Erlang

hello_world() -> io:fwrite("hello, world").

Ben Mezger

2016-06-12 Sun

Outline

1. Introdução
2. Erlang Shell
3. Variáveis
4. Tuplas
5. Listas
6. Funções
7. Processos
8. Paralelismo
9. Conclusões

Visão Geral [1]

- Linguagem concorrente/funcional;
- Também é um sistema garbage-collected runtime;
- Seu design é muito adequado para os seguintes sistemas;
  - Distribuídos;
  - Tolerância a falhas;
  - *Real-time computing*;
  - Sempre disponível – aplicações que nunca param;
  - *Hot swapping* – modificar o código sem precisar parar o sistema que o roda;

[1] Notebook:`http://bit.ly/29hZhmg`

Visão Geral

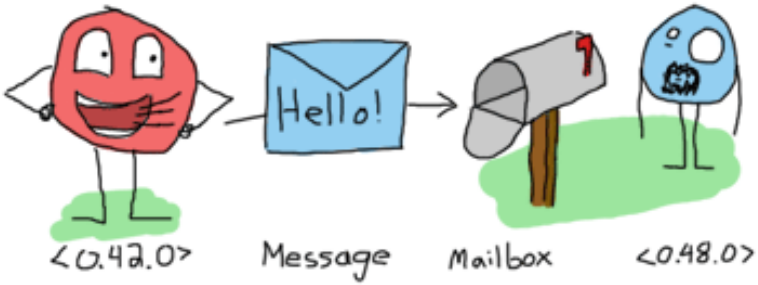
- Originalmente foi uma linguagem proprietária dentro da Ericson;
- Foi escrita por Joe Armstrong, Robert Virding e Mike Williams, em 1986;
- Eventualmente virou open-source em 1998;
- Foi escrita para programar switches/aplicações de telefones;
- Sua licença é Apache License 2.0;
- Ultima versão estável é 18.3;

Actor Model

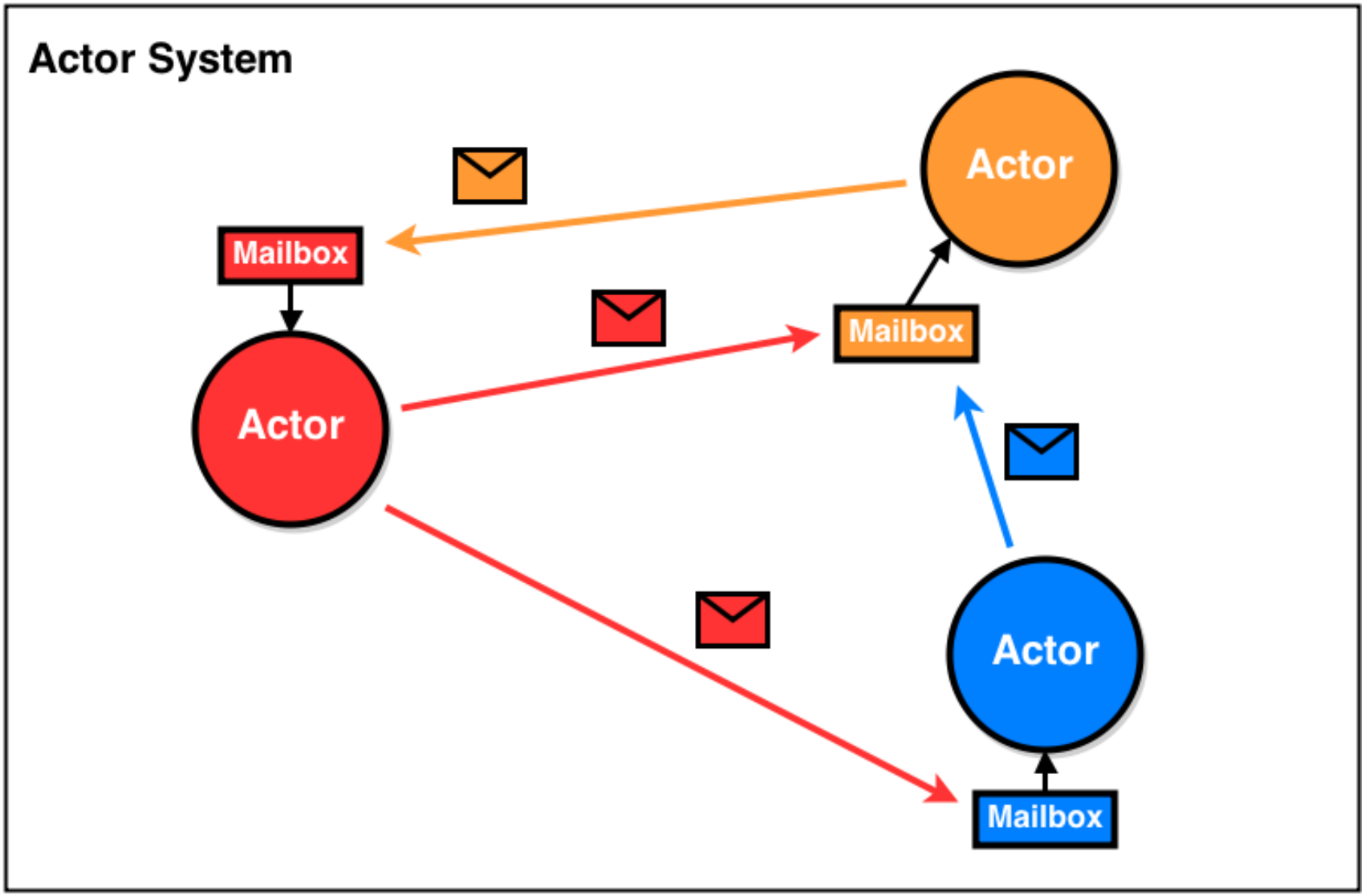
- Erlang usa o modelo *Actor* para rodar varias tarefas ao mesmo tempo;
- Criado em 1973;
- Tudo é um *actor* (similar a filosofia de "*tudo é um objeto em POO*";
- Modelo *Actor* é executado concorrente;
- Um *Actor* é uma entidade computacional que, em resposta a uma mensagem que recebe, ele pode (concorrentemente);
  - Enviar um numero finito de mensagens para outros *actors*;
  - Criar um número finito de novos *actors*;
  - Indicar a ação que deve ser usada para a próxima mensagem;

Actor Model

- Não existe sequencia para as ações acima e elas poderam ser executadas em paralelo;
- Processo separado na máquina virtual;
- Mailbox;



Sistema Actor



O que é?

- Podemos usar o Erlang Shell[2] para testar quase todos as nossas coisas em um emulador;
- Ele irá rodar nossos *scripts* quando compilados;
- Podemos também editar nosso código ao vivo;

```
~ $ erl
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:8:8]
[async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Eshell V7.3  (abort with ^G)
1> 1 + 1.
2
```

[2] Versão online: `http://www.tryerlang.org/`

Números

- Expressões precisam terminar com um "." (ponto) seguido por um espaço;
- Você pode separar expressões usando a "," (virgúla), porém, apenas o resultado da última expressão será mostrada;

```
1> 2 + 15.
17
2> 48 * 100.
4900
3> 1892 - 1472.
420
4> 5 / 2. % Erland não quer saber se você está usando float
2.5
5> 4 div 2. % use div/rem para divisão int-to-int.
2
```

Bases númericas

Se você quiser representar um inteiro em uma base diferente, use *Base#Numero*.

```
1> 2#101010.
42
2> 8#0677.
447
3> 16#AE.
174
```

Variáveis invariáveis

- Variáveis não podem ser variáveis em programação funcional;
- Variáveis iniciam com uma letra maiúscula;

```
1> One.
*1: variable 'One' is unbound
2> One = 1.
1
3> Un = Uno = One = 1.
1
4> Two = One + One.
2
5> Two = 2.
2
6> Two = Two + 1.
*exception error: no match of right hand side value 3
```

Variáveis invariáveis

- Váriavel *Two* pode ser igual a *Two* = *Two* apenas se o lado esquerdo for igual ao lado direito;
- = também é usado para comparar valores;

```
1> 47 = 45 + 2.
47
2> 47 = 45 +
*exception error: no match of right hand side value 48
```

Atoms

- Atoms são constantes/valores com seu próprio nome;
- O Atom *gato* significa *gato*, mais nada;
- Deve ser usado aspa simples;
- Um Atom é referenciado em um "*Atom Table*", que consume 4bytes de memória em um sistema de 32bits, e 8 em um de 64bits;
- Um Atom não é coletado em um *Garbage Collector*;

```
1> atom.
atom
2> atoms_rule.
atoms_rule
3> atoms_rule@erlang.
atoms_rule@erlang
4> 'Atoms can be cheated!'.
'Atoms can be cheated!'
5> atom = 'atom'.
atom
```

Algebra Boleana & Operatores de Comparação

```
1> true and false.
false
2> false or true.
true
3> true xor false.
true
4> not false.
true
5> not (true and true).
false
```

```
6> 5 =:= 5. % 5 == 5
true
7> 1 =:= 0.
false
8> 1 =/= 0. % 1 != 0
true
9> 5 =:= 5.0.
false
10> 5 == 5.0.
true
11> 5 /= 5.0.
false
```

Introdução

- Modo de organizar dados;
- Agrupar vários termos juntos quando você sabe quantos tem;

$$\{\text{Element}^1, \text{Element}^2, \text{Element}^3, \text{Element}^n\}$$

- Então...

```
1> {1, 2, 3, 4, 5, 6, 7}.
{1,2,3,4,5,6,7}
```

Extração de informação

```
3> Point = {4,5}.
{4,5}
4> {X,Y} = Point. % unpack
{4,5}
5> X.
4
6> {X,_} = Point. % váriavel anônima
{4,5}
7> {_,_} = {4,5}.
{4,5}
8> {_,_} = {4,5,6}.
**exception error: no match of right hand side value {4,5,6}
```

Extração de informação

- Podemos misturar *tuples* + *atoms*

```
9> Temperature = 23.213.
23.213
10> PreciseTemperature = {celsius, 23.213}.
{celsius,23.213}
11> {kelvin, T} = PreciseTemperature.
**exception error: no match of right hand side value
  {celsius,23.213}
```

Introdução

- Lista pode contar qualquer tipo;
  - Números, atoms, tuplas, listas;

[1, 2, 3, {numbers, [4, 5, 6]}, 5.34, atom].

- Porém...

```
2> [97, 98, 99].
"abc" % strings = list!
3> [97,98,99,4,5,6].
[97,98,99,4,5,6] % uh?!
4> [233].
"é"
```

Strings ≈ Problema

- Erlang é conhecido por ser péssimo com *strings*;
- Não existe nenhum tipo *string builtin*;
- Empresas de telecomunicações nunca (ou quase nunca) usam *strings*;
  - Nunca sentiram a necessidade de adicionar um "tipo" *string*;
- Manipulação de *string* está sendo migrada aos poucos;
- Sua VM já suporta Unicode strings;

Contanenação de listas

```
5> [1,2,3] ++ [4,5].
[1,2,3,4,5]
6> [1,2,3,4,5] -- [1,2,3].
[4,5]
```

- O primeiro elemento da lista se chama *head*;
- O resto se chama *tail*;

```
11> hd([1,2,3,4]).
1
12> tl([1,2,3,4]).
[2,3,4]
13> List = [2,3,4].
[2,3,4]
14> NewList = [1|List].
[1,2,3,4]
```

List comprehensions

- Um modo de criar/modificar listas;
- Baseado na idéia matemática de *set notation*;

$$\{x \in \mathbb{R} : x = x^2\} \tag{1}$$

```
1> [2*N || N <- [1,2,3,4]]. % {2^n : n in L}
[2,4,6,8]
2> [X || X <- [1,2,3,4,5,6,7,8,9,10], X rem 2 =:= 0].
[2,4,6,8,10]
```

Syntax

- *Pattern matching*;
- Cláusulas de funções;

```
greet(male, Name) ->
    io:format("Hello, Mr. ~s!", [Name]);
greet(female, Name) ->
    io:format("Hello, Mrs. ~s!", [Name]);
greet(_, Name) ->
    io:format("Hello, ~s!", [Name]).
```

```
function(X) ->
    Expression;
function(Y) ->
    Expression;
function(_) ->
    Expression.
```

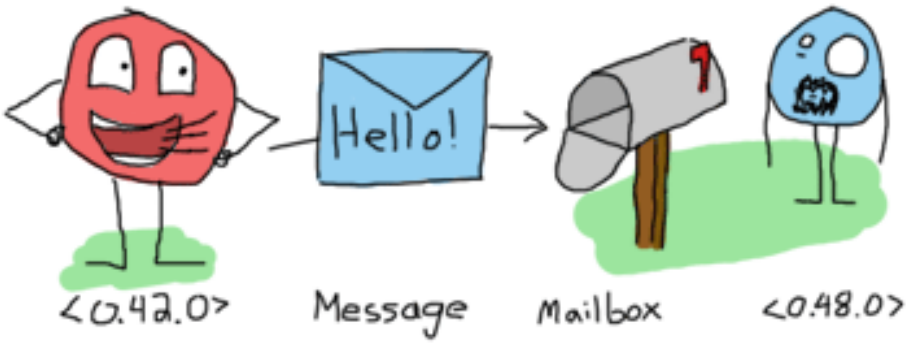
Guards

- *Pattern Matching* é limitada
  - Não conseguimos expressar um *range* de valores ou tipos de dados;
- Cláusulas adicionais que pode ser colocada em uma função (na cabeça) para tornar o *pattern matching* mais expressivo;

```
velho_suficiente(X) when X >= 16 -> true;
velho_suficiente(_) -> false.
```

Como funciona

- No Erlang, podemos colocar funções como processos externos do programa;
- A VM executa uma função em um processo diferente, e depois desaparece;
- Um processo necessita de um *hidden state* = mailbox para mensagens;
- Um processos não retornam nada;



Inicializando um processo

```
1> F = fun() -> 2 + 2 end.
#Fun<erl_eval.20.67289768>
2> spawn(F).
<0.44.0>

3> spawn(fun() -> io:format("~p~n",[2 + 2]) end).
4
<0.46.0>
```

Introdução

- Processos rodam ao mesmo tempo;
- Ordem não é garantida;
- VM do Erlang usa vários "truques" para decidir qual processo rodar primeiro;
- Para verificar mensagens recebidas, use *flush().*;

Exemplo de uma função paralela

```
4> G = fun(X) -> timer:sleep(10), io:format("~p~n", [X]) end.
#Fun<erl_eval.6.13229925>
5> [spawn(fun() -> G(X) end) || X <- lists:seq(1,10)].
[<0.273.0>,<0.274.0>,<0.275.0>,<0.276.0>,<0.277.0>,
<0.278.0>,<0.279.0>,<0.280.0>,<0.281.0>,<0.282.0>]
2
1
4
3
5
8
7
6
10
9
```

Passagem de mensagens

- Queremos passar mensagens de um lado para o outro nos processos;
- Precisamos verificar as mensagens recebidas;
- Necessitamos um modo de acessar nosso *mailbox*;
- O operator *!* é usado para passar mensagens de um lado para o outro;
  - No lado esquerdo ele recebe o PID e no lado direito o um termo qualquer;

    ```
    self() ! hello.
    ```
  - Podemos enviar a mesma mensagem para vários processos diferentes;

    ```
    self() ! self() ! double.
    ```

Mailbox das mensagens

- Mensagens são guardades nas ordens que foram recebidas;
- Para ver o conteúdo do nosso *mailbox*, usamos *flush().*;
- Ver exemplo Mailbox 1, 2 e 3[3];

[3] Notebook: `http://bit.ly/29hZhmg`

Referências

- Erlang Programming Language - https://en.wikipedia.org/wiki/Erlang_(programming_language)
- Erlang - http://www.erlang.org/
- Learn you some Erlang - http://learnyousomeerlang.com
- The Actor Model in 10 minutes - http://www.brianstorti.com/the-actor-model/
- Erlang Style mailboxes - http://www.dalnefre.com/wp/2011/10/erlang-style-mailboxes/