:ok

iex(4)> Counter.increment()
:ok

iex(5)> Counter.value()
2
```

Concepts (4)

Fail fast / let it crash

Process indépendants, qui se relancent automatiquement en cas d'erreur.

```
iex> File.read("hello")
{:error, :enoent}

iex> File.write("hello", "world")
:ok

iex> File.read("hello")
{:ok, "world"}

# Ex avec pattern matching:
iex> case File.read("hello") do
...>   {:ok, body} -> IO.puts("Success: #{body}")
...>   {:error, reason} -> IO.puts("Error: #{reason}")
...> end
```

Particularités (1)

Documentation

Documentation traitée comme « first-class citizen »

```
iex(1)> h trunc
          def trunc(number)
@spec trunc(number()) :: integer()
```

guard: true

Returns the integer part of number.

Allowed `in` guard tests. Inlined by the compiler.

Examples

```
iex> trunc(5.4)
5
```

```
iex> trunc(-5.99)
-5
```

```
iex> trunc(-5)
-5
```

Particularités (2)

Pattern matching

L'opérateur = réalise derrière les décors un pattern matching.

```
iex> [a, b] = [1, 2]
[1, 2]
```

```
iex> a
1
```

```
iex> b
2
```

```
iex> x = 1
1
```

```
iex> x
1
```

Particularités (3)

Compatibilité avec Erlang

Les modules Erlang sont directement disponibles dans le code Elixir.

```
iex> Base.encode16(:crypto.hash(:sha256, "Elixir"))  
"3315715A7A3AD57428298676C5AE465DADA38D951BDFAC9348A8A31E9C7401CB"
```

Présentation projet

Canvas de dessin collaboratif en temps réel.

- Canvas partagé
- Synchronisation à chaque point et pas à chaque fin de trait
- Tests de montée en charge

