International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 3 | ISSN : 2456-3307

Elixir: A Functional Programming

Sourabh Dadhich, Varun Dadhich, Avishek Kumar

Information Technology, MCKV institute of engineering, Howrah, West Bengal, India

ABSTRACT

The rise of a new programming language provides the need to contrast its contribution in field of programming. The functional programming paradigm was explicitly created to support a pure functional approach to problem solving. Functional programming promotes a coding style that helps developers write code that is short, fast, and maintainable. Functional programming is a form of declarative programming .In this article, we are going to show the advantages and disadvantages of a particular functional programming called Elixir .We compare the language Elixir with different languages genres. Eg . Procedural, OOP.

Keywords: Functional Programming, Erlang, Tuples, Modules

I. INTRODUCTION

Elixir is the functional programming language created by Jose valim during, an R&D project of Plataformatec and it was presented in 2011. Elixir is great for writing highly concurrent web applications because it supports meta programming via macros polymorphism via protocols Elixir is a dynamic, functional language designed for building scalable and maintainable applications. Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain. Elixir was launched to offer an alternative to

Erlang because this has features which commonly evolved from it was created and has a greater difference with procedural and object-oriented programming. However, prominent programming languages which support functional programming such as Common Lisp, Scheme, Clojure, Wolfram Language (also known as Mathematica), Racket, Erlang, OCaml, Haskell, and F# have been used in industrial and commercial applications by a wide variety of organizations. Elixir is inspired as per the other functional programming languages as mentioned earlier. The elixir's syntax is quite different from other because it does not support pointers to such an extent. For instance, In elixir, a function can be passed as an argument to another function but in programming language like C/C++, a function pointer referring to a function can be passed to a function that can invoke the external function through dereference. That's why function pointer have a lot of limitation. Due to these facts, it is necessary to a study about elixir to check if elixir can make a place in field of functional programming.

II. LANGUAGE CHARACTERISTICS

Platform features:

A. Scalability

All Elixir code runs inside lightweight threads of execution (called processes) that are isolated and exchange information via messages:

```
Current_process = self()
# Spawn an Elixir process (not an operating system one!)
spawn_link(fn ->
  send current_process, {:msg, "hello world"}
end)
# Block until the message is received
```

```
receive do
  {:msg, contents} -> IO.puts contents
End
```

Due to their lightweight nature, it is not uncommon to have hundreds of thousands of processes running concurrently in the same machine. Isolation allows processes to be garbage collected independently, reducing system-wide pauses, and using all machine resources as efficiently as possible (vertical scaling).Processes are also able to communicate with other processes running on different machines in the same network. This provides the foundation for distribution, allowing developers to coordinate work across multiple nodes (horizontal scaling).

B. Fault-tolerance

The unavoidable truth about software running in production is that things will go wrong .Even more when we take network, file systems, and other third-party resources into account. To cope with failures, Elixir provides supervisors which describe how to restart parts of your system when things go awry, going back to a known initial state that is guaranteed to work:

```
1) import Supervisor.Spec
2) children = [
3)   supervisor(TCP.Pool, []),
4)   worker(TCP.Acceptor, [4040])
5) ]
6)
7) Supervisor.start_link(children,
strategy: :one_for_one)
```

Language feature

A. Functional programming

Functional programming promotes a coding style that helps developers write code that is short, fast, and maintainable. For example, pattern matching allows developers to easily destructure data and access its contents:

```
%User{name: name, age: age} = User.get("John Doe")
name #=> "John Doe"
```

When mixed with guards, pattern matching allows us to elegantly match and assert specific conditions for some code to execute:

```
8) def serve_drinks(%User{age: age}) when
age >= 21 do
9)   # Code that serves drinks!
10) end
11) serve_drinks User.get("John Doe")
12) #=> Fails if the user is under 21
```

Tooling features

A. A growing ecosystem

Elixir ships with a great set of tools to ease development. Mix is a build tool that allows you to easily create projects, manage tasks, run tests and more:

```
$ mix new my_app
$ cd my_app
$ mix test
Finished in 0.04 seconds (0.04s on load,
0.00s on tests)
1 tests, 0 failures
```

B. Erlang compatible

Elixir runs on the Erlang VM giving developers complete access to Erlang's ecosystem, used by companies like Heroku, WhatsApp, Klarna, Basho and many more to build distributed, fault-tolerant applications. An Elixir programmer can invoke any Erlang function with no runtime cost:

```
iex> :crypto.hash(:md5, "Using crypto from
Erlang OTP")
<<192, 223, 75, 115, ...>>
```

III. WORKING WITH ELIXIR

A. Running code

Elixir has an interactive shell called iex. Compiling Elixir code can be done with elixirc(which is similar to Erlang's erlc). Elixir also provides an executable named elixir to run Elixir code. The module defined above can be written in Elixir as:

```
# module_name.ex
defmodule ModuleName do
```

```
  def hello do
    IO.puts "Hello World"
  end
end
```

However notice that in Elixir you don't need to create a file only to create a new module, Elixir modules can be defined directly in the shell.

In Elixir, expressions are delimited by a line break or a semicolon ;.

B. Variable names

Elixir allows you to assign to a variable more than once. If you want to match against the value of a previously assigned variable, you should use ^:

```
iex> a = 1
1
iex> a = 2
2
iex> ^a = 3
** (MatchError) no match of right hand side value: 3
```

C. Data types

In Erlang, an atom is any identifier that starts with a lowercase letter, e.g. ok, tuple, donut. Identifiers that start with a capital letter are always treated as variable names. Elixir, on the other hand, uses the former for naming variables, and the latter are treated as atom aliases. Atoms in Elixir always start with a colon :.

```
:im_an_atom
:me_too
im_a_var
x = 10
13)Module  # this is called an atom alias; it expands to :'Elixir.Module'
```

D. String

In Erlang, the word **string** means a UTF-8 binary and there is a String module that works on such data. Elixir also expects your source files to be UTF-8 encoded. On the other hand, **string** in Erlang refers to char lists and there is a :string module, that's not UTF-8 aware and works mostly with char lists.Elixir also supports multiline strings (also called *heredocs*):

```
is_binary """
This is a binary
spanning several
lines.
"""
#=> true
```

E. Modules

Here we create a module named `hello_module`. In it we define three functions, the first two are made available for other modules to call via the `export` directive at the top. It contains a list of functions, each of which is written in the format `<function name>/<arity>`. Arity stands for the number of arguments.

```
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
```

F. First-class functions

Anonymous functions are first-class values, so they can be passed as arguments to other functions and also can serve as a return value. There is a special syntax to allow named functions be treated in the same manner.

```
defmodule Math do
  def square(x) do
    x * x
  end
end
```

G. Control flow

The `case` construct provides control flow based purely on pattern matching.

```
case {x, y} do
  {:a, :b} -> :ok
  {:b, :c} -> :good
  other -> other
end
```

If:

```
test_fun = fn(x) ->
  cond do
    x > 10 ->
      :greater_than_ten
    x < 10 and x > 0 ->
      :less_than_ten_positive
    x < 0 or x === 0 ->
      :zero_or_negative
    true ->
      :exactly_ten
  end
end
```

H. Sending and receiving messages

```
pid = Kernel.self
send pid, {:hello}
receive do
  {:hello} -> :ok
  other -> other
after
  10 -> :timeout
end
```

IV. CONCLUSION

Elixir provides far better handling of UTF-8 strings out of the box. It has improvements over OTP such as Agents and GenEvent, and it has meta programming /macros which is huge. Elixir has a better ecosystem of tools and will continue to add features that improve programmer productivity because that is a primary design goal. It overcame all the demerits present in Erlang. A lot of multinational tech companies are working on elixir.eg. pinterest, 22cans, MOZ puppet,discord, thoughtbot. Apart from this, Whatsapp and facebook messenger works on elixir. So, definitely there is not any question regarding the future of this language.

V. REFERENCES

[1]. Simon St. Laurent & J. David Eisenberg, “Atoms, Tuples, and Pattern Matching” in Introducing Elixir, 1st ed. Sebastopol, USA.

[2]. http://elixir-lang.org

[3]. https://en.wikipedia.org/wiki/Elixir_(programming_language)

[4]. Davyd Thomas, Programming Elixir: 1.3 , ISBN 1680500538

[5]. https://github.com/sger/ElixirBooks

[6]. Benjamin Tan Wei Hao, The Little Elixir & OTP Guidebook 1st Edition, Manning Publications, ISBN 9781633430112.

[7]. Kenny Ballou, ”tuples and modules” in Learning Elixir vol.1 , pp,27-44 ,Dec-2015.

[8]. http://alexott.net/en/fp/books/

[9]. https://github.com/chrismccord/elixirexpress/blob/master/basics/03basics.md

[10]. http://www.sitepoint.com/functional-programming-pure-functions/