The Elixir Programming Language
By Zorawar Moolenaar

Elixir was released in 2011 by José Valim.

Background

- Originated in the Industry:
    - Valim was part of the Ruby on Rails (RoR) core team;
    - began working on Elixir after departing from RoR while trying to solve high-throughput server concurrency issues (Ruby is notoriously bad with this, switched to Erlang, somewhere after Elixir was born)
    - Inspired by Erlang, Ruby, Clojure and LISP
- Wields the power of the 33 year old mature Erlang Ecosystem:
    - Compiles to the Erlang Virtual Machine
    - Compatible with the Erlang Open Telecom Platform
    - Zero performance overhead in calling Erlang Libraries

"Elixir brings ruby-inspired 'non-scary' syntax and a load of other goodies to Erlang." - Joe Armstrong, co-creator of Erlang

The Elixir Programming Language

- Functional-style Concurrent Programming Language
    - "Real Object Oriented Language"
- Strong but Dynamic typing discipline
- General Purpose Programming Language with metaprogramming
    - Features LISP style macros to create Domain Specific Languages
- Key Abstractions
    - Data Types
    - Modules
    - Processes
- Functional Paradigm
    - Immutable Data Types
    - Pattern Matching
    - Tail recursion; no loops
    - Guards
- Concurrent, Distributed and Fault Tolerant
    - Uses the *Actor Model* for Concurrency. In Erlang/Elixir, Processes are modeled as Actors.
    - Processes are very lightweight, efficient and strongly isolated.

    - Can write massively scalable, fault tolerant applications
    - Hot Code Swapping -- Upgrade system while it is running
- "Write Once, Run Forever"
    - High Availability --- For uninterrupted Services
    - Built for real time and consistent performance
- Modern Tools
    - Excellent build tool, package manager
    - Documentation is first-class entity
    - Mix of compiled and interpreted: run as binary or script
- Learning Curve:
    - Meant to afford a modern programming experience
    - Small, new language --- easy to pick up in a few days
    - Elixir itself is easy, but some (distributed) concepts may take time

Elixir Design Goals

- **Extend** the Erlang Programming Experience
    - Introduces Data Type Polymorphism / Dynamic Dispatch
    - Lazy Streams
    - UTF-8 Strings
    - Data Pipelining
- Increase Developer **Productivity**
    - Modern Programming Experience
        - More Intuitive Syntax
        - Excellent Build/documentation/interaction tooling
    - Creates simple abstractions for common tasks
- Be **Compatible** with the Erlang VM and the existing ecosystem
    - Embraces and Extends Erlang/OTP without breaking it
    - Zero Performance Overhead converting to or from Erlang code

Notable Uses

- Of the Erlang VM (that powers Elixir)
    - WhatsApp was able to **serve 1 Billion Users with 10 employees**
    - **90% of Internet traffic** routes through Erlang controlled nodes
    - Top 100 Service Providers including T-Mobile and AT&T use Erlang
    - Powers GPRS, 3G, LTE
- Of Elixir
    - Discord was able to serve more than **5M concurrent users**
    - Massachusetts Bay Transport Authority saw a **300% speedup** in its API rewritten in Elixir

- Pinterest saw a **200% speedup** with Elixir over its Java app responsible for 14,000 notifications p.s., running on merely 15 servers
- Bleachers Report rewrote its server using Elixir. Elixir version with 5 servers outperformed Ruby on Rails App with 150 servers. **96% resources dropped**.

Fun Facts from the Stack Overflow 2019 Survey

- 28th most popular language
- 8th most loved Language
- 21st Most Wanted Language
- 9th highest average salary in the US: $123k p.a.

Marco-Polo Call Response (Inter Process Communication)

```
defmodule MarcoPolo do
  @moduledoc """
  Demonstrates message passing between processes via a Marco-Polo
call-response
  """

  @doc """
  create a new MarcoPolo server!
  """
  def new_link do
    spawn(MarcoPolo, :start, [])
  end

  @doc """
  start listening for marco or polo messages
  """
  def start do
      listenForMessages()
  end

  @doc """
  Listen for marco or polo messages, and respond appropriately
  """
  def listenForMessages() do
    receive do
```

```
        {"marco", from} ->
            IO.puts "Marco ---> "
            :timer.sleep 500
            send(from, {"polo", self()})
        {"polo", from} ->
            IO.puts "          <--- Polo"
            :timer.sleep 500
            send(from, {"marco", self()})
    end
    listenForMessages()
  end
end
```

Waterways Example (Distributed State Model via stack-based Agent)

File 01: lib/waterways.ex

```
defmodule Waterways do
  @moduledoc """
  This module simulates waterways that connects port cities.
  """

  defstruct [:from, :to]

  @doc """
  Spawn a port city process by the name of `portCity`
  """
  def new_port(portCity) do
    Waterways.Port.start_link(portCity)
  end

  @doc """
  Create a link between two port cities: `from` -> `to`
  `data` is buffered in `from` port city,
  ready to be deployed to `to` port city
  """
  def open_channel(from, to, data) do
    Waterways.Port.push_all(from, data)

    %Waterways{from: from, to: to}
  end
```

```
  @doc """
  Flush/Deploy a single value between the waterway link
  """
  def flush_once(waterway) do
    case Waterways.Port.pop(waterway.from) do
      :error -> :ok
      {:ok, poppedValue} -> Waterways.Port.push(waterway.to,
poppedValue)
    end
    IO.puts "#{waterway.from} has
#{Waterways.Port.get(waterway.from) |> Enum.join(" ")}"
    IO.puts "#{waterway.to} has #{Waterways.Port.get(waterway.to)
|> Enum.join(" ")}"
  end

  @doc """
  Get the information in the `portCity`
  """
  def port_info(portCity), do: Waterways.Port.get(portCity)
end
```

File 02: lib/waterways/port.ex

```
defmodule Waterways.Port do
  @moduledoc """
  Create a Port City that can store elements
  in a list!
  """

  @doc """
  Spawn a port city process by the name of `portCity`
  """
  def start_link(portCity) do
    Agent.start_link(fn -> [] end, name: portCity)
  end

  @doc """
  Get the information in the `portCity`
  """
  def get(portCity) do
    Agent.get(portCity, fn list -> list end)
```

```
  end

  @doc """
  Add elements of `list` to the `portCity`
  """
  def push_all(portCity, list) do
    # Enum.each(Enum.reverse(list), fn x ->
Waterways.Port.push(portCity, x) end)
    list |> Enum.reverse |> Enum.each(fn x ->
Waterways.Port.push(portCity, x) end)
  end

  @doc """
  Add a single `value` to the `portCity`
  """
  def push(portCity, value) do
    Agent.update(portCity, fn list -> [value|list] end)
  end

  @doc """
  Remove the most recent value.
  ## Returns
  the value removed
  """
  def pop(portCity) do
    Agent.get_and_update(portCity, fn
      []    -> {:error, []}
      [x|xs] -> {{:ok, x}, xs}
    end)
  end

end
```

File 03: lib/waterways/application.ex

```
defmodule Waterways.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application
```

```
  def start(_type, _args) do
    # List all child processes to be supervised
    children = [
      # Starts a worker by calling:
Waterways.Worker.start_link(arg)
      # {Waterways.Worker, arg}
    ]

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: Waterways.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Adding Fault Tolerance to Waterways

File 01: lib/waterways.ex (diff)

```
defmodule Waterways do
  @moduledoc """
  This module simulates waterways that connects port cities.
  """

  defstruct [:from, :to]

  @doc """
  Spawn a port city process by the name of `portCity`
  """
  def new_port(portCity) do
    Supervisor.start_child(Waterways.Supervisor, [port])
  end

  @doc """
  Create a link between two port cities: `from` -> `to`
  `data` is buffered in `from` port city,
  ready to be deployed to `to` port city
  """
  def open_channel(from, to, data) do
    Waterways.Port.push_all(from, data)
```

```
    %Waterways{from: from, to: to}
  end

  @doc """
  Flush/Deploy a single value between the waterway link
  """
  def flush_once(waterway) do
    case Waterways.Port.pop(waterway.from) do
      :error -> :ok
      {:ok, poppedValue} -> Waterways.Port.push(waterway.to,
poppedValue)
    end
    IO.puts "#{waterway.from} has
#{Waterways.Port.get(waterway.from) |> Enum.join(" ")}"
    IO.puts "#{waterway.to} has #{Waterways.Port.get(waterway.to)
|> Enum.join(" ")}"
  end

  @doc """
  Get the information in the `portCity`
  """
  def port_info(portCity), do: Waterways.Port.get(portCity)
end
```

File 03: lib/waterways/application.ex (diff)

```
defmodule Waterways.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  def start(_type, _args) do
    import Supervisor.Spec, warn: false
    children = [
      worker(Waterways.Port, [])
    ]
    opts = [strategy: :simple_one_for_one, name:
Waterways.Supervisor]
```

```
    Supervisor.start_link(children, opts)
  end
end
```

References

Aaron Lebo. The UNIX Philosophy and Elixir as an Alternative to Go. 22 June, 2015, http://lebo.io/2015/06/22/the-unix-philosophy-and-elixir-as-an-alternative-to-go.html

Alex et al. Elixir Koans. https://github.com/elixirkoans/elixir-koans

Brian Cardarella. "Elixir: The only Sane Choice in an Insane World." GOTO 2017. https://www.youtube.com/watch?v=gom6nEvtl3U

Discord Engineering. How Discord Scaled Elixir to 5,000,000 Concurrent Users. https://blog.discordapp.com/scaling-elixir-f9b8e1e7c29b

Exercism Elixir Track. https://github.com/exercism/elixir

Gil Barbara. Erlang Logo. Svgporn.com, https://cdn.svgporn.com/logos/erlang.svg

Guilherme Carvalho. "Basically: Erlang Powers the Internet. #CodeBEAMSTO Pic.twitter.com/SnLymkSiAD". Twitter, 1 June 2018, https://twitter.com/guieevc/status/1002494428748140544

Hex Docs. https://hexdocs.pm/elixir/

Joe Armstrong. First impressions. Elixir Forum, 12 September 2018, https://elixirforum.com/t/learning-elixir-frst-impressions-plz-dont-kill-me/16424/52

Joe Armstrong. Inside Erlang - Creator Joe Armstrong Tells His Story. Ericsson.com, 4 December 2014, https://www.ericsson.com/en/news/2014/12/inside-erlang--creator-joe-armstrong-tells-his-story

Joe Armstrong. One Week with Elixir. Github.io, 31 May 2013, https://joearms.github.io/published/2013-05-31-a-week-with-elixir.html

José Valim. Elixir Design Goals.Elixir-lang.org, 08 August 2013, https://elixir-lang.org/blog/2013/08/08/elixir-design-goals/

Jose Valim. Elixir. How I Start, 13 March 2017, https://howistart.org/posts/elixir/1/

José Valim. Official Elixir Logo. Wikimedia Commons, 9 October 2015, https://commons.wikimedia.org/wiki/File:Official_Elixir_logo.png

Jusabe Guedes. Elixir for Java Developers Series. Medium.com, 23 May 2017, https://medium.com/skyhub-labs/elixir-for-java-developers-episode-i-66b65c862652

omgneering. Elixir Supervisor. YouTube.com, 20 December 2016, https://www.youtube.com/watch?v=Gdf8JXeaPjw

The Elixir Language. https://elixir-lang.org/

Иван Сергеев. Thoughts On Elixir. https://habr.com/en/post/449522/