Elixir

Press Space for next page →

Sommaire

- Historique
- Erlang
- Utilisation
- Avantages
- Fonctionnalités
- Particularités

Historique

- Créé par José Valim en 2012
- Acteur majeu 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

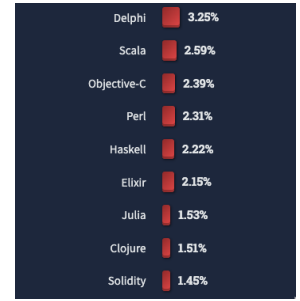
Machine virtuelle Erlang

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

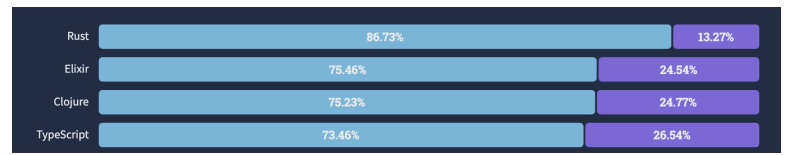
- Besoins principaux:
  - Haute concurrence
  - Bonne scalabilité

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.

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.

Fonctionnalités (1)

Dynamiquement typé…

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

mais fortement typé !

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

Fonctionnalités (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
```

Fonctionnalités (3)

Pattern matching

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

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