Functional && Concurrent programming with **Elixir**

fewlines

@kdisneur

Me

- Elixir developer
- Work at **fewlines**
- @kdisneur on Twitter & GitHub

fewlines

- Tech education for executives
- Build API first software

Elixir

- Modern language (2012)
- Run on the old Erlang VM (1986)
- Built for concurrence
- Built for fault tolerance
- Built for distribution

You can join us on Meetup
https://www.meetup.com/fr-FR/Lille-Elixir/

Functional

We are not going to dive really deep in the functional world, just the bare minimum to understand Elixir.

- Immutability
- Functions
- Pattern matching

Immutability

- **All** variables are immutable
- No global state
- No mutation, only transformation

Immutability

```elixir
defmodule MyMap do
  def add_keys_to_map(my_map) do
    my_map
    |> Map.put(:a_key, 42)
    |> Map.put(:another_key, 1337)
  end
end

original_map = %{foo: "bar"}
new_map = MyMap.add_keys_to_map(original_map)

IO.inspect(original_map)
# => %{foo: "bar"}
IO.inspect(new_map)
# => %{a_key: 42, another_key: 1337, foo: "bar"}
```

Functions

- Anonymous functions (lambda)
- High-order functions
- Functions are first class citizen

Functions

```elixir
sum = fn x, y -> x + y end
sum.(1, 2)
# => 3

Enum.reduce([1, 2, 3], 0, sum)
# => 6

Enum.reduce([1, 2, 3], 0, &Kernel.+/2)
# => 6
```

Pattern Matching

- Like a switch statement but on steroid 💪
- Destructuring assignment
- Available in function definitions
- Available in case statements

Pattern Matching

```elixir
defmodule Person do
  defstruct name: nil, age: nil, city: nil
end

defmodule Greeting do
  def extract_name(%Person{name: nil}) do
    {:error, :name_not_found}
  end
  def extract_name(%Person{name: name}) do
    {:ok, name}
  end
end

extract_name(%Person{age: 28, name: "Kevin"})
# => {:ok, "Kevin"}
extract_name(%Person{age: 28, city: "Lille"})
# => {:error, :name_not_found}
```

Pattern Matching

```elixir
defmodule Greeting do
  def say_hello(person) do
    case extract_name(person) do
      {:ok, name} -> "Hello #{name}!"
      {:error, :name_not_found} -> "Hello stranger"
    end
  end

  # Previously defined functions here
end

Greeting.say_hello(%Person{age: 28, name: "Kevin"})
# => "Hello Kevin!"
Greeting.say_hello(%Person{age: 28, city: "Lille"})
# => "Hello stranger"
```

Now we know about functional...

Let's talk concurrency

Concurrency

Definition

Several execution environment

- Run at the same time
- Run on the same VM
- Can interact with each other
- Use all CPU cores available

Why concurrency sucks

Let's imagine a system where

- An order contains a fix amount of loyalty points
- A customer has several orders
- A customer loyalty points is the sum of the loyalty points of its orders

Now imagine this feature

- Compute and display the customer loyalty points

Why concurrency sucks

Imagine the Customer has 10 Orders for
a total of 55 loyalty points.

```
class Customer {
  private int loyaltyPoints;

  public void computeLoyaltyPoints() {
    final List<Order> orders = OrderRepository.fetchAll(this);
    this.loyaltyPoints = 0;

    for (final Order order : orders) {
      this.loyaltyPoints += order.getLoyaltyPoints();
    }
  }
}
```

http://bit.ly/2r7juYq

Why concurrency sucks

I added a second thread and suddenly
my customer loyalty points were inconsistent

```
 Functional & Concurrent Programming with Elixir ツ java Main
95
 Functional & Concurrent Programming with Elixir ツ java Main
74
 Functional & Concurrent Programming with Elixir ツ java Main
109
 Functional & Concurrent Programming with Elixir ツ java Main
96
 Functional & Concurrent Programming with Elixir ツ java Main
55
 Functional & Concurrent Programming with Elixir ツ java Main
74
 Functional & Concurrent Programming with Elixir ツ java Main
107
 Functional & Concurrent Programming with Elixir ツ java Main
95
```

reminder: expected result was 55

Why concurrency sucks

Problems?

- No guarantee to be the only one accessing the data
- The outcome can be a really "unexpected" result

Solutions?

- Locks/Mutex

New problems?

- Adds a lot of complexity
- Can lead to dead locks
- No guarantee to have created all the required ones

How does it **work** with **Elixir**?

Concurrency

Process

From now on, we won't talk about OS but Elixir processes.

- Lightweight thread of Erlang VM (1 or 2 KB)
- Isolated from each other (data & crash)
- Can have an internal state
- Can only receive messages (no access to state)
- Can receive one message at a time (blocking)

Concurrency
GenServer

- Abstraction on top of Process
  - Handle incoming messages (mailbox)
  - Handle the state stored inside the Process
  - Handle the message type (sync / async)

How it works under the hood: http://bit.ly/2rpqXQZ

Concurrency

If you remember we told you earlier:

A Process is **isolated**. Data and crash wise.

OK... So, how do we deal with failures?

Concurrency

Failures

Defensive programming? No, thanks!

- Make code harder to comprehend
- Can't catch all potential errors
- Catching an exception can lead to an unexpected state

We prefer to **supervise** processes and
restart to a **known state**

Concurrency

Supervisor

- Start some Processes
- Monitor some Processes
- Define a strategy to apply when Processes fail

Concurrency

Crawler app

We want:

- A form where we can input a website
- For each website, we start a new crawler
- A crawler can crawl several URLs at the same time
- A crawler reports in "real time" the new URLs found

Crawler app

History

Store root URL + visited URLs

Know next batch of URLs to fetch

Worker

Ask for new batch of URLs

Ask to fetch URLs

Save new URLs found

Fetcher
x10

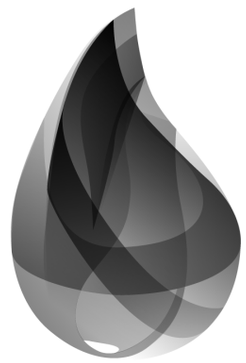
Fetch an URL

Find all URLs inside the fetched page

Conclusion

Well... concurrency doesn't have to be tough!

Pick the right tool



elixir

Thank You 🤦

Twitter @kdisneur

Lille Elixir
Meetup https://www.meetup.com/fr-FR/Lille-Elixir/
Slack https://lille-elixir.herokuapp.com

Lille FP
Meetup https://www.meetup.com/fr-FR/Lille-FP/
Slack https://slackin-lillefp.herokuapp.com/

Rebuild a GenServer http://bit.ly/2rpqXQZ

Crawler repository http://bit.ly/2qFYYPU