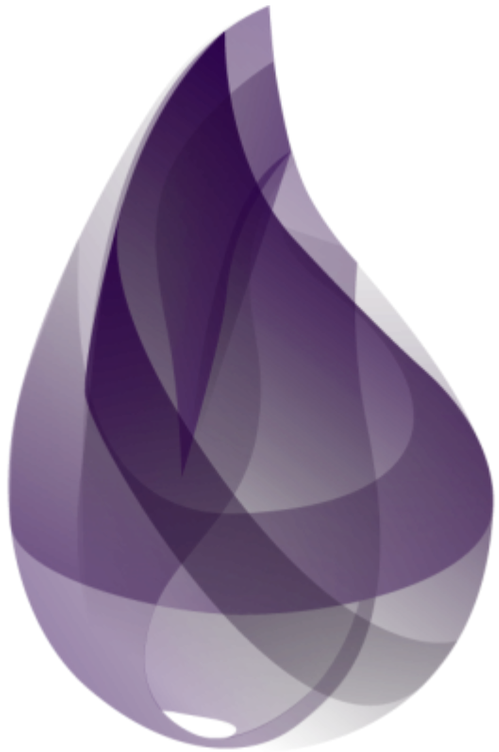

Функциональное программирование в Elixir.

Вольдэмар Дулецкий (@r00takaspin), компания Evrone

|> Roadmap

- Вводная в Elixir/Erlang
- Примеры кода из реального проекта
- Великий и могучий OTP 🔥
- Полезный инструментарий: dialyzer, exunit, dogma
- Область применения: где стоит использовать, а где совсем не стоит
- Что почитать и куда двигаться дальше?

|> Фичи Erlang

- функциональный язык
- многопоточный
- паттерн матчинг
- акторная модель
- динамически типизирован
- компилируемый

*язык задумывался чтобы параллелить вызовы библиотек на C

Пример кода:

```
-module(sup).
-export([start/2, start_link/2, init/1, loop/1]).

start(Mod,Args) ->
  spawn(?MODULE, init, [{Mod, Args}]).

start_link(Mod,Args) ->
  spawn_link(?MODULE, init, [{Mod, Args}]).

init({Mod,Args}) ->
  process_flag(trap_exit, true),
  loop({Mod,start_link,Args}).

loop({M,F,A}) ->
  Pid = apply(M,F,A),
  receive
        {'EXIT', _From, shutdown} -> exit(shutdown);
        {'EXIT', Pid, Reason} -> io:format("Process ~p exited for reason ~p~n",[
  loop({M,F,A})
end.
```

|> Проблемы Erlang 😡

- устаревший синтаксис (место для вашего холивара)
- язык очень медленно развивается
- долгое время не было хорошего менеджера зависимостей (с rebar3 дела стали идти получше)
- постоянно приходилось ставить точку в конце строчки или запятую
- переменные нельзя переопределять
- мало современной документации по построению веб-приложений

* не показывать эрлангистам, они могут быть очень злыми

* никогда-никогда-никогда не упоминайте JAVA в присутствии эрлангистов: услышите много оскорблений про себя, членов вашей семьи и ваших домашних животных 💩

|> ELIXIR! ❤️

- production ready
- активно развивается, писал бывший рубист José Valim
- хороший менеджер пакетов
- отличная дока, много современных гайдов
- локаничность Ruby (приятный синтаксис, плюшки функциональных языков)
- позволяет писать и использовать приятные DSL
- возможности OTP

|> Приготовьтесь, будет много сахара 😻

|> Булевы функции:

```
  def nick_valid?(nick) do
    Regex.match?(~r/\A[a-z_\-\[\]\\^{}|`][a-z0-9_\-\[\]\\^{}|`]{2,9}\z/i, nick)
  end

  "john" |> nick_valid? #=> true
```

|> Атомы (очень похожи на Symbols) ☘

Очень удобны для обмена сообщениями между процессами

```
#iex> :hello
:hello
```

* не создавайте в памяти много атомов - от них сильно течет память

|> Pattern matching ❤️

```
iex> {A, B} = {1, 2}
{1, 2}
```

hello.ex

```
defmodule Hello do
  def say(1), do: "one"
  def say(2), do: "two"
  def say("hello"), do: "world"
end

Hello.say(1) #=> "one"
Hello.say(2) #=> "two"
Hello.say("hello") #=> "world"
```

|> Pattern matching (из реального приложения) ❤️

```elixir
defmodule M3.API.V1.ProfileController do
  ...

  def update(conn, %{"private" => "false"}, %{current_user: current_user}) do
    current_user |> M3.MakePublicProfile.call |> user_response(conn)
  end

  def update(conn, params, %{current_user: current_user}) do
    current_user
    |> User.changeset(params)
    |> Repo.update()
    |> user_response(conn)
  end
end
```

|> Pipe Operator (`|>`) 🙏

Можем переписать:

```
foo(bar(baz(new_function(other_function()))))
```

Так:

```
other_function() |> new_function() |> baz() |> bar() |> foo()
```

*передает в следующую функцию в цепочке первым аргументом результат выполнения предыдущей

|> Pipe Operator , ближе к практике

```
  def search(current_user, name, page) do
    name = name |> String.trim

    current_user
      |> build_subquery(name)
      |> build_query(name)
      |> paginate(page)
      |> Repo.all
  end
```

практический пример: код построение SQL запроса

|> Документация 📘

`Elixir treats documentation as a first-class citizen` - из доки

```elixir
defmodule IRC.Command do
  @doc ~S"""
  Парсит IRC комманду в кортеж.
    iex> IRC.Command.parse("USER guest 0 * :Ronnie Reagan")
    {:ok, {:user, "guest", "0", "Ronnie Reagan"}}

    iex> IRC.Command.parse("USER 2340-230489 923")
    {:ok, :user}
  """

  def parse(line) do
    line
    |> String.split
    |> case do
         ...
  end
end
```

то что в @doc попадает в ExUnit и там выполняется

|> То есть 🟦

Пишем документацию и одновременно тесты, говорят это удобно. Так же практикуется в Python.

|> Pattern matching, atoms, pipe operator

```
socket
|> read_line()
|> Command.parse
|> case do
     {:ok, command} ->
       case Command.run(session, command) do
         {:error, reply} -> Reply.error(reply)
         {:ok, reply} -> Reply.reply(reply)
         :ok -> Reply.reply
       end
     {:error, reply} -> Reply.error(reply)
   end
|> write_line(socket)
```

if, case, for - это тоже функции

|> METAPROGRAMMING

|> Пример матчера для ExUnit

```elixir
defmodule M3.Matchers.Change do
  ...
  defmacro changes(action, [from: from, to: to] = params, do: block) do
    quote do
      [from: from, to: to] = [unquote_splicing(params)]
      before = unquote(block)
      assert before == from
      unquote(action)
      after_ = unquote(block)
      assert to == after_
    end
  end
end

# пример использования

visitor
|> Complaints.create(post, @complaint_params)
|> changes from: [post.id], to: [] do
   visitor |> Posts.search() |> Enum.map(& &1.id)
 end
```

|> Пример построения и сложного SQL запроса на ruby

```ruby
  def composed_articles_list
    scope = articles_table.
      join(issues_table).on(articles_table[:issue_id].eq(issues_table[:id])).
      join(journals_table).on(issues_table[:journal_id].eq(journals_table[:id])).
      project(issues_table[:year].as('article_year'), issues_table[:journal_id],

    scope = apply_filters(scope)

    Arel::Nodes::As.new(cte_articles_list, scope)
  end
```

*кто работал с Arel, в цирке не смеется

|> Пример построение SQL запроса на Ecto 📝

```
  defp build_query(subquery, name) do
    from l in Location,
         left_join: searches in subquery(subquery), on: searches.id == l.id,
         left_join: posts in Post, on: posts.location_id == l.id,
         where: ilike(l.name, ^"%#{name}%") and is_nil(l.deleted_at),
         select: %{ l |
           recently_searched: fragment("? IS NOT NULL as recently_searched", sea
           post_count: fragment("COUNT(?) as post_count", posts.id)
         },
         order_by: [desc: fragment("recently_searched"), desc: fragment("post_cou
         group_by: [l.id, searches.id]
  end
```

*на самом деле кто пишет на Ecto, тоже в цирке не смеется

|> ActiveRecord? Fuck you! Hello DDD and Contexts

Примеры:

- RegisterUser.call(...)
- SearchLocation.call(...)
- CreateOrder.call(...)

всегда понятно что делают, абстрагируют остальной код от работы с базой данных, быстрые, легко тестируются

|> Ничего не напоминает?

подсказка: Callable Object, ServiceObject (ну или Actor, привет Александр)

|> Пример из Ruby

```ruby
class CreateOrder
  def initialize(publisher, articles)
    ...
  end

  def call
    order = Order.create!(publisher: publisher)
    CreateOrderItems.new(
      article_ids: articles.pluck(:id),
      default_doi_price: default_doi_price,
      order_id: order.id
    ).call
    publish(:order_created, performer: publisher.account, order: order)
    order
  end

  private

  attr_reader :publisher, :articles, :default_doi_price
end
```

|> Пример из Elixir

```elixir
defmodule CreateSearch do
  ...
  def call(user, entity), do: user |> create_or_bump(entity)

  defp create_or_bump(%User{} = owner, %Post{} = post) do
    owner
    # pipe operator!
    |> find_search(post)
    |> Repo.one
    |> case do
         nil -> owner |> create_search(post)
         record -> bump(record)
       end
  end
  # pattern matching!
  defp create_or_bump(%User{} = owner, %User{} = user) do
    ...
  end
end
```

|> router.ex

Просто пример хорошего DSL:

```
defmodule M3.API.Router do
  use M3.API, :router
  ...
      get "/auto_tags", AutoTagController, :index
      get "/user_tags", UserTagController, :index

      get "/locations", LocationController, :index

      get "/complaints/subtypes", ComplaintController, :subtypes
      resources "/complaints", ComplaintController, only: [:index, :update, :sho

      scope "/searches", alias: Searches, as: :searches do
        resources "/posts", PostController, only: [:create, :index]
        resources "/users", UserController, only: [:create, :index]
        resources "/locations", LocationController, only: [:create, :index]
        resources "/user_tags", UserTagController, only: [:create, :index]
      end
  ...
end
```

|> При этом все очень быстро 🚶

- язык компилируемый
- нет долго маппинга результатов SQL в объекты, потому что объектов нет, есть только хэши и подобные им структуры

|> Итого

- если вы понимаете MVC, то никаких проблем с освоением Phoenix не возникнет
- более правильна архитекрура, запросы проходят за миллисикунды

|> Но

- язык очень молодой, многие фичи могут быстро появляться и исчезать, проект нужно постоянно апдейтить
- более сложная схема деплоя

|> И это только начало, да здравствует OTP ⚡

|> Неважно какой язык 👑

На самом деле Elixir/Erlang - это только инструменты для работы с OTP. Код из elixir можно вызывать из erlang и наоборот.

Что это такое 😕

- виртуальная машина
- встроенная СУБД Mnesia
- ets (встроенный из короки Redis)
- RPC из коробки, очень просто построить кластер
- отладчик
- процессы можно визуально отлаживать
- взаимодействие с другими языками
- слабая типизация путем описания контрактов между функциями (отдельно)

Важно правильно выстроить взаимодействие процессов 🚦

Процессы делятся на два типа:

- процессы которые что-то делают
- процессы которые следят за другими процессами и в случае их падения быстро перезапускат (или нет)

Процесс 🚆

- легковесный (от 2кб)
- stateless - легко перезапустить, ведь все данные иммутабельны
- никаких race conditions, все данные разделены
- может понимать что код был изменен и в этом случаче умеет изменять стейт под новый код (есть кобек)

Супервизор (тоже процесс): 🚉

- обладает стратегией перезапуска дочерних процессов в случае их падения
- может пересоздавать процесс с таким же стейтом каким он был при падении

Трушное ООП 👽

- процесс - как запущенный инстанс объекта
- message-passing - можно утрировать как вызов методов
- behaviour - схож с интерфейсами

Инструменты 🔌

Редакторы:

- Atom
- IDEA
- Visual Studio Code

Инструменты:

- Credo – линтер
- Dogma – линтер
- Dialyzer – тайпчекер
- IEx – дебагер

Обасть применения 🕶️

- Существует точка зрения что Erlang был написан, чтобы паралелить вызовы к библиотекам написанным на C, виртуальная машина не оптимизирована для вычислений
- Идеально подходит для передачи или тиражирования информации: чаты, веб-сокеты, сервера онлай-игр, очереди сообщений (привет RabbitMQ), API

В нашем случае, где стоит использовать:

- Написание API для мобильных приложений
- Обслуживание вебсокетов

Где не стоит использовать:

- админки
- статические сайты где много серверного рендеринга

Что мне прочитать и куда двигаться дальше? ☀️

- Изучай Erlang во имя добра! - моя настольная книга!
- Очень-очень-очень хороший курс по Erlang от разрабочтика от World of Tanks
- Erlang Programming - книга одного из авторов языка, поговаривают, что она писалась под воздейсвтием психотропных веществ, лучше оставить на потом
- WUNSH, телеграм
- Телеграм канал по Elixir и Phoenix

Пробовал, написал свой блог и забил. 😲

Что дальше делать?

🔥🔥🔥 УГАРАТЬ 🔥🔥🔥

Давайте напишем свой IRC сервер?

- спецификация протокола IRC открыта, бери и имплемнтируй
- протокол текстовый, легко дебажить (telnet!)
- программирование на сокетах, хардкор, асинхронность

репозиторий уже есть: https://github.com/r00takaspin/exircd

Процесс разработки

- код ревью
- покрытие тестами
- документирование
- делает релизы
- непрерывная инеграция

ВОПРОСЫ

ссылка на презентацию: https://github.com/r00takaspin/gdg-meetup-17.02.2018