An Introduction to Erlang

Alejandro Gómez Londoño

Universidad EAFIT

8th March, 2013

What’s Erlang?

Erlang is a concurrent functional programming language, designed for writing concurrent programs that “run forever.”

History

- Developed in Ericsson Computer Science Laboratory (1986)

History

"Solve Ericsson's software problem." [1]

SPOTS project

It was an initiative to find better ways to write telecom software under some requirements like...

- Very large number of concurrent activities
- Real time requirements
- Fault tolerance

[1] J. Armstrong. A history of Erlang. In Proceedings of the third ACM SIGPLAN conference on History of programming languages, HOPL III, pages 6–1–6–26, New York, NY, USA, 2007. ACM.

History
SPOTS project

Make the phones ring

To do so, they wrote POTS[1] in almost every programming language available and some of the conclusions that came out were:

- "Small languages were thought desirable"[2]
- "Functional programming was liked"
- "Logic programming was best in terms of elegance"
- Concurrency was essential

[1] Plain Ordinary Telephone Service.

[2] J. Armstrong. A history of Erlang. In Proceedings of the third ACM SIGPLAN conference on History of programming languages, HOPL III, pages 6–1–6–26, New York, NY, USA, 2007. ACM.

History

- ✓ Developed in Ericsson Computer Science Laboratory (1986)
- It was first implemented in Prolog

History
Early experiments

Joe Armstrong was working in his "POTS"smalltalk implementation alongside developing a graphical notation for defining his programs.

A certain day he showed his work to a colleague only to find him saying “but that’s a Prolog program.”[1]

Joe quickly adapted his work to Prolog and threw away his smalltalk stuff.

[1] J. Armstrong. A history of Erlang. In Proceedings of the third ACM SIGPLAN conference on History of programming languages, HOPL III, pages 6–1–6–26, New York, NY, USA, 2007. ACM.

History

- ✓ Developed in Ericsson Computer Science Laboratory (1986)
- ✓ It was first implemented in Prolog
- Open Source Erlang was released in 1998

History
Erlang becomes open source

After Ericson banned the project in 1998, the development team obtained the approval to release Erlang at the end of the year.

Most of the Erlang development team left Ericsson, and started a new company called "Blue tail"for product development using Erlang.

History

- ✓ Developed in Ericsson Computer Science Laboratory (1986)
- ✓ It was first implemented in Prolog
- ✓ Open Source Erlang was released in 1998

Main features

- Concurrency
- Fault tolerance
- Garbage collection
- Functional

Suitable for...

- Large software for server use
- Higher-level protocol implementation
- Soft real-time systems
- Banking, e-commerce, computer telephony and instant messaging systems are also a good choice

Non suitable for...

- Number crunching
- Low level drivers
- Image processing

Basic data types

Numbers

```
1> 1 + 2.0.
3.0
2> 2/3.
0.6666666666666666
3> 2 div 3.
0
4> 2*3+1.
7
5> 2*(3+1).
8
```

Basic data types

Variables

```
1> Var.
* 1: variable 'Var' is unbound
2> Var = 14.
14
3> Var.
14
4> Var + 1.
15
5> Var = 34.
** exception error: no match of
right hand side value 34
```

Basic data types

Atoms

```
1> am_an_atom.
am_an_atom
2> is_atom(am).
true
3> Var=atom_1.
atom_1
4> is_atom(Var_1).
* 1: variable 'Var_1' is unbound
5> is_atom(Var).
true
```

Basic data types

Boolean Algebra & Comparison operators

```
1> true and false.
false
2> true or
2> (false and true).
true
3> true or
3> false and true.
true
4> false and true or true.
true
```

```
5> 49 == 49.0.
true
6> 49 =:= 49.0.
false
7> 49 =/= 49.1.
true
8> 49 /= 49.0.
false
```

Basic data types

Tuples

```
1> { 15 , the_atom }.
{15,the_atom}
2> X = { 15 , the_atom }.
15,the_atom
3> { A, _ } = X.
{15,the_atom}
4> A.
15
```

Basic data types

Lists

```
1> [1,this,is,madness,2,3,4].
[1,this,is,madness,2,3,4]
2> [X|Y] = [1,3,4,5,6,7].
[1,3,4,5,6,7]
3> X.
1
4> Y.
[3,4,5,6,7]
```

Basic data types

Bit Syntax

```
0> <<Sign:1,Exp:8,Man:23>>=<<255,255,255,0>>.
<<255,255,255,0>>
1> Sign.
1
2> Man.
8388352
3> Exp.
255
```

Functions

Pattern matching

```
    fibonacci(0) ->
        0;
    fibonacci(1) ->
        1;
    fibonacci(X) ->
        fibonacci(X-1) + fibonacci(X-2).
```

Functions

Pattern matching (Lists)

```
head([X|_]) ->
    X.

length([]) ->
    0;
length([_|Y]) ->
    1 + length(Y).
```

Functions

Guards

```
what_is(X) when is_atom(X) ->
    "is an atom";
what_is(X) when is_list(X) ->
    "is a list";
what_is(X) when is_integer(X) ->
    "is a number".
```

Functions

Anonymous functions

```
fun(Args1) ->
     Expression1, Exp2, ..., ExpN;
    (Args2) ->
     Expression1, Exp2, ..., ExpN;
    (Args3) ->
     Expression1, Exp2, ..., ExpN
    end
```

Other features

- Call by value
- Strong typing
- Dynamic typing
- High order functions

Concurrency

- “Is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other”
- “For many Erlangers, concurrency refers to the idea of having many actors running independently, but not necessarily all at the same time”[1]

[1]F. Hèbert. Learn You Some Erlang for Great Good!: A Beginner’s Guide. No Starch Press Series. No Starch Press, 2013.

Actor model

"Erlang uses the actor model, and each actor is a separate process in the virtual machine. In a nutshell, if you were an actor in Erlang's world, you would be a lonely person, sitting in a dark room with no window, waiting by your mailbox to get a message"

"Erlang's actor model can be imagined as a world where everyone is sitting alone in their own room and can perform a few distinct tasks. Everyone communicates strictly by writing letters and that's it."[1]

[1] F. Hèbert. Learn You Some Erlang for Great Good!: A Beginner's Guide. No Starch Press Series. No Starch Press, 2013.

Some Code[1]

```
go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        Pid2, Msg ->
            io:format("P1 ~w~n",[Msg])
    end,
    Pid2 ! stop.
```

[1]F. Hèbert. Learn You Some Erlang for Great Good!: A Beginner's Guide. No Starch Press Series. No Starch Press, 2013.

Some Code[1]

```
loop() ->
    receive
        {From, Msg} ->
            From ! self(), Msg,
            loop();
        stop ->
            true
    end.
```

[1] F. Hèbert. Learn You Some Erlang for Great Good!: A Beginner's Guide. No Starch Press Series. No Starch Press, 2013.

Research and cool stuff

- Use of Prolog for developing a new programming language[1]
- A practical subtyping system for Erlang [2]
- Practical Type Inference Based on Success Typings [3]
- HiPe

[1] J. L. Armstrong, S. Virding, and M. C. Williams. Use of Prolog for developing a new programming language, 1992.

[2] S. Marlow and P. Wadler. A practical subtyping system for Erlang. In In Proceedings of the International Conference on Functional Programming (ICFP '97, pages 136–149. ACM Press, 1997.

[3] T. Lindahl and K. Sagonas. Practical type inference based on success typings. In Proceedings of the 8th ACM SIGPLAN international conference on Principles and practice of declarative programming, PPDP '06, pages 167–178, New York, NY, USA, 2006. ACM.