

Paso de mensajes con Erlang

Miguel Emilio Ruiz Nieto

15 de diciembre de 2021

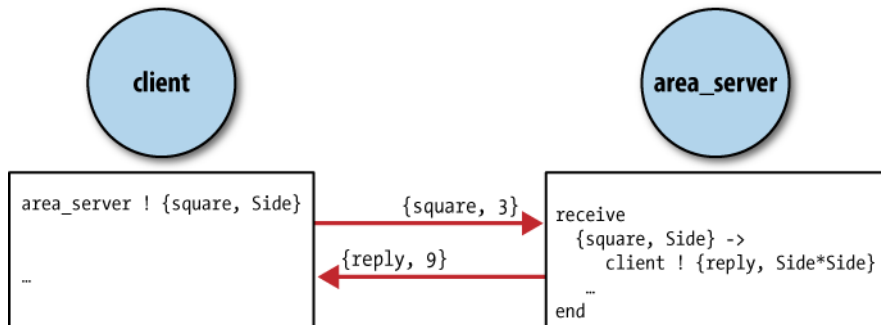
Contenidos

- 1 Introducción
- 2 Erlang
- 3 Ejemplo práctico
- 4 Conclusiones
- 5 Bibliografía

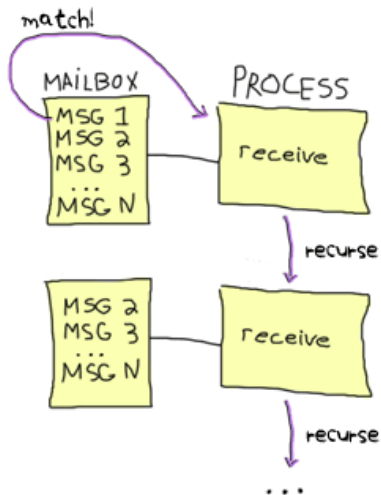
- Hemos visto cómo funciona en computación distribuida el paso de mensajes y los tipos de problemas que puede resolver
- Ahora nos centraremos en cómo funciona el paso de mensajes en Erlang y sus aplicaciones

- Lenguaje de programación desarrollado en Ericsson
- Orientado a sistemas distribuidos:
 - Modelo de actores
 - Paso de mensajes
 - Tolerancia a fallos
 - Alta disponibilidad
 - Filosofía “Let it crash”

Erlang. Modelo de actores



Erlang. Paso de mensajes



```
% Proceso se envia mensaje así mismo  
1> Pid = self().  
<0.84.0>  
2> Pid ! hello.  
hello
```

```
% Crear un nuevo proceso  
Pid = spawn(fun() -> ok end).  
<0.86.0>  
2> Pid ! hello .  
hello
```

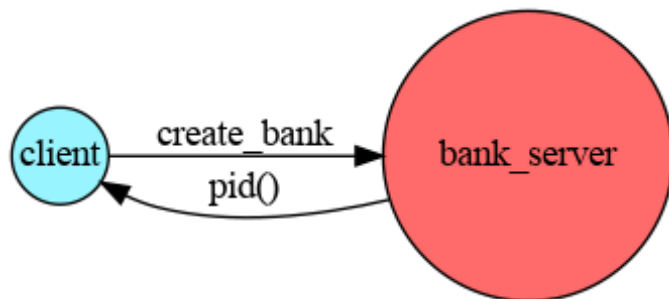


```
loop(State) ->  
  receive  
    Pattern1 when Guard1 -> Expr1;  
    Pattern2 when Guard2 -> Expr2;  
    Pattern3 -> Expr3  
  end.
```

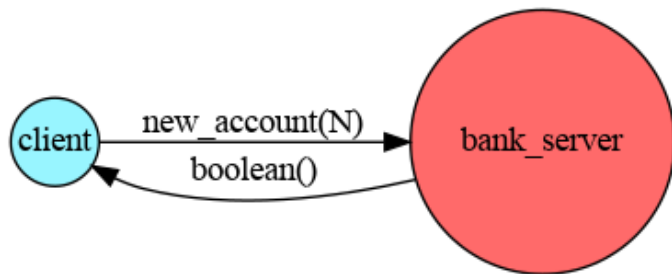
Ejemplo práctico

- Banco que gestiona cuentas del tipo
`{Id::integer(), Balance::integer()}`
- Operaciones:
 - Arrancar servidor
 - Crear cuenta
 - Ingresar y sacar dinero de una cuenta
 - Transferir dinero entre cuentas
 - Consultar saldo de una cuenta
 - Parar servidor

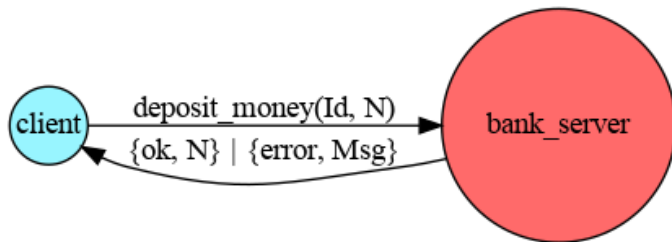
Ejemplo práctico



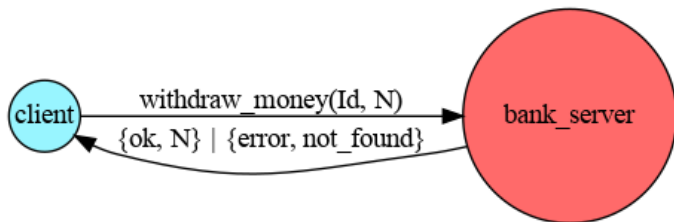
Ejemplo práctico



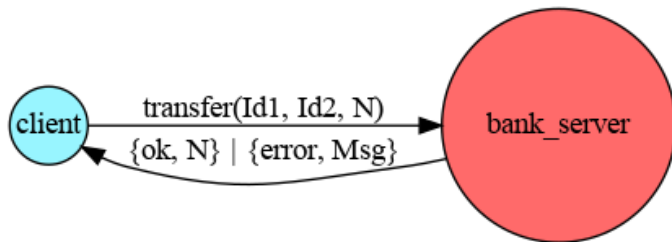
Ejemplo práctico



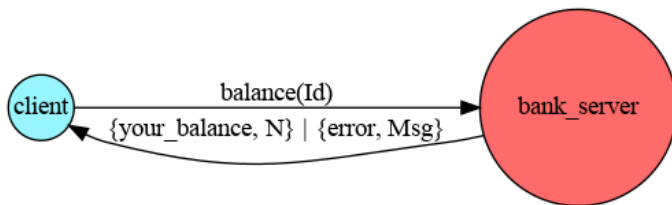
Ejemplo práctico



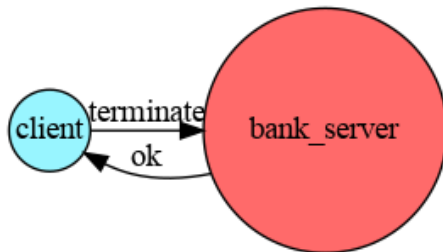
Ejemplo práctico



Ejemplo práctico



Ejemplo práctico



Ejemplo práctico. Implementación

```
%% API
```

```
create_bank() -> spawn(fun() -> loop([]) end).
```

```
terminate(Pid) ->
```

```
Pid ! {terminate,self()},
```

```
receive
```

```
    Msg -> Msg
```

```
end.
```

```
new_account(Pid,AccountNumber) ->
```

```
Pid ! {new_account,AccountNumber,self()},
```

```
receive
```

```
    Msg -> Msg
```

```
end.
```

Ejemplo práctico. Implementación

```
withdraw_money(Pid,AccountNumber,Quantity) ->  
  Pid ! {withdraw_money,AccountNumber,Quantity,self()},  
  receive  
    Msg -> Msg  
  end.
```

```
deposit_money(Pid, AccountNumber,Quantity) ->  
  Pid ! {deposit_money,AccountNumber,Quantity,self()},  
  receive  
    Msg -> Msg  
  end.
```

Ejemplo práctico. Implementación

```
transfer(Pid,FromAccount,ToAccount,Quantity) ->  
  Pid ! {transfer,FromAccount,ToAccount,Quantity,self()},  
  receive  
    Msg -> Msg  
  end.
```

```
balance(Pid,Account) ->  
  Pid ! {balance,Account,self()},  
  receive  
    Msg -> Msg  
  end.
```

Ejemplo práctico. Implementación

```
loop(Accounts) ->
  receive
    {new_account, AccountNumber, From} ->
      case proplists:lookup(AccountNumber, Accounts) of
        none -> %% Podemos crear esa cuenta
          From ! true,
          loop([{AccountNumber, 0} | Accounts]);
        {AccountNumber, _} -> %% Ya existe esa cuenta
          From ! false,
          loop(Accounts)
      end;
  ...
```

Ejemplo práctico. Implementación

```
{withdraw_money, AccountNumber, Quantity, From} ->
  case check_balance(AccountNumber, Accounts) of
    none -> %% No existe la cuenta
      From ! {error, not_found},
      loop(Accounts);
    Balance when Quantity <= Balance ->
      From ! {ok, Quantity},
      UpdatedAccount = {AccountNumber, Balance - Quantity},
      NewAccounts = lists:keyreplace(AccountNumber, 1,
                                     Accounts, UpdatedAccount),
      loop(NewAccounts);
    Balance when Quantity > Balance ->
      From ! {error, not_money},
      loop(Accounts)
  end;
  ...
```

Ejemplo práctico. Implementación

```
{deposit_money,AccountNumber,Quantity,From} ->
  case proplists:lookup(AccountNumber,Accounts) of
    none -> %% No existe la cuenta
      From ! {error, not_found},
      loop(Accounts);
    {AccountNumber,_} ->
      Balance = proplists:get_value(AccountNumber,Accounts)
      NewBalance = Balance + Quantity,
      From ! {ok,NewBalance},
      NewAccounts = lists:keyreplace(AccountNumber,1,
                                     Accounts,{AccountNumber,NewBalance}),
      loop(NewAccounts)
  end;
  ...
```

Ejemplo práctico. Implementación

```
{transfer,FromAccount,ToAccount,Quantity,From} ->
  case check_balance(ToAccount,Accounts) of
    none ->
      From ! {error, to_account_not_found},
      loop(Accounts);
    ToBalance ->
      case check_balance(FromAccount,Accounts) of
        none ->
          From ! {error, from_account_not_found},
          loop(Accounts);
        FromBalance when Quantity <= FromBalance ->
          NewFromBalance = FromBalance - Quantity,
          NewToBalance = ToBalance + Quantity,
          From ! {ok,Quantity},
          NewAccounts = lists:keyreplace(FromAccount,1,
                                         lists:keyreplace(ToAccount,1,
                                                           Accounts,{ToAccount,NewToBalance}),
                                         {FromAccount,NewFromBalance}),
          loop(NewAccounts);
        FromBalance when Quantity > FromBalance ->
          From ! {error, from_account_not_money},
          loop(Accounts)
      end
  end;
...
```


Ejemplo práctico. Implementación

```
{balance,Account,From} ->  
  case proplists:lookup(Account,Accounts) of  
    none ->  
      From ! {error,not_found},  
      loop(Accounts);  
    {Account,_} ->  
      Balance = proplists:get_value(Account,Accounts),  
      From ! {your_balance, Balance},  
      loop(Accounts)  
  end;  
  ...
```

Ejemplo práctico. Implementación

```
{terminate, From} ->  
    From ! ok,  
    ok;  
Unknown ->  
    io:format("Unexpected message: ~p~n", [Unknown]),  
    loop(Accounts)  
end.
```

Ejemplo práctico. Pruebas

```
1> Bank = bank:create_bank().  
<0.86.0>  
2> bank:new_account(Bank, 1).  
true  
3> bank:new_account(Bank, 2).  
true  
4> bank:deposit_money(Bank, 2, 50).  
{ok,50}  
5> bank:withdraw_money(Bank, 1, 20).  
{error,not_money}
```

Ejemplo práctico. Pruebas

```
6> bank:transfer(Bank, 2, 1, 30).  
{ok,30}  
7> bank:balance(Bank, 2).  
{your_balance,20}  
8> bank:balance(Bank, 1).  
{your_balance,30}  
9> bank:withdraw_money(Bank, 1, 20).  
{ok,20}  
10> bank:balance(Bank, 1).  
{your_balance,10}  
11> bank:terminate(Bank).  
ok
```

Ejemplo práctico

- Este ejemplo es la manera más *explícita* de implementar un servidor con estado
- Existen *behaviors* dentro del lenguaje para construir este tipo de arquitecturas

- Erlang y MPI no resuelven los mismos problemas
- Erlang ofrece diversos mecanismos para crear aplicaciones concurrentes, con alta disponibilidad y fiabilidad

- Getting Started with Erlang
https://www.erlang.org/doc/getting_started/intro.html
- Learn You Some Erlang for Great Good! - Fred Hébert
<https://learnyousomeerlang.com>
- Erlang Programming - Francesco Cesarini & Simon Thompson