Erlang: проще некуда?

Марат Ахин

10 ноября 2017 г.

Санкт-Петербургский политехнический университет

История языка Erlang

- 1982–1985 — Эрикссон пытается найти подходящий язык для разработки телекоммуникационных систем
- 1985–1986 — Эксперименты с Lisp/Prolog
- 1987 — Первые наброски будущего Erlang'а
- 1988–1989 — Использование Erlang внутри Эрикссон
- 1990 — Erlang вышел в свет
- ...
- 1998 — Erlang стал open-source проектом

Erlang на практике?

- You know them
  - WhatsApp
  - Facebook Messenger
  - Amazon SimpleDB
- You probably do not know them
  - RabbitMQ
  - CouchDB
  - ejabberd
  - ...

Erlang 101

```
-module(test).
-export([fac/1]).

fac(1) ->
    1;
fac(N) ->
    N * fac(N - 1).
```

- Функции нужно явно экспортировать
- Переменные начинаются с большой буквы
- Точка заканчивает объявления

Атомы

```
-module(log).
-export([log/2]).

log(Value, Base) -> ... ;

% Как задать `e` в качестве основания?

log(Value, 2.718281828459045...) -> ... ; % :-(

% Может, можно лучше?

log(Value, e) -> ... ;
```

- Атомы начинаются с маленькой буквы
- Именуют (обозначают) какой-либо объект в программе
- Не имеют значения

Кортежи

```
-module(log).
-export([log/1]).

log({Value, e})    -> math:log(Value);
log({Value, 10})   -> math:log10(Value);
log({Value, 2})    -> math:log2(Value);
log({Value, Base}) -> math:log(Value) / math:log(Base).
```

Guards

```
-module(log).
-export([log/1]).

log({Value, e})    -> math:log(Value);
log({Value, 10})   -> math:log10(Value);
log({Value, 2})    -> math:log2(Value);
log({Value, Base}) when not (Base =< 0) andalso Base /= 1
                   -> math:log(Value) / math:log(Base).
```

- `andalso` / `oralso` “закорачивают” вычисления
- “Меньше или равно” обозначается как `=<`

Списки

```
-module(log).
-export([log/1, log_of_list/1]).

log_of_list([H | T]) -> [log(H) | log_of_list(T)];
log_of_list([]) -> [].
```

Строки


WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING

Строки

```
> [72,101,108,108,111,32,87,111,114,108,100,10].
"Hello World\n"

> "foo" ++ "bar".
"foobar"

> [FirstLetter | Rest] = "foobar".
"foobar"
```

- Строка — это список символов (чисел)
- Можно делать все, что угодно
    - Вот только не очень удобно

Pattern matching

```
foo() ->
    X = 42,
    {Y, Z} = {foo, "bar"},
    [H, X | T] = [1, 42, 2, 3],
    X = 43, % Nope
    [H, X | T] = [100, 200, 300]. % Nope
```

- Шаблон состоит из
  - литералов (`42`, `foo`)
  - вакансий (`_`)
  - свободных переменных (только слева)
  - связанных переменных
- Связанные переменные изменять нельзя

Useful stuff

- `http://erlang.org/doc/man/erlang.html`
- `http://erlang.org/doc/man/lists.html`

Давайте посмотрим, что там =)

Intermission

Ключевая особенность Erlang

- Одна из первых (если не самая первая) реализация модели акторов
- Очень сильно упрощает разработку
    - Не без некоторых ограничений
- Очень сильно увеличивает надежность системы

Модель акторов

- Все — это актор
- У актора есть адрес, на который отправляются сообщения
- Актор может обработать сообщение и
  - отправить конечное число сообщений другим акторам (в том числе, самому себе)
  - изменить свое поведение
  - создать конечное число новых акторов

А что в Erlang?

Модель акторов

Акторы	Erlang
Актор	Одиночный процесс
Адрес	PID (Process IDentifier)
Сообщение	Любой валидный терм
Поведение	Функция
Создание актора	`spawn`
Удаление	Завершение функции
Отправка сообщения	`To ! Message`
Обработка сообщения	`receive ... end`

Пинг понг

```
-module(ping_pong).

-export([start/0, ping/1, pong/0]).

start() ->
    register(pong, spawn(ping_pong, pong, [])),
    spawn(ping_pong, ping, [17]).
```

Пинг понг

```
ping(0) ->
    pong ! finished,
    io:format("Ping finished~n", []);

ping(N) ->
    pong ! {ping, self()},
    receive
        pong ->
            io:format("Ping received pong~n", [])
    end,
    ping(N - 1).
```

Пинг понг

```
pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, PingPID} ->
            io:format("Pong received ping~n", []),
            PingPID ! pong,
            pong()
    end.
```

Пинг понг

```
pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, PingPID} ->
            io:format("Pong received ping~n", []),
            PingPID ! pong,
            pong()
    after 1000 ->
        io:format("Where's my bo... ping?~n", []),
        pong()
    end.
```

Сообщения

- У каждого процесса есть очередь входящих сообщений
- `receive` достает из очереди самое старое сообщение, подходящее под один из шаблонов
    - Если ни один из шаблонов не подходит, сообщение остается в очереди
    - `_ -> ignore`
- Сообщения доставляются асинхронно
    - Порядок доставки для произвольных процессов неспецифицирован
    - Сообщения между какой-либо парой процессов доставляются в порядке отправки

Intermission

Напиши свой сервер за 5 минут

```
-module(generic_server).
-export([start/2, stop/1, request/2]).

start(InitialState, Handler) ->
    spawn(fun () -> loop(InitialState, Handler) end).

stop(Server) ->
    Server ! {stop, self(), make_ref()},
    ok.
```

Напиши свой сервер за 5 минут

```
loop(State, Handler) ->
    receive
        {request, From, Ref, Request} ->
            case Handler(State, Request) of
                {reply, NewState, Result} ->
                    From ! {response, Ref, Result},
                    loop(NewState, Handler)
            end;
        {stop, _From, _Ref} -> ok;
        _ -> ignore
    end.
```

Напиши свой сервер за 5 минут

```
request(Server, Request) ->
    Ref = make_ref(),
    Server ! {request, self(), Ref, Request},
    receive {response, Ref, Result} -> Result end.
```

Напиши свой сервер за 5 минут

```
-module(math_server).
-export([start/0, stop/1, log/2]).

start() ->
    generic_server:start(orddict:new(), fun math_handler/2).

stop(Server) ->
    generic_server:stop(Server).

log(Server, Args) ->
    generic_server:request(Server, {log, Args}).
```

Напиши свой сервер за 5 минут

```
update_state(OldState) ->
    orddict:update_counter(processed, 0, OldState).

math_handler(OldState, {log, Args}) ->
    {reply, update_state(OldState), log:log(Args)}.
```

Обработка ошибок

- Что будет, если `math_handler` упадет?
  - Упадет весь сервер
  - Вряд ли это то, что мы хотим...
- Одна из возможных идиом обработки ошибок — явно их обрабатывать

Обработка ошибок

```
loop(State, Handler) ->
    receive
        {request, From, Ref, Request} ->
            try Handler(State, Request) of
                {reply, NewState, Result} ->
                    From ! {response, Ref, Result},
                    loop(NewState, Handler)
            catch Type:Reason ->
                From ! {error, Ref, {Type, Reason}},
                loop(State, Handler)
            % after -> ...
            end;
        {stop, _From, _Ref} -> ok;
        _ -> ignore
    end.
```

Hot swap

- В телекоммуникационных системах нельзя ничего выключать
- Функциональность надо развивать

Что делать?

Hot swap

```
loop(State, Handler) ->
    receive
        {request, From, Ref, Request} ->
            ...
        {update, From, Ref, NewHandler} ->
            From ! {ok, Ref},
            loop(State, NewHandler);
        {stop, _From, _Ref} -> ok;
        _ -> ignore
    end.
```

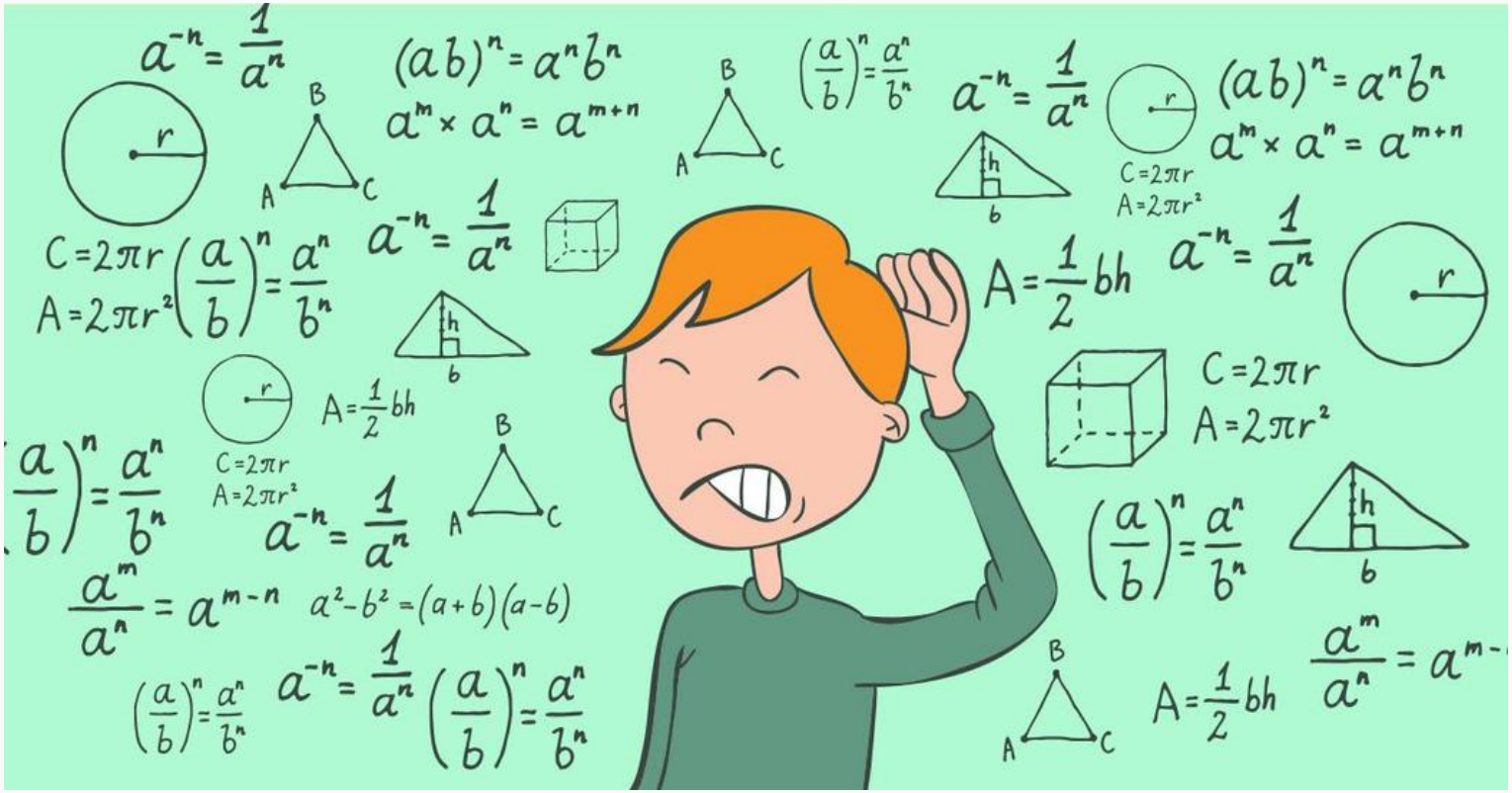
Hot swap

```
update(Server, NewHandler) ->
    Ref = make_ref(),
    Server ! {update, self(), Ref, NewHandler},
    receive {ok, Ref} -> ok end.
```

OTP

- Open Telecom Platform
- Содержит реализацию всех полезных шаблонов
    - в том числе, все, что мы с вами сделали раньше
- Основа большинства реальных проектов на Erlang

And now for some code...



What I did not learn today

- Супервизоры
- Библиотеки из OTP
- Erlang VM
- `dialyzer`
- ETS (Erlang Term Storage)
- Отображения (maps)
- Работа с бинарными данными
- `if`
- List comprehensions
- Записи (records)
- Макросы
- Как это все собирается и запускается

What I learned today