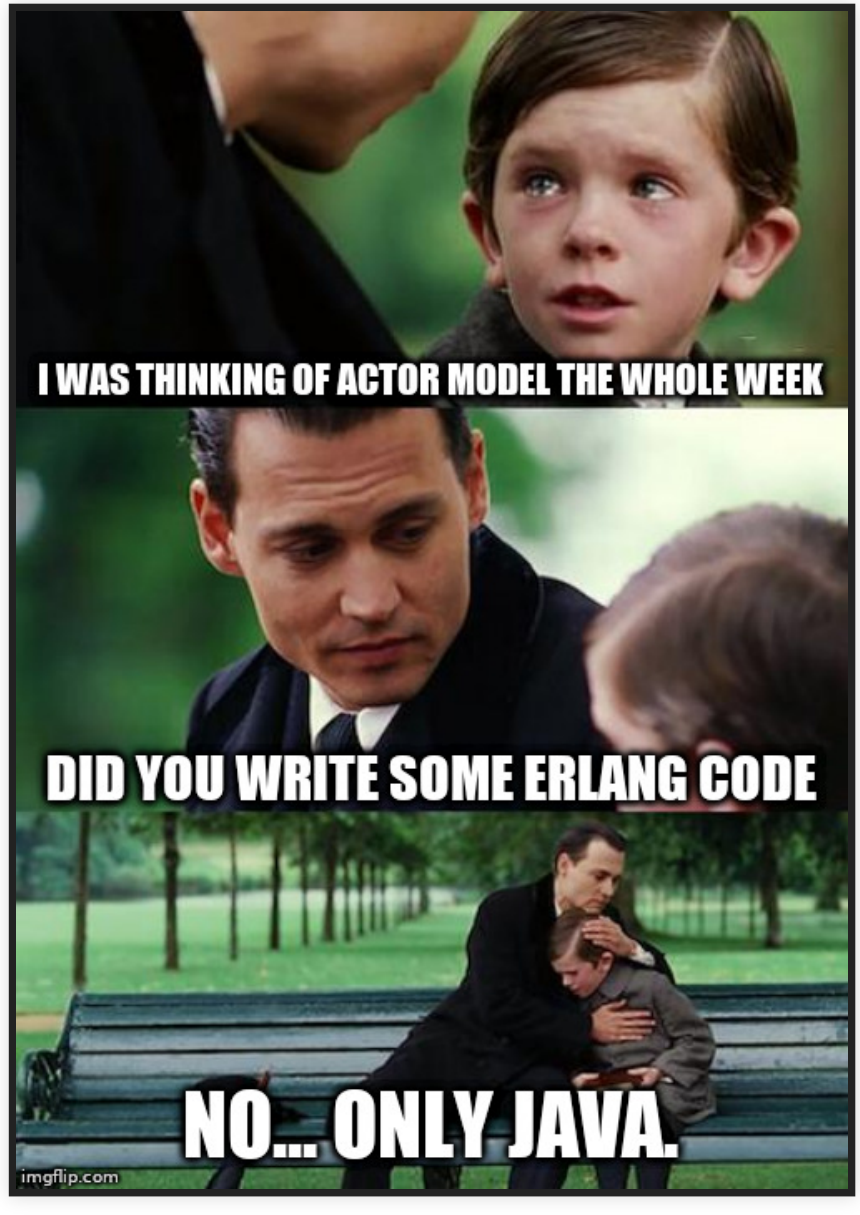
ERLANG PROGRAMMING

Mathieu Kerjouan | @niamtokik


I WAS THINKING OF ACTOR MODEL THE WHOLE WEEK
DID YOU WRITE SOME ERLANG CODE
NO... ONLY JAVA.
imgflip.com

- Boolean: `true` or `false`
- Number: `42` or `$A` or `16#FF` or `4.2` or `4.2e42`
- Atom: `my_atom` or `ok` or `'ERROR'`
- Bitstring: `<<"test">>` or `<<42:8>>`
- Reference: `#Ref<0.33588.38705.27935>`
- Fun: `#Fun<erl_eval.6.39074546>`
- Port: `#Port<0.0>`
- Pid: `<0.42.0>`

- Tuple: `{1, atom, "list"}`
- Map: `#{ key => value }`
- List: `[1,2,42]`
- String: `"test"`
- Record: `#test{ key=value, key2=value2 }`


DEV
ERLANG AND
PATTERN MATCHING
OOP
LANGUAGE
imgflip.com

```
my_match_function(Input) ->
  case Input of
    [H|T] -> do_with_list(H);
    #{ key => Value } -> do_with_map(Value);
    {ok, Value} -> do_something_with(Value);
    {error, Reason} -> alert(Reason);
    Pid when is_pid(Pid) -> do_with_pid(Pid);
    Ref when is_ref(Ref) -> do_with_ref(Ref);
    Else -> wrong_value(Else)
  end
end.
```

I hate exception Excep-what?

```
my_catch_function(Input) ->
  try execute_with(Input)
  catch
    throw:Reason -> do_throw(Reason);
    exit:Reason -> do_exit(Reason);
    error:Reason:Stack -> do_error(Reason, Stack)
  end
end.
```


ONE SHALL USE ERLANG/OTP
imgflip.com

```erlang
-module(my_server).
-export([init/1,terminate/2]).
-export([handle_call/3, handle_cast/2, handle_info/2]).
-behaviour(gen_server).

init(_Args) -> {ok, []}.
terminate(Reason, State) -> ok.

handle_call(Message, From, State) -> {reply, Message, State}.
handle_cast(Message, State) -> {noreply, State}.
handle_info(Message, State) -> {noreply, State}.
```

```erlang
1> ModArgs=[].
2> OTPOpts=[].
3> {ok, Pid} = gen_server:start(my_server, ModArgs, OTPOpts).
4> gen_server:call(Pid, "call message").
5> gen_server:cast(Pid, "cast message").
6> Pid ! "info message".
```

```
-module(my_supervisor).
-export([init/1]).
-behaviour(supervisor).

init(_Args) ->
SupFlags = #{ strategy => one_for_one },
  StartArgs = [my_server, [], []],
  ChildSpec = #{ id => my_server
               , start => {gen_server, start_link, StartArgs}
               },
  Specs = {SupFlags, [ChildSpec]},
  {ok, Specs}.
```

```
1> {ok, Sup} = supervisor:start_link(my_supervisor, []).
2> supervisor:which_children(Sup).
3> [ supervisor:get_childspec(Sup, Id)
   || Id <- supervisor:which_children(Sup) ].
```

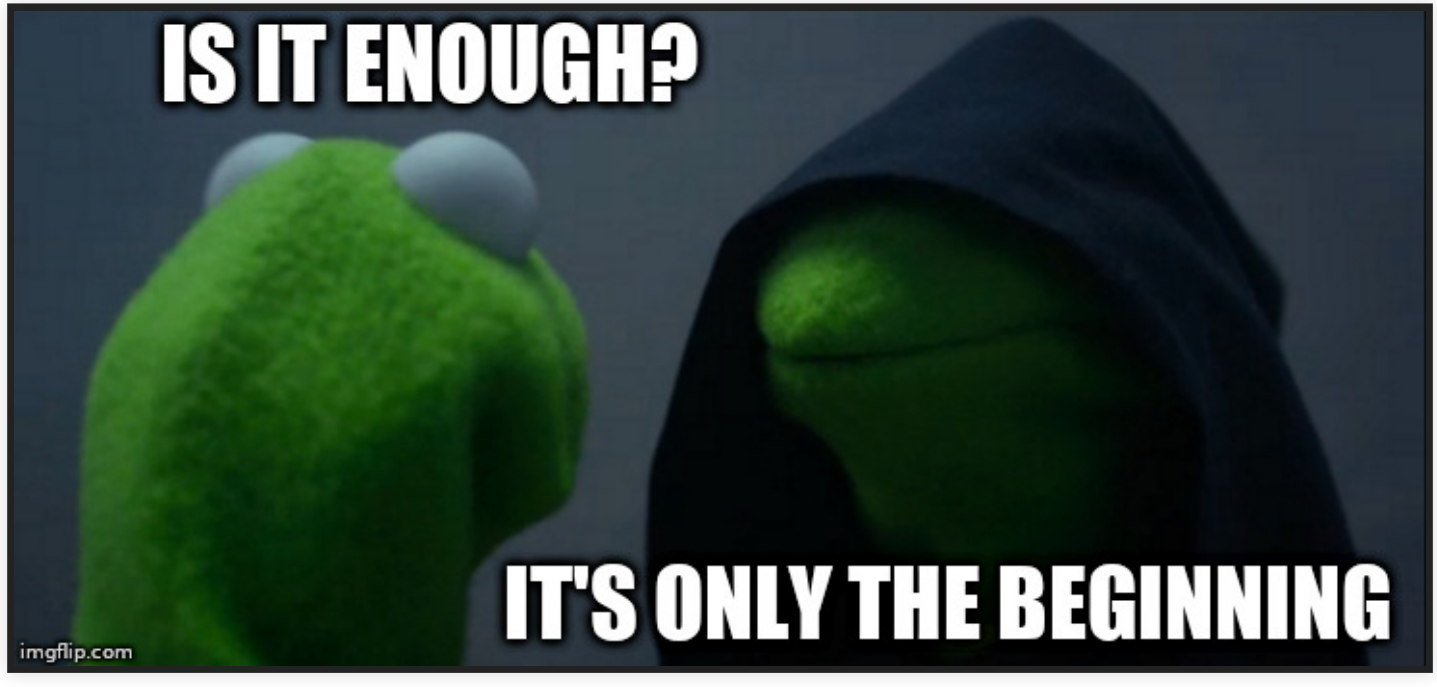
```erlang
-module(my_fsm).
-export([callback_mode/0, init/1, terminate/3]).
-export([first/3, last/3]).
-behaviour(gen_statem).

callback_mode() -> state_functions.
init(_Args) -> {ok, first, []}.
terminate(State, Reason, Data) -> ok.

first(EventType, EventContent, Data) ->
  {next_state, last, Data}.

last(EventType, EventContent, Data) ->
  {next_state, first, Data}.
```

```erlang
1> {ok, FSM} = gen_fsm:start(my_fsm, [], []).
2> gen_fsm:call(FSM, "call message", 1000).
3> gen_fsm:cast(FSM, "cast message").
4> FSM ! "info message".
```


IS IT ENOUGH?
IT'S ONLY THE BEGINNING
imgflip.com

```erlang
-module(my_logger).
-export([adding_handler/1, changing_config/3]).
-export([filter_config/1, log/2, removing_handler/1]).

adding_handler(Config) -> {ok, Config}.

changing_config(_, _, Config) -> {ok, Config}.

filter_config(Config) -> Config.

log(LogEvent, Config) -> io:format("~p~n", [LogEvent]).

removing_handler(Config) -> ok.
```

```erlang
1> MyLogger = #{ id => my_logger, config => [],
                 level => all, module => my_logger }.
2> logger:add_handler(my_logger, my_logger, MyLogger).
3> logger:get_handler_ids().
4> logger:remove_handler(default).
5> logger:info("my message").
```

AND MANY MORE...

- metaprogramming with merl
- unit testing with eunit
- integration testing with commont_test
- static analysis with dialyzer and typer
- integrated debugger/tracer
- benchmark tool with fprof, eprof and fprof
- distributed database with mnesia
- http/s, ftp, ssh client and server with inets
- X widgets with wx
- xml parser with xmerl

Questions?