Erlang 101

R Team Workshop

David Duque Fuertes

https://github.com/duqueGZ/erlang_101

Contenidos

- Why Erlang?
- Introducción a Erlang
- Programación Secuencial
- Excepciones
- Programación Secuencial Avanzada
- Erlang en funcionamiento

Why Erlang?

- **Built to kick ass**
  - scalable, fault-tolerant, distributed, non-stop, soft-realtime.
- **Battle-proven**
- **Saves time & money**
- **Easy to learn**
- **Other awesome features**
  - lightweight concurrency, transparent distribution, hot code upgrades, OTP…

http://veldstra.org/whyerlang/

Introducción a Erlang

- Lenguaje de programación concurrente
- El subcojunto secuencial: lenguaje funcional con evaluación estricta, asignación única y tipado dinámico
- Originalmente, lenguaje propietario de Ericsson. Desde 1998, software de código abierto
- A. K. Erlang / ERicsson LANGuage

Introducción a Erlang

- **Instalación: http://www.erlang.org/downloads**
- **La Shell de Erlang (erl)**
  - Expresiones aritméticas (enteros y punto flotante)
  - Variables
  - Atoms
  - Strings
  - Tuplas
  - Listas

Introducción a Erlang: Shell (I)

- Expresiones aritméticas (enteros y punto flotante)

```
Erlang/OTP 18 [erts-7.3] [64-bit] [smp:4:4] [async-threads:10]

Eshell V7.3  (abort with ^G)
1> (10 - 1) * 2.
18
2> 123456789 * 9876543210 * 9991119998882227776663333.
12182435492257795806352196530649760969857700
3> 16#ebba * 32#imatia.
3786146721786 0
4> 7/3.
2.3333333333333335
5> 10/2.
5.0
6> 7 div 3.
2
7> 7 rem 3.
1
8>
```

Introducción a Erlang: Shell (II)

- Variables

```
4> X = 10.
10
5> X.
10
6> X*X.
100
7> X = 5.
** exception error: no match of right hand side value 5
8>
```

- Atoms

```
8> imatia.
imatia
9> 'hello world'.
'hello world'
10>
```

- Strings

```
25> Company = "Imatia".
"Imatia"
26> [73, 109, 97, 116, 105, 97].
"Imatia"
27> [$I, $m, $a, $t, $i, $a].
"Imatia"
28>
```

Introducción a Erlang: Shell (III)

- Tuplas

```
10> P = {point, 5, 3}.
{point,5,3}
11> CTT = {ticket, {id, 1234}, {ntt, 1001}}.
{ticket,{id,1234},{ntt,1001}}
12> {ticket, {id, Id}, {ntt, CorNtt}} = CTT.
{ticket,{id,1234},{ntt,1001}}
13> Id.
1234
14> CorNtt.
1001
15>
```

- Listas

```
15> RTeam = [angel, denis, vanesa, david].
[angel,denis,vanesa,david]
16> [Member1|RestOfTeam] = RTeam.
[angel,denis,vanesa,david]
17> Member1.
angel
18> RestOfTeam.
[denis,vanesa,david]
19>
```

Programación Secuencial

- Módulos y funciones
- Funs (funciones anóminas)
- Procesado de listas y List Comprehensions
- Guardas
- Registros
- Expresiones Case / If
- Acumuladores
- BIFs
- Binaries
- Bit Syntax
- Y más…

Programación Secuencial. Módulos y funciones

(src/employees.erl)

```
-module(employees).
-export([salary/1]).

salary({internal, Base}) ->
    Base;
salary({external, Base, CommissionBase, CommissionRate}) ->
    Base + CommissionBase*CommissionRate.
```

```
31> c(employees).
{ok,employees}
32> Juan = {internal, 1200}.
{internal,1200}
33> Ana = {external, 1000, 1000, 0.20}.
{external,1000,1000,0.2}
34> employees:salary(Juan).
1200
35> employees:salary(Ana).
1.2e3
```

Programación Secuencial. Funs

```
38> EuroDollarConvert = fun({eur,E}) -> {dol, E*1.1295};
38>                        ({dol,D}) -> {eur, D/1.1295}
38>                     end.
#Fun<erl_eval.6.50752066>
39> EuroDollarConvert({eur,10}).
{dol,11.295}
40> EuroDollarConvert({dol,11.295}).
{eur,10.0}
41>
```

Funciones de alto orden:

```
42> Triple = fun(X) -> 3*X end.
#Fun<erl_eval.6.50752066>
43> lists:map(Triple, [1,2,3,4,5]).
[3,6,9,12,15]
44> Mult = fun(X) -> (fun(Y) -> X*Y end) end.
#Fun<erl_eval.6.50752066>
45> Double = Mult(2).
#Fun<erl_eval.6.50752066>
46> Double(10).
20
47>
```

Programación Secuencial.
Procesado de listas y List Comprehensions

```
53> lists:map(Double,[1,2,3]).
[2,4,6]
54> [2*X || X <- [1,2,3]].
[2,4,6]
55>
```

```
1  -module(quicksort).
2  -export([qsort/1]).
3
4  qsort([]) ->
5      [];
6  qsort([H | T]) ->
7      qsort([ X || X <- T, X < H ]) ++ [H] ++ qsort([ X || X <- T, X >= H ]).
```

```
50> c(quicksort).
{ok,quicksort}
51> quicksort:qsort([23,1,12,13,2,3,5,10,21,9]).
[1,2,3,5,9,10,12,13,21,23]
52>
```

Programación Secuencial. Guardas

```
70> IsRTeamMember = fun(X) when X =:= angel;
70>                             X =:= denis;
70>                             X =:= vanesa;
70>                             X =:= david -> true;
70>                    (X) -> false
70>                 end.
#Fun<erl_eval.6.50752066>
71> IsRTeamMember(vanesa).
true
72> IsRTeamMember(chamoso).
false
73>
```

```
1> IsEvenAndLowerThanTen = fun(X) when (X rem 2 =:= 0), (X < 10) -> true;
1>                            (X) -> false
1>                         end.
#Fun<erl_eval.6.50752066>
2> IsEvenAndLowerThanTen(12).
false
3> IsEvenAndLowerThanTen(2).
true
4> IsEvenAndLowerThanTen(10).
false
5> IsEvenAndLowerThanTen(5).
false
6>
```

Programación Secuencial. Registros

```
12> rr("records.hrl").
[employee]
13> Juan = #employee{name="Juan"}.
#employee{name = "Juan",salary = undefined,satisfied = yes}
14> Ana = #employee{salary=1200, name="Ana"}.
#employee{name = "Ana",salary = 1200,satisfied = yes}
15> Pedro = #employee{name="Pedro", salary=500, satisfied=no}.
#employee{name = "Pedro",salary = 500,satisfied = no}
16> PedroAfterMeeting = Pedro#employee{salary=1500, satisfied=yes}.
#employee{name = "Pedro",salary = 1500,satisfied = yes}
17> #employee{salary=AnaSalary} = Ana.
#employee{name = "Ana",salary = 1200,satisfied = yes}
18> AnaSalary.
1200
19> Ana#employee.salary.
1200
20>
20> rf(employee).
ok
21> Ana.
{employee,"Ana",1200,yes}
22>
```

Programación Secuencial. Expresiones Case / If

```
% CASE EXPRESSION
case Expression of
    Pattern1 [when Guard1] -> Expr_seq1;
    Pattern2 [when Guard2] -> Expr_seq2;
    ...
end

% IF EXPRESSION
if
    Guard1 ->
        Expr_seq1;
    Guard2 ->
        Expr_seq2;
    ...
end
```

Programación Secuencial. Acumuladores

```erlang
-module(acumuladores).
-export([split_list_values/2, split_list_values_acc/2]).

split_list_values(F, L) ->
    Valids = [X || X <- L, (F(X) =:= true)],
    NotValids = [X || X <- L, (F(X) =:= false)],
    {Valids, NotValids}.

split_list_values_acc(F, L) -> split_list_values_acc_aux(F, L, [], []).

split_list_values_acc_aux(F, [H|T], Valids, NotValids) ->
    case F(H) of
        true -> split_list_values_acc_aux(F, T, [H|Valids], NotValids);
        false -> split_list_values_acc_aux(F, T, Valids, [H|NotValids])
    end;
split_list_values_acc_aux(_F, [], Valids, NotValids) ->
    {Valids, NotValids}.
```

Programación Secuencial. BIFs y Binaries

- BIFs (*Built-In Functions*)
  - Normalmente sirven para hacer cosas imposibles de programar en Erlang
  - http://www.erlang.org/doc/man/erlang.html
- Binaries

```
1> Bin = <<1,2,3,4,5>>.
<<1,2,3,4,5>>
2> BinList = list_to_binary([1,2,3,4,5]).
<<1,2,3,4,5>>
3> binary_to_list(BinList).
[1,2,3,4,5]
4> BinTerm = term_to_binary({employee,"Juan",1200}).
<<131,104,3,100,0,8,101,109,112,108,111,121,101,101,107,0,
  4,74,117,97,110,98,0,0,4,176>>
5> binary_to_term(BinTerm).
{employee,"Juan",1200}
6>
```

Programación Secuencial. Bit Syntax

```
6> X = 15.
15
7> Y = 58.
58
8> Z = 8.
8
9> M = <<X:5, Y:6, Z:5>>.
<<127,72>>
10> <<R1:5, Y1:6, Z1:5>> = M.
<<127,72>>
11> R1.
15
12> Y1.
58
13> Z1.
8
14>
```

```
<<E1, E2, ..., En>>
 Ei = Value |
      Value:Size |
      Value/TypeSpecifierList |
      Value:Size/TypeSpecifierList
```

```
TypeSpecifierList:
 - End = little | big | native
 - Sign = signed | unsigned
 - Type = integer | float | binary
 - Unit = 1 | 2 | ... | 255
```

Excepciones

- Excepciones en Erlang
- Bloques try… catch
- Expresiones catch
- Stack traces

Excepciones. Lanzar excepciones

- **exit(Why)**
  - Termina el proceso actual.
  - Envía el mensaje {'EXIT', Pid, Why} a todos los procesos conectados al actual.
- **throw(Why)**
  - Lanza una excepción que el llamador decide si capturar o no.
- **erlang:error(Why)**
  - Errores no esperados

Excepciones. Capturar excepciones (I)

- try... catch

```
try FuncOrExpressionSequence of
    Pattern1 [when Guard1] -> Expressions1;
    Pattern2 [when Guard2] -> Expressions2;
    ...
catch
    ExceptionType: ExPattern1 [when ExGuard1] -> ExExpressions1;
    ExceptionType: ExPattern2 [when ExGuard2] -> ExExpressions2;
    ...
after
    AfterExpressions
end
```

Excepciones. Capturar excepciones (II)

- catch

```
32> catch exceptions:normal_result().
hello
33> catch exceptions:exit_result().
{'EXIT',e}
34> catch exceptions:throw_result().
e
35> catch exceptions:error_result().
{'EXIT',{e,[{exceptions,error_result,0,
                        [{file,"exceptions.erl"},{line,7}]},
            {erl_eval,do_apply,6,[{file,"erl_eval.erl"},{line,674}]},
            {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,431}]},
            {shell,exprs,7,[{file,"shell.erl"},{line,686}]},
            {shell,eval_exprs,7,[{file,"shell.erl"},{line,641}]},
            {shell,eval_loop,3,[{file,"shell.erl"},{line,626}]}]}}
36>
```

Excepciones. Stack traces

```
41> catch exceptions:normal_result().
hello
42> erlang:get_stacktrace().
[]
43> catch exceptions:exit_result().
{'EXIT',e}
44> erlang:get_stacktrace().
[{exceptions,exit_result,0,
              [{file,"exceptions.erl"},{line,5}]},
 {erl_eval,do_apply,6,[{file,"erl_eval.erl"},{line,674}]},
 {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,431}]},
 {shell,exprs,7,[{file,"shell.erl"},{line,686}]},
 {shell,eval_exprs,7,[{file,"shell.erl"},{line,641}]},
 {shell,eval_loop,3,[{file,"shell.erl"},{line,626}]}]
45>
```

Referencias

- http://www.erlang.org/
- http://erlang.org/doc/reference_manual/users_guide.html
- http://erlang.org/erldoc
- *Programming Erlang. Software for a Concurrent World* – Joe Amstrong
- http://learnyousomeerlang.com