Pourquoi Elixir?

•••

ConFoo 2024

Oui allo?

👋 Moi c'est **Marco**

🏢 Directeur des technologies chez **Rum&Code**

💜 Amateur d'Elixir depuis 2020

Rum&Code, c'est qui?

Rum&Code est une agence de développement sur mesure, basée **en Mauricie**.

Plus d'une **vingtaine de projets actifs** dans plusieurs de domaines d'affaires différents:

- Finances
- Éducation
- IoT
- Agriculture
- et bien d'autres

On utilise **Elixir depuis 2020** pour tous nos **nouveaux développements**.

En une slide, pourquoi Elixir?

- Un langage très pragmatique, pour les temps modernes
- **Le 2e langage le plus admiré***
- Un runtime de première classe
- La programmation fonctionnelle, c'est pas juste le futur, **c'est maintenant**

*Selon le StackOverflow Developer Survey de 2023

Elixir, c'est quoi?

Elixir est un langage **fonctionnel et dynamique** pour construire des applications **évolutives et maintenables**.

Elixir roule sur la **VM d'Erlang** (BEAM), un langage reconnu pour créer des **applications distribuées, *low-latency* et *fault-tolerant***.

Erlang, né chez Ericsson en 1989, est utilisé encore aujourd'hui dans des **systèmes de télécommunications** et autres **systèmes temps-réel**.

WhatsApp, bâti sur Erlang

WhatsApp, fondée en 2009, a *scale* à 1**00M d'utilisateurs** en 3 ans, avec **seulement 35 développeurs**.

Acquise par Facebook en 2014, WhatsApp dessert aujourd'hui **plus de 2 milliards d'utilisateurs**. Roule sur Erlang encore à ce jour.

Source: https://thechipletter.substack.com/p/ericsson-to-whatsapp-the-story-of

Qui utilise Elixir?

100M d'utilisateurs actifs mensuellement, Elixir est la colonne vertébrale de leur infrastructure de *chat*.

Traite jusqu'à **1M de requêtes / minute**.

Refonte de leur **service d'analytics**. Traite **~3-4K requêtes / seconde** avec 99% des **temps de réponse entre 0-1ms**.

Web dashboard construit avec Phoenix.

Source: https://elixir-lang.org/cases.html

Qui utilise Elixir?

- Adobe
- Bet365
- CodeSandbox
- FarmBot
- Pinterest
- TheScore
- et plusieurs autres

Source: https://elixir-companies.com/en/companies

Cool, mais montre-moi du code!

— tout le monde, probablement

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    numbers
    |> Enum.map(fn x -> x * x end)
    |> Enum.reduce(fn a, b -> a + b end)
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    squares = Enum.map(numbers, fn x -> x * x end)
    sum = Enum.reduce(squares, fn a, b -> a + b end)

    sum
  end
end
```

```elixir
defmodule Example do
  def sum_of_squares(numbers) do
    numbers
    |> Enum.map(fn x -> x * x end)
    |> Enum.reduce(fn a, b -> a + b end)
  end
end

Example.sum_of_squares(1..10) # => 385
```

```
defmodule Example do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n) when n > 1, do: fib(n - 2) + fib(n - 1)
end
```

```
defmodule Example do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n) when n > 1, do: fib(n - 2) + fib(n - 1)
end
```

```elixir
defmodule Example do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n) when n > 1, do: fib(n - 2) + fib(n - 1)
end

Example.fib(9)
# => 34
```

```elixir
defmodule Example do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n) when n > 1, do: fib(n - 2) + fib(n - 1)
end

Example.fib(9)
# => 34


Example.fib(-1)
# => ** (FunctionClauseError) no function clause matching in Example.fib/1
```

```elixir
defmodule Example do
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(n) when n > 1, do: fib(n - 2) + fib(n - 1)
  def fib(n), do: nil
end

Example.fib(-1)
# => nil
```

```elixir
defmodule Circle do
  defstruct [:radius]
end

defmodule Rectangle do
  defstruct [:width, :height]
end

rectangle = %Rectangle{
  width: 20,
  height: 30
}

circle = %Circle{
  radius: 15
}
```

```
defmodule Circle do
  defstruct [:radius]
end

defmodule Rectangle do
  defstruct [:width, :height]
end

rectangle = %Rectangle{
  width: 20,
  height: 30
}

circle = %Circle{
  radius: 15
}
```

```
defmodule Circle do
  defstruct [:radius]
end

defmodule Rectangle do
  defstruct [:width, :height]
end

rectangle = %Rectangle{
  width: 20,
  height: 30
}

circle = %Circle{
  radius: 15
}
```

```elixir
defmodule Circle do
  defstruct [:radius]
end

defmodule Rectangle do
  defstruct [:width, :height]
end

rectangle = %Rectangle{
  width: 20,
  height: 30
}

circle = %Circle{
  radius: 15
}
```

```elixir
defmodule Geometry do
  def area(%Circle{radius: r}) do
    3.14159 * r * r
  end

  def area(%Rectangle{width: w, height: h}) do
    w * h
  end
end

Geometry.area(circle)
# => 706.85775

Geometry.area(rectangle)
# => 600
```

```
defmodule Geometry do
  def area(%Circle{radius: r}) do
    3.14159 * r * r
  end

  def area(%Rectangle{width: w, height: h}) do
    w * h
  end
end

Geometry.area(circle)
# => 706.85775

Geometry.area(rectangle)
# => 600
```

```elixir
defmodule Geometry do
  def area(%Circle{radius: r}) do
    3.14159 * r * r
  end

  def area(%Rectangle{width: w, height: h}) do
    w * h
  end
end

Geometry.area(circle)
# => 706.85775

Geometry.area(rectangle)
# => 600
```

Elixir, un langage fonctionnel

Fonctionnel = Simple

Transparence référentielle

Mêmes inputs = mêmes outputs

Simple != facile

Elixir, un langage fonctionnel

Patterns OO

- Single Responsibility Principle
- Dependency Inversion
- Factory Pattern
- Strategy Pattern
- Visitor Pattern
- Command Pattern

- Interfaces
- Polymorphisme

Équivalent en fonctionnel

- Fonctions
- Fonctions
- Fonctions, encore
- Des fonctions
- Toujours des fonctions
- Juste des fonctions

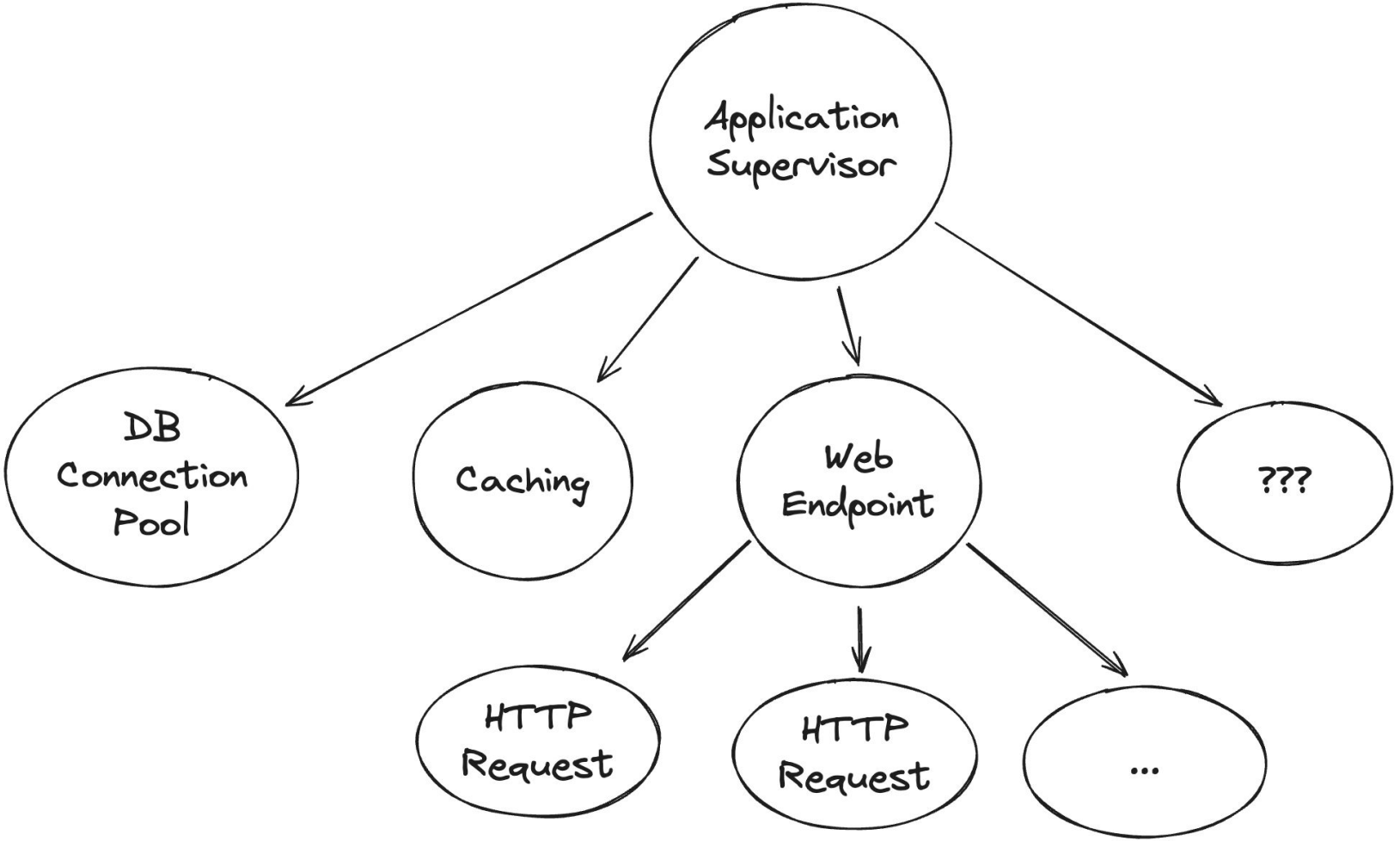
Équivalent en Elixir

- Behaviours
- Protocols

OK, mais encore?!

— les sceptiques dans la salle

Tout est dans le runtime


Application Supervisor
DB Connection Pool
Caching
Web Endpoint
???
HTTP Request
HTTP Request
...

Un runtime *distribué*

```elixir
defmodule WeatherAPI do
  def max_temperature(date) do
    # ... somehow get data from API
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [...]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [
      max_concurrency: 10,
      timeout: 2000,
      on_timeout: :kill_task
    ]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

```elixir
defmodule AsyncExample do
  def max_temperatures(dates) do
    options = [
      max_concurrency: 10,
      timeout: 2000,
      on_timeout: :kill_task
    ]

    dates
    |> Task.async_stream(&WeatherAPI.max_temperature/1, options)
    |> Enum.map(fn
      {:ok, value} -> value
      {:exit, {date, reason}} -> %{date: date, error: reason}
    end)
    |> Enum.to_list()
  end
end
```

Elixir, un langage dynamique*

Plutôt que fortement typé.

Offre une **énorme flexibilité**, mais demande une **très grande rigueur**.

Possibilité d'ajouter des *type specs* dans le code, mais ce n'est pas enforcé par le compilateur. Utile à des fins de *linting* seulement.

***** Un ***type system* graduel** est en cours de développement

Un écosysteme mature

Build tool, package manager?	Mix, Hex.pm
Test automatisés?	ExUnit, ExMachina, Faker
Web framework?	Phoenix + LiveView
L'accès aux données?	Ecto
Job Queue?	Oban
Analyse du code, linter?	Credo, Sobelow
API GraphQL?	Absinthe, Dataloader
Parsers + encoders CSV, XML, JSON, etc	Plusieurs choix
Clients HTTP	Plusieurs choix

Elixir, ça fait quoi d'autre?

Intelligence artificielle:

- Nx (tensors)
- Axon (neural networks)
- Bumblebee (importation de modèles HuggingFace)

Livebook: un notebook interactif à la Jupyter

Applications mobiles avec LiveView Native (en développement)

Finalement, pourquoi Elixir?

Expérience développeur (DX) moderne

Les avantages de la programmation fonctionnelle sont indéniables

Langage très pragmatique, basé dans le monde réel

Simplification de votre stack techno, *embrace the (distributed) monolith*

Une communauté florissante

Elixir est le **2e langage le plus admiré**, juste derrière Rust, selon le StackOverflow Developer Survey de 2023

~23 000 utilisateurs sur elixirforum.com

~28 000 utilisateurs sur /r/elixir

Plusieurs podcasts dédiés à Elixir

Beaucoup de formations disponibles

À votre tour

Merci 💜

Références

- Slides: https://bit.ly/pourquoi-elixir
- Elixir: https://elixir-lang.org/
- Elixir Case Studies: https://elixir-lang.org/cases.html
- Liste des compagnies qui utilisent Elixir: https://elixir-companies.com/en/companies
- L'histoire de Erlang: https://thechipletter.substack.com/p/ericsson-to-whatsapp-the-story-of

Questions?