Clean and Clear Code with Erlang

Anthony Eden
@aeden
DNSimple

What is Erlang?

A complete development environment for concurrent programming

The world is concurrent
Things in the world don't share data
Things communicate with messages
Things fail

Model this in a language

-Joe Armstrong
Concurrency Oriented Programming in Erlang
Nov 9, 2002

Joe Armstrong

An Application Operating System

Purely functional

```erlang
-module(example1).

-export([sum/2]).

sum(A, B) ->
   A + B.
```

```
$ erl
Erlang R15B (erts-5.9)
Eshell V5.9  (abort with ^G)

1> c(example1).
{ok,example1}

2>example1:sum(10, 20).
30
```


Concise

Functions can be stored, passed around and called later

Functions can construct other functions (higher order functions)

Higher Order Functions

```erlang
-module(example1_1).

-export([multiplier/1]).

multiplier(X) ->
  fun(A) ->
      A * X
  end.
```

```
1> c(example1_1).
{ok,example1_1}
2> M = example1_1:multiplier(2).
#Fun<example1_1.0.86284448>
3> lists:map(M, [1,2,3,4]).
[2,4,6,8]
4> lists:map(example1_1:multiplier(10),
[1,2,3,4]).
[10,20,30,40]
```

Single Assignment

```
1> X = 10.
10
2> X = 20.
** exception error: no match of right hand side value 20
3> 10 = X.
10
```


Built for concurrency
Easier to reason about
Easier to debug

Powerful Pattern Matching

```erlang
-module(example2).

-export([sum/1]).

sum(Values) ->
  sum(Values, 0).

sum([], Total) -> Total;
sum([Value|Rest], Total) ->
  sum(Rest, Total + Value).
```

```
1> c(example2).
{ok,example2}
2> example2:sum([10, 20, 30, 40, 50, 100]).
250
```


Pattern matching through polymorphic functions or case statements
Not shown, but functions and cases also support guard conditions
Pattern matches shape of arguments

Concurrency

Concurrency is about structure
Parallelism is about execution

Taken from Rob Pike's talk at Waza, 2012.

```erlang
-module(example3).

-export([babble/1]).

count(N) ->
  timer:sleep(10),
  io:format("The number is ~p~n", [N]).

babble(NumberOfTimes) ->
  [spawn(fun() -> count(N) end) || N <- lists:seq(1, NumberOfTimes)].
```

```
1> c(example3).
{ok,example3}
2> example3:babble(100).
[<0.38.0>,<0.39.0>,<0.40.0>,<0.41.0>,<0.42.0>,<0.43.0>,
 <0.44.0>,<0.45.0>,<0.46.0>,<0.47.0>,<0.48.0>,<0.49.0>,
 <0.62.0>,<0.63.0>,<0.64.0>,<0.65.0>,<0.66.0>|...]
The number is 4
The number is 3
The number is 2
The number is 1
The number is 56
The number is 61
The number is 55
```

Actor Model

Messaging

Processes have a “mailbox”
Messages are sent to a process with the exclamation point

```erlang
-module(example4).

-export([adder/0]).

adder() ->
  receive
    {From, add, A, B} -> From ! sum(A, B);
    {From, _} -> From ! error
  end.

sum(A, B) ->
  A + B.
```

```
1> c(example4).
{ok,example4}
2> Adder = spawn(example4, adder, []).
<0.38.0>
3> Adder ! {self(), add, 10, 30}.
{<0.31.0>,add,10,30}
4> flush().
Shell got 40
ok
```


Actors operate independently, communicate with other actors via messaging, and spawn new actors.

OTP

A set of libraries and guidelines for building applications.

Spawn -> Init -> Receive -> Exit

Behaviors

Behaviors represent the generic parts of a process (implemented by OTP)
Callback modules represent the specific parts of a process (implemented by you)

```erlang
-module(example5).

-behavior(gen_server).

% Gen server hooks
-export([init/1, handle_call/3, handle_cast/2]).
-export([handle_info/2, terminate/2, code_change/3]).

-record(state, {}).

% Gen server callbacks
init([]) ->
  {ok, #state{}}.

handle_call(Message, From, State) ->
  io:format("Received call from ~p: ~p~n", [From, Message]),
  {reply, "Thanks for playing", State}.

handle_cast(Message, State) ->
  timer:sleep(10),
  io:format("Received cast: ~p~n", [Message]),
  {noreply, State}.

handle_info(_Message, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_PreviousVersion, State, _Extra) -> {ok, State}.
```

```
1> c(example5).
{ok,example5}

2> gen_server:start_link({local, example5},
example5, [], []).
{ok,<0.38.0>}

3> gen_server:call(example5, "Hello, there").
Received call from {<0.31.0>,#Ref<0.0.0.77>}:
"Hello, there"
"Thanks for playing"

4> gen_server:cast(example5, {foo, 5}).
ok
Received cast: {foo,5}
```

An Extended Example

```erlang
-module(example6).

-behavior(gen_server).

-export([start_link/0, sum/1, sum/2, async_sum/2]).

% Gen server hooks
-export([init/1, handle_call/3, handle_cast/2]).
-export([handle_info/2, terminate/2, code_change/3]).

-record(state, {}).

% Public API
start_link() ->
  gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).

sum(A, B) ->
  gen_server:call(?MODULE, {sum, A, B}).
sum(Values) ->
  gen_server:call(?MODULE, {sum, Values}).

async_sum(A, B) ->
  gen_server:cast(?MODULE, {sum, A, B}).

% Gen server callbacks
init([]) ->
  {ok, #state{}}.

handle_call({sum, A, B}, _, State) ->
  {reply, sum_list([A, B]), State};
handle_call({sum, Values}, _, State) ->
  {reply, sum_list(Values), State}.

handle_cast({sum, A, B}, State) ->
  io:format("Sum: ~p~n", [sum_list([A, B])]),
  {noreply, State}.

handle_info(_Message, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_PreviousVersion, State, _Extra) -> {ok, State}.

% Internal functions
sum_list(Values) ->
  lists:foldl(fun(X, Sum) -> X + Sum end, 0, Values).
```

```erlang
-module(example6).

-behavior(gen_server).

-export([start_link/0, sum/1, sum/2, async_sum/2]).

% Gen server hooks
-export([init/1, handle_call/3, handle_cast/2]).
-export([handle_info/2, terminate/2, code_change/3]).

-record(state, {}).
```

```erlang
% Public API
start_link() ->
  gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).

sum(A, B) ->
  gen_server:call(?MODULE, {sum, A, B}).
sum(Values) ->
  gen_server:call(?MODULE, {sum, Values}).

async_sum(A, B) ->
  gen_server:cast(?MODULE, {sum, A, B}).
```

```erlang
% Gen server callbacks
init([]) ->
  {ok, #state{}}.

handle_call({sum, A, B}, _, State) ->
  {reply, sum_list([A, B]), State};
handle_call({sum, Values}, _, State) ->
  {reply, sum_list(Values), State}.

handle_cast({sum, A, B}, State) ->
  io:format("Sum: ~p~n", [sum_list([A, B])]),
  {noreply, State}.

handle_info(_Message, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_PreviousVersion, State, _Extra) -> {ok, State}.
```

```erlang
% Internal functions
sum_list(Values) ->
  lists:foldl(fun(X, Sum) -> X + Sum end, 0, Values).
```


Other behaviors include gen_event, gen_fsm and supervisor
New behaviors can be defined (behaviour is generic part, callback is specific part)

```
1> c(example6).
{ok,example6}
2> example6:start_link().
{ok,<0.38.0>}
3> example6:sum(1, 2).
3
4> example6:sum([10, 20, 30, 40]).
100
5> example6:async_sum(1, 2).
Sum: 3
ok
```

Supervisors

Manages OTP processes
Can define conditions for restart
Supervision trees are common constructs in OTP applications

```erlang
-module(example7).

-behavior(supervisor).

-export([start_link/0]).

% Supervisor hooks
-export([init/1]).

% Public API
start_link() ->
  supervisor:start_link(?MODULE, []).

% Supervisor callbacks
init([]) ->
  {ok, {{one_for_one, 10, 10}, [{example7_1, {example7_1,
start_link, []}, permanent, 1000, worker, [example7_1]}]}}.
```

```erlang
-module(example7_1).

-behavior(gen_server).

-export([start_link/0, do_it/0]).

% Gen server hooks
-export([init/1, handle_call/3, handle_cast/2]).
-export([handle_info/2, terminate/2, code_change/3]).

-record(state, {count, max_iterations=3}).

% Public API
start_link() ->
  io:format("~p start_link() called.~n", [?MODULE]),
  gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).

do_it() ->
  gen_server:call(?MODULE, doit).

% Gen server callbacks
init([]) ->
  {ok, #state{count=0}}.

handle_call(doit, _, State) ->
  MaxIterations = State#state.max_iterations,
  case State#state.count of
    MaxIterations -> {stop, normal, ok, State};
    _ ->{reply, State#state.count, State#state{count=State#state.count + 1}}
  end.

handle_cast(doit, State) ->
  {noreply, State#state{count=State#state.count + 1}}.

handle_info(Message, State) ->
  io:format("Received info message: ~p~n", [Message]),
  {noreply, State}.

terminate(Reason, _State) ->
  io:format("Terminated with reason: ~p~n", [Reason]),
  ok.

code_change(_PreviousVersion, State, _Extra) -> {ok, State}.
```

```erlang
-module(example7_1).

-behavior(gen_server).

-export([start_link/0, do_it/0]).

% Gen server hooks
-export([init/1, handle_call/3, handle_cast/2]).
-export([handle_info/2, terminate/2, code_change/3]).

-record(state, {count, max_iterations=3}).
```

```erlang
% Public API
start_link() ->
  io:format("~p start_link() called.~n", [?MODULE]),
  gen_server:start_link({local, ?MODULE}, ?MODULE, [],
[]).

do_it() ->
  gen_server:call(?MODULE, doit).
```

```erlang
% Gen server callbacks
init([]) ->
  {ok, #state{count=0}}.

handle_call(doit, _, State) ->
  MaxIterations = State#state.max_iterations,
  case State#state.count of
    MaxIterations -> {stop, normal, ok, State};
    _ ->{reply, State#state.count, State#state{count=State#state.count + 1}}
  end.

handle_cast(doit, State) ->
  {noreply, State#state{count=State#state.count + 1}}.
```

```erlang
handle_info(Message, State) ->
  io:format("Received info message: ~p~n", [Message]),
  {noreply, State}.

terminate(Reason, _State) ->
  io:format("Terminated with reason: ~p~n", [Reason]),
  ok.

code_change(_PreviousVersion, State, _Extra) -> {ok, State}.
```

```
1> c(example7).
{ok,example7}

2> c(example7_1).
{ok,example7_1}

3> example7:start_link().
example7_1 start_link() called.
{ok,<0.43.0>}

4> example7_1:do_it().
0

5> example7_1:do_it().
1

6> example7_1:do_it().
2

7> example7_1:do_it().
Terminated with reason: normal
example7_1 start_link() called.
ok

8> example7_1:do_it().
0
```

Applications

Reusable component that implements some functionality
Can be started and stopped as a unit

Real-world Erlang with erl-dns

Example: Content Parsing

```erlang
parse_content(Content, _, ?DNS_TYPE_NS_BSTR) ->
  #dns_rrdata_ns{dname=Content};
parse_content(Content, _, ?DNS_TYPE_CNAME_BSTR) ->
  #dns_rrdata_cname{dname=Content};
parse_content(Content, _, ?DNS_TYPE_PTR_BSTR) ->
  #dns_rrdata_ptr{dname=Content};
```


Demonstration of pattern matching

Example: Record Type Filtering

```erlang
% Function generator
match_type(Type) ->
  fun(R) when is_record(R, dns_rr) ->
      R#dns_rr.type =:= Type
  end.
```

```erlang
% Applying the function to a list of records
lists:filter(match_type(?DNS_TYPE_SOA), Records)
```


Demonstrates higher-order functions.

Example: Packet Cache

Demonstrates an OTP gen_server

```erlang
-module(erldns_packet_cache).

-behavior(gen_server).

% API
-export([start_link/0, get/1, put/2, sweep/0, clear/0]).

% Gen server hooks
-export([init/1,
    handle_call/3,
     handle_cast/2,
     handle_info/2,
     terminate/2,
     code_change/3
         ]).

-define(SERVER, ?MODULE).
-define(SWEEP_INTERVAL, 1000 * 60 * 10). % Every 10 minutes

-record(state, {ttl, tref}).
```

```erlang
%% Public API
start_link() ->
  gen_server:start_link({local, ?SERVER}, ?MODULE, [], []).

get(Question) ->
  gen_server:call(?SERVER, {get_packet, Question}).
put(Question, Response) ->
  gen_server:call(?SERVER, {set_packet, [Question, Response]}).
sweep() ->
  gen_server:cast(?SERVER, {sweep, []}).
clear() ->
  gen_server:cast(?SERVER, {clear}).
```

```erlang
%% Gen server hooks
init([]) ->
  init([20]);
init([TTL]) ->
  ets:new(packet_cache, [set, named_table]),
  {ok, Tref} = timer:apply_interval(?SWEEP_INTERVAL, ?MODULE, sweep, []),
  {ok, #state{ttl = TTL, tref = Tref}}.
handle_call({get_packet, Question}, _From, State) ->
  case ets:lookup(packet_cache, Question) of
    [{Question, {Response, ExpiresAt}}] ->
      {_,T,_} = erlang:now(),
      case T > ExpiresAt of
        true ->
          lager:debug("Cache hit but expired"),
          {reply, {error, cache_expired}, State};
        false ->
          lager:debug("Time is ~p. Packet hit expires at ~p.", [T, ExpiresAt]),
          {reply, {ok, Response}, State}
      end;
    _ -> {reply, {error, cache_miss}, State}
  end;
handle_call({set_packet, [Question, Response]}, _From, State) ->
  {_,T,_} = erlang:now(),
  ets:insert(packet_cache, {Question, {Response, T + State#state.ttl}}),
  {reply, ok, State}.
handle_cast({sweep, []}, State) ->
  lager:debug("Sweeping packet cache"),
  {_, T, _} = erlang:now(),
  Keys = ets:select(packet_cache, [{{'$1', {'_', '$2'}}, [{'<', '$2', T - 10}], ['$1']}]),
  lager:debug("Found keys: ~p", [Keys]),
  lists:foreach(fun(K) -> ets:delete(packet_cache, K) end, Keys),
  {noreply, State};
handle_cast({clear}, State) ->
  ets:delete_all_objects(packet_cache),
  {noreply, State}.
handle_info(_Message, State) ->
  {noreply, State}.
terminate(_Reason, _State) ->
  ets:delete(packet_cache),
  ok.
code_change(_PreviousVersion, State, _Extra) ->
  {ok, State}.
```

```
init([]) ->
  init([20]);
init([TTL]) ->
  ets:new(packet_cache, [set, named_table]),
  {ok, Tref} = timer:apply_interval(?SWEEP_INTERVAL, ?MODULE,
sweep, []),
  {ok, #state{ttl = TTL, tref = Tref}}.
```

```
handle_call({get_packet, Question}, _From, State) ->
  case ets:lookup(packet_cache, Question) of
    [{Question, {Response, ExpiresAt}}] ->
      {_,T,_} = erlang:now(),
      case T > ExpiresAt of
        true ->
          lager:debug("Cache hit but expired"),
          {reply, {error, cache_expired}, State};
        false ->
          lager:debug("Time is ~p. Packet hit expires at ~p.", [T, ExpiresAt]),
          {reply, {ok, Response}, State}
      end;
    _ -> {reply, {error, cache_miss}, State}
  end;
handle_call({set_packet, [Question, Response]}, _From, State) ->
  {_,T,_} = erlang:now(),
  ets:insert(packet_cache, {Question, {Response, T + State#state.ttl}}),
  {reply, ok, State}.
```

```
handle_cast({sweep, []}, State) ->
  lager:debug("Sweeping packet cache"),
  {_, T, _} = erlang:now(),
  Keys = ets:select(packet_cache, [{{'$1', {'_', '$2'}}, [{'<', '$2',
T - 10}], ['$1']}]),
  lager:debug("Found keys: ~p", [Keys]),
  lists:foreach(fun(K) -> ets:delete(packet_cache, K) end, Keys),
  {noreply, State};
handle_cast({clear}, State) ->
  ets:delete_all_objects(packet_cache),
  {noreply, State}.
```

```erlang
handle_info(_Message, State) ->
  {noreply, State}.
terminate(_Reason, _State) ->
  ets:delete(packet_cache),
  ok.
code_change(_PreviousVersion, State, _Extra) ->
  {ok, State}.
```

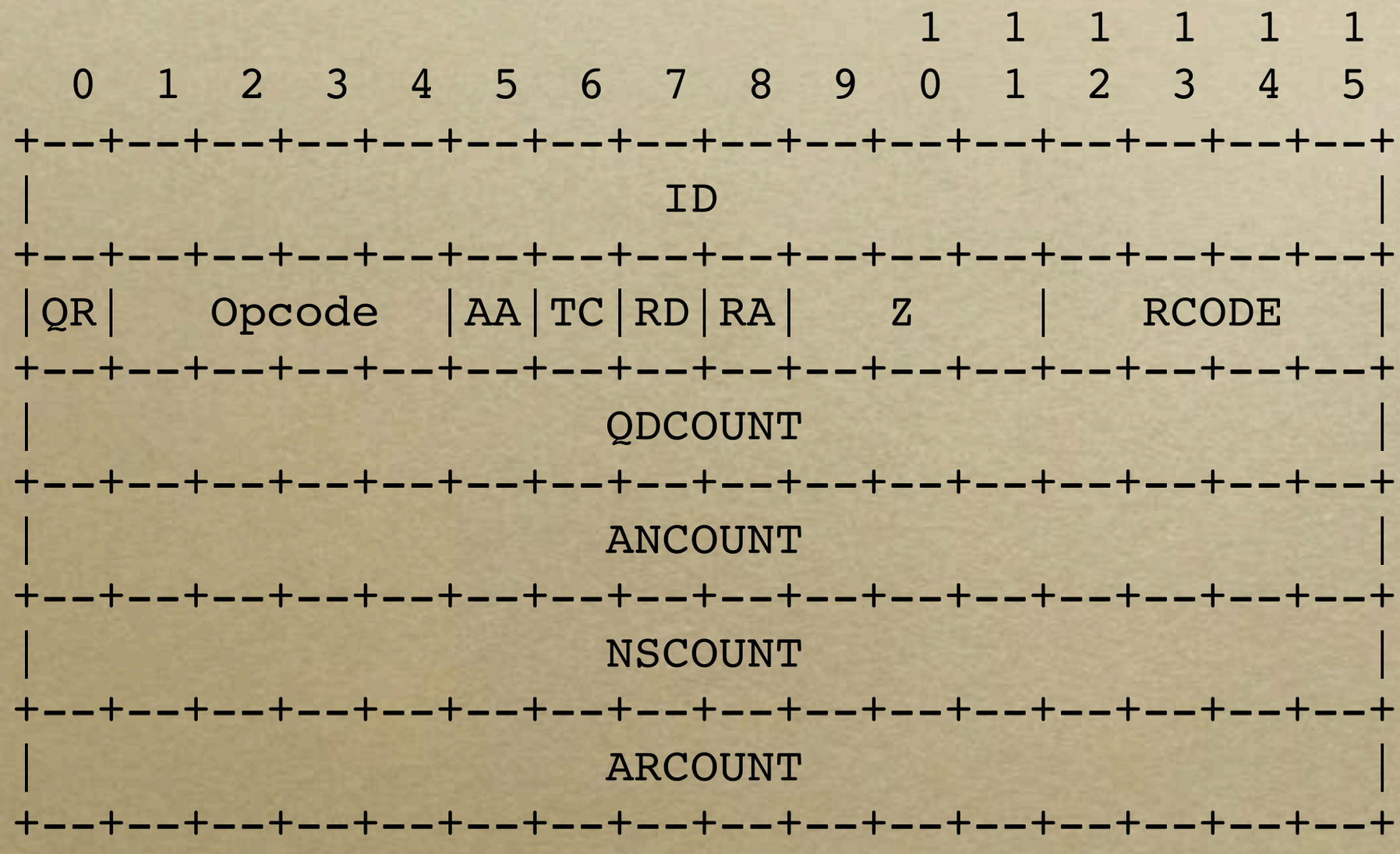
Example: Bit Syntax

Demonstrates Erlang's bit syntax

```
decode_message(<<Id:16, QR:1, OC:4, AA:1, TC:1, RD:1, RA:
1, 0:1, AD:1, CD:1, RC:4, QC:16, ANC:16, AUC:16, ADC:16,
Rest/binary>> = MsgBin) ->
```


Destructuring binary data
Anyone know what this is?

```
decode_message(<<Id:16, QR:1, OC:4, AA:1, TC:1, RD:1, RA:
1, 0:1, AD:1, CD:1, RC:4, QC:16, ANC:16, AUC:16, ADC:16,
Rest/binary>> = MsgBin) ->
```



Ohai RFC 1035

http://www.ietf.org/rfc/rfc1035.txt Section 4.1.1

How deep down the rabbit hole do you want to go?
Hot code swapping, distributed Erlang, Mnesia

Want to Learn More?

http://learnyousomeerlang.com

http://erlang.org

Error handling, distributed systems, ETS, DETS, mnesia and more
Many libraries on github

Clean and Clear Code with Erlang

Anthony Eden
@aeden
DNSimple

https://gist.github.com/4483378