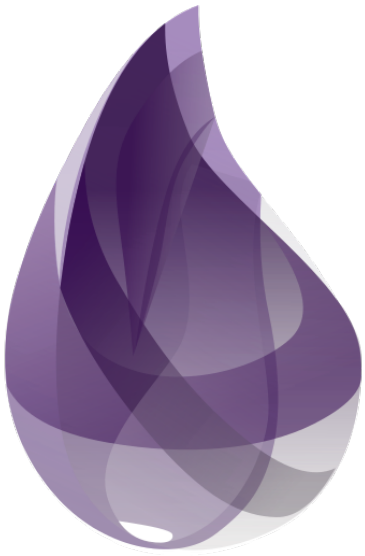
Elixir

LP - Treball Dirigit

Autor: 8216



Índex

1 Introducció

Elixir és un llenguatge de programació funcional orientat a processos dissenyat per crear aplicacions escalables i sostenibles per a sistemes distribuïts en temps real, com ara aplicacions web. Algunes de les aplicacions més conegudes escrites amb Elixir són Discord, Pinterest i Adobe.

El seu principal objectiu és crear sistemes tolerants a errors i de baixa latència, utilitzant una alta concurrència de manera eficient i proporcionant una sintaxi expressiva.

1.1 La seva relació amb Erlang

Erlang és un llenguatge de programació dissenyat per a la creació de sistemes distribuïts en temps real, amb tolerància d'errors i alta escalabilitat. Alguns dels seus usos són en disciplines de telecomunicacions, banca, comerç electrònic, telefonia informàtica i missatgeria instantània [1]. L'aplicació més coneguda escrita amb Erlang és WhatsApp.

Elixir està dissenyat per ser compatible i treballar conjuntament amb Erlang i la seva infraestructura de llibreries. Elixir s'executa en la màquina virtual d'Erlang (BEAM), i per tant té accés a totes les llibreries i eines que suporta, com per exemple la capacitat de mantenir i executar milions de processos concurrents i la flexibilitat de poder actualitzar codi mentre el sistema s'executa sense la necessitat de reiniciar-lo.

A més, Elixir està creat per ser un llenguatge alternatiu per a BEAM, amb la finalitat de proporcionar un codi més net, compacte i que millor reveli les intencions del programador. Amb aquest objectiu, Elixir afegeix algunes característiques addicionals per millorar la usabilitat i la programació.

Per exemple, Elixir té un sistema de macros que permet als programadors definir les seves pròpies construccions del llenguatge. En la següent figura tenim un exemple senzill d'una macro en Elixir que és equivalent a l'expressió *if not*.

```
unless some_expression do
    block_1
else
    block_2
end
```

Listing 1: Exemple de macro en Elixir

A part de les macros, Elixir també fa un gran treball a l'hora de simplificar i reduir codi. Per veure això, contrastarem dos blocs de codi escrits en Erlang i Elixir que implementen un procés de servidor senzill que suma dos nombres.

```erlang
-module(sum_server).

-behaviour(gen_server).
-export([
    start/0, sum/3,
    init/1, handle_call/3, handle_cast/2, handle_info/2,
    terminate/2, code_change/3
]).

start() -> gen_server:start(?MODULE, [], []).
sum(Server, A, B) -> gen_server:call(Server, {sum, A, B}).

init(_) -> {ok, undefined}.

handle_call({sum, A, B}, _From, State) -> {reply, A + B, State};
handle_cast(_Msg, State) -> {noreply, State}.
handle_info(_Info, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_OldVsn, State, _Extra) -> {ok, State}.
```

Listing 2: Procés de servidor basat en Erlang que suma dos nombres

```elixir
defmodule SumServer do
    use GenServer

    def start do
        GenServer.start(__MODULE__, nil)
    end

    def sum(server, a, b) do
        GenServer.call(server, {:sum, a, b})
    end

    def handle_call({:sum, a, b}, _from, state) do
        {:reply, a + b, state}
    end
end
```

Listing 3: Procés de servidor basat en Elixir que suma dos nombres

Podem veure com el bloc de codi en Elixir és significativament més petit i, per tant, més fàcil de llegir i mantenir. La seva intenció és més clara i està menys

carregada de soroll. I, tanmateix, la versió en Elixir és tan capaç i flexible com la d'Erlang, es comporta exactament igual en temps d'execució i conserva completament la semàntica. [2]

Com hem pogut veure, Elixir i Erlang estan estretament relacionats i es complementen entre si. Erlang proporciona una base sòlida per a la programació concurrent i distribuïda, mentre que Elixir afegeix característiques addicionals per millorar la programació i la usabilitat.

2 Història

Elixir va ser creat l'any 2011 per José Valim amb l'objectiu de proporcionar un llenguatge amb codi més net, compacte i entenedor per a la màquina virtual d'Erlang.

Figure 1: Fotografia de José Valim

Va treballar en el llenguatge de programació Ruby i era un membre important de l'equip Ruby on Rails, un framework de desenvolupament web en Ruby que proporciona un patró de disseny, característiques i convencions per a la creació ràpida d'aplicacions web.

Valim va formar part d'aquest equip durant molts anys, però els problemes de concurrència de Ruby el van fer abandonar i va decidir investigar altres llenguatges de programació més enfocats a la concurrència. El seu objectiu era trobar un llenguatge que li permetés escriure un programari elegant i robust que s'executés en diversos nuclis. En aquell moment va ser quan va descobrir Erlang i el seu model de concurrència que el va enamorar [3].

Va dedicar un temps a estudiar Erlang, i com més aprenia més li agradava. A mesura que va anar aprenent més sobre Erlang, a Valim se li van començar a sorgir idees sobre els avantatges que tenia BEAM i algunes possibles millores per a Erlang. Aquest va ser el principal motiu que el va portar a crear Elixir.

2.1 Relació amb LPs similars

La seva experiència prèvia amb altres llenguatges de programació van portar a Valim a crear un llenguatge molt influenciat amb Ruby i Erlang.

Ruby vs. Elixir

Valim va incorporar moltes de les característiques principals de Ruby a Elixir, com per exemple el seu sistema de tipus dinàmic, el paradigma de programació funcional i el fort èmfasi que tenen els dos llenguatges en proporcionar un codi d'alta llegibilitat i expressivitat.

A continuació proporcionem una comparació de l'algorisme d'exponenciació ràpida en els dos llenguatges per a veure en el codi les característiques que hem mencionat.

```elixir
defmodule Math do
  def pow(_, 0), do: 1
  def pow(x, 1), do: x
  def pow(x, n) when rem(n, 2) == 0 do
    pow(x * x, div(n, 2))
  end
  def pow(x, n) do
    x * pow(x * x, div(n - 1, 2))
  end
end
```

Listing 4: Exponenciació ràpida en Elixir

```ruby
def pow(x, n)
  return 1 if n == 0
  return x if n == 1

  if n % 2 == 0
    pow(x * x, n / 2)
  else
    x * pow(x * x, (n - 1) / 2)
  end
end
```

Listing 5: Exponenciació ràpida en Ruby

Com veiem els dos llenguatges tenen una sintaxi i una expressivitat molt similar.

Erlang vs. Elixir

Erlang i Elixir comparteixen una sèrie de similituds, ja que Elixir va ser dissenyat per aprofitar els punts forts d'Erlang i millorar-ne la seva sintaxi i característiques. Algunes d'aquestes similituds són:

- L'ús d'un paradigma de programació funcional, dissenyat per ser altament concurrent i tolerant a errors.
- La utilització d'un model de pas de missatges per a la comunicació entre processos, facilitant la creació de sistemes distribuïts que gestionen grans quantitats de trànsit.
- *Pattern matching*: Permet coincidir i fer *matching* condicions específiques amb elegància perquè s'executi un tros de codi. En l'exemple de l'exponenciació ràpida hem utilitzat *pattern matching* per definir els casos base.

3 Sistema de tipus

Elixir té un sistema de tipus dinàmic i fortament tipat. Això significa que els tipus de les variables es comproven en temps d'execució i no en temps de compilació, per la qual cosa l'usuari no ha de declarar els tipus de les variables quan està programant el codi.

Per altra banda, el tipat fort indica que el llenguatge imposa restriccions sobre les operacions que es poden realitzar entre les variables, i no deixa fer operacions entre variables de tipus diferents sense que es faci una conversió prèvia de tipus.

En el següent bloc de codi veiem un exemple de declaració de variables i operacions entre elles en Elixir:

```
# Declaracions de variables
num = 42          # variable de tipus enter
str = "Hola"      # variable de tipus string
lst = [1, 2, 3]   # variable de tipus llista

# Operacions
res1 = num + 10       # suma d'un enter amb un altre enter
res2 = str <> " món"  # concatenació d'strings
res3 = lst + 5        # error de tipus
```

Com veiem no hem declarat el tipus de les variables en el seu moment de creació, ja que això s'assignarà en temps d'execució.

També trobarem un error de tipus en efectuar la tercera operació, perquè estem usant l'operació suma amb dos variables de tipus diferents: estem intentant sumar una llista amb un enter.

4 Paradigmes de programació

Elixir és un llenguatge multiparadigma, és a dir, suporta múltiples paradigmes de programació per permetre als seus usuaris utilitzar l'estil i les construccions del llenguatge que més s'adeqüin al seu treball.

A continuació donarem una breu introducció als paradigmes més importants que utilitza Elixir:

4.1 Programació funcional

Elixir utilitza el paradigma funcional basat en funcions pures, recursivitat i composició de funcions per crear un codi que és breu, concís i fàcil de mantenir utilitzant estructures de dades immutables, com són les funcions.

Aquí mostrem un exemple de com calcular el factorial d'un nombre utilitzant programació funcional:

```
defmodule Factorial do
  def calculate(0), do: 1
  def calculate(n), do: n * calculate(n - 1)
end
```

Listing 6: Exemple de programació funcional en Elixir

4.2 Programació orientada a processos

El paradigma de programació orientada a processos es basa a crear múltiples processos lleugers i fer que es comuniquin entre ells a través de missatges, amb la finalitat de manejar càrregues de treball concurrents de manera eficient i escalable.

Tot el codi d'Elixir s'executa en processos lleugers que s'aïllen i intercanvien informació mitjançant missatges. D'aquesta manera es poden executar centenars de milers de processos simultàniament en la mateixa màquina i es poden comunicar amb processos que s'executen en màquines diferents per coordinar el treball entre diversos nodes [4].

4.3 Metaprogramació

La metaprogramació és una tècnica de programació avançada mitjançant la qual podem escriure programes que tracten altres programes com les seves dades. La metaprogramació permet als programes llegir, generar i modificar altres programes, i a més, aquests programes es poden modificar a si mateixos de manera significativa [5].

5 Sistema d'execució

Com hem mencionat anteriorment, Elixir fa servir la màquina virtual d'Erlang (BEAM) per a l'execució dels seus programes. Explicarem amb més detall el procés de compilació des d'Elixir fins a fitxer binari.

En primer pas, el codi d'Elixir es compila utilitzant el compilador d'Elixir cap a BEAM bytecode, una representació en baix nivell del codi en Elixir que facilita a BEAM la compilació cap a fitxer binari. Aquest pas genera un fitxer en format *.beam*.

A continuació, aquest fitxer es compila en BEAM i es converteix en un fitxer binari que es pot executar.

A més a més, Elixir disposa d'una *shell* interactiva anomenada iex, que proporciona eines per ajudar a la programació amb Elixir, com per exemple la funció d'autocompletat, la depuració i la recàrrega de codi, i d'entre altres moltes funcions. A continuació, es mostra un exemple de les comandes que es poden executar en iex:

```
$ iex
Interactive Elixir - press Ctrl+C to exit (type h() ENTER for help)
iex> h String.trim            # Prints the documentation
iex> i "Hello, World"         # Prints information about a data type
iex> break! String.trim/1     # Sets a breakpoint
iex> recompile                # Recompiles the current project
```

Listing 7: Exemple de comandes en iex [4]

6 Visió personal i crítica

Des del meu punt de vista, considero Elixir un llenguatge de programació molt adequat per als desenvolupadors que busquen crear pàgines web amb l'objectiu d'obtenir una gran quantitat d'usuaris i que consideren l'escalabilitat un factor important.

A més, el llenguatge té una gran expressivitat i inclou una sintaxi senzilla que, juntament amb el tipat dinàmic i la programació funcional, simplifiquen la tasca als programadors a l'hora de generar codi. Això dona lloc a què els programadors vulguin utilitzar Elixir per la simplicitat que ofereix el llenguatge.

Elixir també proporciona moltes eines als seus usuaris perquè la programació sigui més eficient i productiva, com per exemple l'ús de Mix: una eina que ofereix tasques per crear, compilar i provar aplicacions creades amb Elixir i també gestionar les seves dependències i molt més. [6]

7 Avaluació de les fonts

Les fonts d'informació utilitzades per aquest treball són principalment les pàgines web oficials dels llenguatges de programació d'Elixir i Erlang, així com un llibre de programació sobre Elixir. Per a conceptes més específics i dubtes puntuals, s'han consultat les pàgines corresponents a Wikipedia i altres fonts de tercers, i pel que fa a la història d'Elixir, s'ha obtingut la informació a partir d'un pòdcast que es va fer a José Valim.

S'ha donat preferència a la informació de les fonts oficials d'Elixir i Erlang, ja que són les que considero més fiables. En canvi, la informació procedent de Wikipedia i d'altres fonts de tercers s'ha verificat amb la informació que proporcionen les fonts oficials, perquè considero que no són gaire dignes de confiança. Respecte a la informació extreta del llibre *Elixir in Action* i el pòdcast, crec que la informació que ofereixen és fiable i, per tant, els he considerat una font segura d'informació.

Per redactar la secció de la introducció d'aquest treball he consultat la pàgina web oficial d'Elixir [4] i una pàgina de tercers on fan una introducció a Elixir [7] per obtenir tota la informació necessària, i per l'apartat "La seva relació amb Erlang" he usat la informació que proporciona el llibre *Elixir in Action* [2] i la pàgina web oficial d'Erlang [1].

Per la història d'Elixir he consultat l'entrevista que van fer a José Valim en un pòdcast [3], i per aconseguir informació sobre les característiques dels llenguatges de Ruby i Erlang vaig consultar les seves pàgines oficials [1] [8].

Per descobrir quin sistema de tipus utilitza Elixir vaig decidir buscar-ho en la seva pàgina de Wikipedia [9], i una vegada descobert el seu tipus vaig cercar més informació sobre el sistema de tipus dinàmic i el sistema de tipat fort utilitzant les seves pàgines corresponents de Wikipedia [10] [11].

Una vegada vaig obtenir tota la informació, vaig decidir aprendre a usar operacions amb tipus bàsics en Elixir per poder escriure un codi representatiu que expliqués bé el seu sistema de tipus [7].

Vaig obtenir informació sobre els paradigmes de programació d'Elixir mitjançant el llibre *Elixir in Action* [2], la pàgina oficial d'Elixir [4] i la pàgina de Wikipèdia d'Elixir [9] i de cada paradigma tractat [12] [13] [14]. Per l'apartat de Metaprogramació vaig haver de consultar una altra font per a obtenir informació més clara i precisa [5].

Finalment, per obtenir informació sobre el sistema d'execució en Elixir vaig consultar la font oficial del llenguatge [4], la font oficial de l'intèrpret [15] i de Mix [6].

8 Bibliografia i Webgrafia

References

[1] Index - Erlang/OTP — erlang.org. https://www.erlang.org/. [Accessed 4-May-2023].

[2] Sasa Juric. *Elixir in Action*. Manning, 2015.

[3] Remote Ruby — José Valim, creator of Elixir and form Rails core contributor — remoteruby.com. https://remoteruby.com/178. [Accessed 4-May-2023].

[4] elixir-lang.github.com — elixir-lang.org. https://elixir-lang.org/. [Accessed 2-May-2023].

[5] An introduction to metaprogramming with Elixir — educative.io. https://www.educative.io/blog/introduction-metaprogramming-elixir. [Accessed 6-May-2023].

[6] Mix x2014; Mix v1.14.4 — hexdocs.pm. https://hexdocs.pm/mix/1.14/Mix.html. [Accessed 8-May-2023].

[7] A beginner's guide to the Elixir programming language — educative.io. https://www.educative.io/blog/elixir-functional-programming. [Accessed 6-May-2023].

[8] Documentation — ruby-lang.org. https://www.ruby-lang.org/en/documentation/. [Accessed 7-May-2023].

[9] Elixir (programming language) - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Elixir$_{(}programming_{l}anguage$).[$Accessed4-May-2023$].

[10] Strong and weak typing - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Strong$_{a}nd_{w}eak_{t}yping$. [$Accessed7-May-2023$].

[11] Dynamic programming language - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Dynamic$_{p}rogramming_{l}anguage$.[$Accessed7-May-2023$].

[12] Functional programming - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Functional$_{p}rogramming$. [$Accessed8-May-2023$].

[13] Process-oriented programming - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Process-oriented$_{p}rogramming$.[$Accessed8-May-2023$].

[14] Metaprogramming - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Metaprogramming. [Accessed 8-May-2023].

[15] IEx x2014; IEx v1.14.4 — hexdocs.pm. https://hexdocs.pm/iex/IEx.html. [Accessed 8-May-2023].