Einführung in

RDF on Elixir

Marcel Otto

Agenda

1. Elixir-Crashkurs
2. RDF on Elixir

Was ist Elixir?

elixir

a dynamic, functional language designed for building scalable and maintainable applications.

Läuft auf derselben VM wie Erlang - die BEAM.

Syntax stark von Ruby beeinflusst.

Semantik jedoch sehr nah an Erlang.

Warum Elixir?

- Skalierbarkeit (über Cores und Maschinen)

Warum Elixir?

- Skalierbarkeit (über Cores und Maschinen)
- Stabilität (Ausfallsicherheit und Verfügbarkeit)

Warum Elixir?

- Skalierbarkeit (über Cores und Maschinen)
- Stabilität (Ausfallsicherheit und Verfügbarkeit)
- Wartbarkeit (des Codes)

Warum Elixir?

- Skalierbarkeit (über Cores und Maschinen)
- Stabilität (Ausfallsicherheit und Verfügbarkeit)
- Wartbarkeit (des Codes)
- Produktivität (der Entwickler)

Warum Elixir?

- Skalierbarkeit (über Cores und Maschinen)
- Stabilität (Ausfallsicherheit und Verfügbarkeit)
- Wartbarkeit (des Codes)
- Produktivität (der Entwickler)
- Tooling

Warum Elixir?

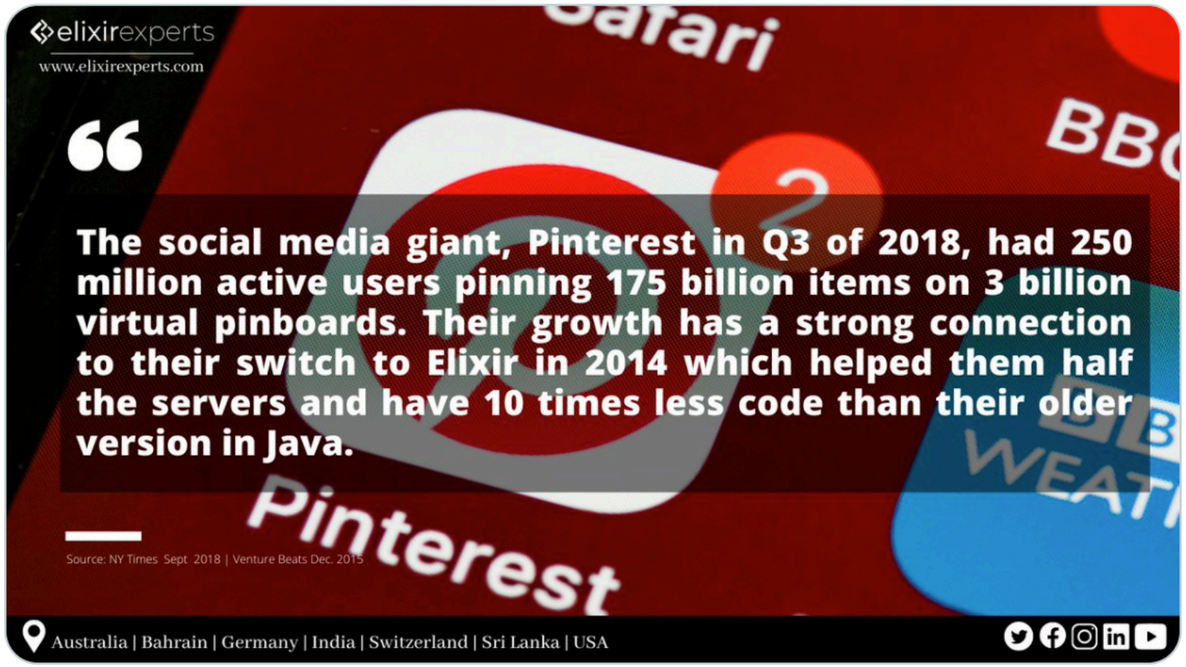
- Skalierbarkeit (über Cores und Maschinen)
- Stabilität (Ausfallsicherheit und Verfügbarkeit)
- Wartbarkeit (des Codes)
- Produktivität (der Entwickler)
- Tooling
- Erlernbarkeit

elixirexperts
@elixir_experts

#Pinterest halved their servers & has 10 times less code now, as they switched to #elixir from #Java.

#elixirexperts #Facts #Elixirlang #design #uiux #deployment #technology #cloud #blockchain #AI #IoT #integration #services #SocialMedia #Application #developement #agile #scrum

Tweet übersetzen



Iteron Tech und 3 weitere Personen

12:21 nachm. · 18. März 2021 · Twitter Web App

1. Elixir-Crashkurs

Elixir: Datentypen

```
# Strings
"Foo"

# Integers
42

# Floats
3.14

# Booleans
true
false

# Atoms
:foo
```

Elixir: Datenstrukturen

```elixir
# Tuples
{42, 3.14, "Foo"}

# Lists
[42, 3.14, "Foo"]

# Keyword lists
[foo: 1, bar: 3.14] # == [{:foo, 1}, {:bar, 3.14}]

# Maps
%{:foo => 1, :bar => 3.14}
%{foo: 1, bar: 3.14}

# Structs
%User{name: "Max", email: "max@mustermann.de"}
```

Elixir: Sigils

```
# Strings
~s(this is a string with "double" quotes)

# Regular expressions
~r/foo|bar/

# Word lists
~w(foo bar bat) # == ["foo", "bar", "bat"]

# Dates
~D[2019-10-31]
```

Elixir: Pattern-Matching

```
i = 42

{:ok, result} = Some.fun(42)

%{foo: [value | _]} = %{foo: [1, 2, 3]}
```

Elixir: Funktionen

```
square = fn x -> x * x end
square.(2)

div = fn
  x, 0 -> nil
  x, y -> x / y
end
```

Elixir: Module

```
defmodule Math do
  def fib(0), do: 0

  def fib(1), do: 1

  def fib(n) do
    fib(n-1) + fib(n-2)
  end
end

Math.fib(23)
```

Elixir: Module

```elixir
defmodule Some.Deeply.Nested.Module do
  def hello(), do: "Hello"
end

defmodule Some.Other.Module do
  alias Some.Deeply.Nested.Module, as: Module

  def greet(name) do
    IO.puts("#{Module.hello()} #{name}")
  end
end

alias Some.Other.Module
Module.greet("World")
```

Elixir: Pipe-Operator

```
Math.fib(42) == 2 |> Math.fib()

String.replace(String.downcase(String.trim(name)), " ", "-")

name
|> String.trim()
|> String.downcase()
|> String.replace(" ", "-")
```

Elixir: Enum

```
[1, 2, 3]
|> Enum.map(fn x -> x * 2 end)

%{a: "foo", b: "bar"}
|> Enum.map(fn
     {key, value} when is_binary(value) ->
       {key, String.upcase(value)}

     other -> other
end)
```

Elixir: Makros

```
defmodule Sample.Weather do
  use Ecto.Schema

  schema "weather" do
    field :city
    field :temp_lo, :integer
    field :temp_hi, :integer
    field :prcp,    :float, default: 0.0
  end
end
```

2. RDF on Elixir

RDF on Elixir

- RDF.ex
- SPARQL.ex
- ShEx.ex
- Grax

RDF.ex

- RDF nodes
- RDF graph structures (in-memory)
- basic graph queries (BGP)
- RDF serializations

RDF.ex: Knoten

- `RDF.IRI`
- `RDF.BlankNode`
- `RDF.Literal`
- `RDF.Term`

RDF.ex: URIs

```elixir
# mit Konstruktorfunktion
RDF.iri("http://www.example.com/foo")

# mit Sigils
import RDF.Sigils

~I<http://www.example.com/foo>
```

RDF.ex: URIs mittels Vocabulary Namespaces

```
alias RDF.NS.{RDFS, OWL, SKOS, XSD}

RDFS.subClassOf
# => ~I<http://www.w3.org/2000/01/rdf-schema#subClassOf>

RDF.iri(RDFS.Class)
# => ~I<http://www.w3.org/2000/01/rdf-schema#Class>
```

RDF.ex: Description DSL

```
EX.Foo
|> RDF.type(EX.Bar)
|> EX.baz(1, 2, 3)

# =>
# #RDF.Description<
#   <http://example.com/Foo>
#       a <http://example.com/Bar> ;
#       <http://example.com/baz> 1, 2, 3 .
# >
```

RDF.ex: Vocabulary Namespaces

```elixir
defmodule YourApp.NS do
  use RDF.Vocabulary.Namespace

  defvocab EX,
    base_iri: "http://www.example.com/",
    file: "example_vocabulary.ttl"
end

alias YourApp.NS.{EX}
EX.foo
```

RDF.ex: Literals

```
# untyperisiertes Literal mit Konstruktorfunktion
RDF.literal("foo")

# untyperisiertes Literal mit Sigils
~L"foo"
```

RDF.ex: Literals

```
# language-tagged Literal mit Konstruktorfunktion
RDF.literal("foo", language: "en")

# language-tagged Literal mit Sigils
~L"foo"en
```

RDF.ex: Literals

```elixir
# typerisiertes Literal mit generischer Konstruktorfunktion
RDF.literal("foo", datatype: "http://example.com/datatype")

alias RDF.NS

RDF.literal("42", datatype: NS.XSD.integer)
```

RDF.ex: Literals

```elixir
alias RDF.XSD

XSD.integer("42")

XSD.unsignedInteger(42)

XSD.date("2021-04-06")
```

RDF.ex: Literals

```
RDF.literal(true)

RDF.literal(42)

RDF.literal(3.14)

RDF.literal(~D[2021-04-06])
```

Elixir datatype	XSD datatype
string	xsd:string
boolean	xsd:boolean
integer	xsd:integer
float	xsd:double
Decimal	xsd:decimal
Time	xsd:time
Date	xsd:date
DateTime	xsd:dateTime
NaiveDateTime	xsd:dateTime
URI	xsd:AnyURI

RDF.ex: Data structures

Basis-Datenstrukturen:

- `RDF.Statement`
- `RDF.Description`
- `RDF.Graph`
- `RDF.Dataset`

Weitere:

- `RDF.List`
- `RDF.Diff`

RDF.ex: Statements

```
RDF.triple({EX.S, EX.p, EX.O})
# => {~I<http://www.example.com/S>, ~I<http://www.example.com/p>, ~I<http://www.example.com/O>}

RDF.triple({EX.S, EX.p, "foo"})
# => {~I<http://www.example.com/S>, ~I<http://www.example.com/p>, ~L"foo"}

RDF.triple({EX.S, EX.p, 42})
# => {~I<http://www.example.com/S>, ~I<http://www.example.com/p>, XSD.integer(42)}
```

RDF.ex: Descriptions

```
description =
  EX.S1
  |> EX.p1(EX.O1)
  |> EX.p2("Foo", 42)

#RDF.Description<
  <http://example.com/S1>
      <http://example.com/p1> <http://example.com/O1> ;
      <http://example.com/p2> 42, "Foo" .
>
```

RDF.ex: Graphs

```
graph =
  RDF.graph()
  |> RDF.Graph.add(description)
  |> RDF.Graph.add({EX.S1, EX.p1, EX.O2})
  |> RDF.Graph.add(RDF.graph({EX.S2, EX.p2, "Bar"}))
```

```
#RDF.Graph<name: nil
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

  <http://example.com/S1>
      <http://example.com/p1> <http://example.com/O1>, <http://example.com/O2> ;
      <http://example.com/p2> 42, "Foo" .

  <http://example.com/S2>
      <http://example.com/p2> "Bar" .
>
```

RDF.ex: Graphs

```
graph
|> Enum.map(fn
     {s, p, %RDF.Literal{literal: %XSD.String{value: string}}} ->
       {s, p, String.upcase(string)}

     triple -> triple
   end)
|> RDF.graph(prefixes: [ex: EX])


#RDF.Graph<name: nil
  @prefix ex: <http://example.com/> .

  ex:S1
      ex:p1 ex:O1, ex:O2 ;
      ex:p2 42, "FOO" .

  ex:S2
      ex:p2 "BAR" .
>
```

RDF.ex: Serializations

Unterstützte Formate:

- N-Triples
- N-Quads
- Turtle
- JSON-LD (JSON-LD.ex)
- RDF/XML (RDF-XML.ex)

```
EX.S
|> EX.p()
|> RDF.Turtle.write_string()

"example.nt"
|> RDF.NTriples.read_file!()
|> RDF.Graph.add(additional_statements)
|> RDF.XML.write_file("example.rdf")

"example.nq"
|> RDF.read_file!()
|> RDF.write_file("example.jsonld")
```

RDF.ex: Queries

```
graph
|> RDF.Graph.query([
    {:s?, RDFS.label, "Foo"},
    {:s?, :p?, :o?}
   ])
|> Enum.take(3)
```

```
[
  %{
    o: ~I<http://example.com/O1>,
    p: ~I<http://example.com/p1>,
    s: ~I<http://example.com/S1>
  },
  %{
    o: ~I<http://example.com/O2>,
    p: ~I<http://example.com/p1>,
    s: ~I<http://example.com/S1>
  },
  %{
    o: XSD.integer(42),
    p: ~I<http://example.com/p2>,
    s: ~I<http://example.com/S1>
  }
]
```

SPARQL.ex

- SPARQL Query Engine[1] über RDF.ex Graphen
- SPARQL Client in separatem Projekt (SPARQL.Client)

[1] Implementierung der SPARQL 1.1 specs noch nicht vollständig

SPARQL.ex

```
"""
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
"""
|> RDF.Turtle.read_string!()
|> SPARQL.execute_query("""
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     SELECT ?name ?mbox
     WHERE
       { ?x foaf:name ?name .
         ?x foaf:mbox ?mbox }
     """)
```

```
%SPARQL.Query.Result{
  variables: ["name", "mbox"],
  results: [
    %{
      "mbox" => ~I<mailto:peter@example.org>,
      "name" => ~L"Peter Goodguy"
     },
    %{
      "mbox" => ~I<mailto:jlow@example.com>,
      "name" => ~L"Johnny Lee Outlaw"
     }
  ]
}
```

SPARQL.ex

```
SPARQL.execute_query(graph, """
  PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
  PREFIX schema: <http://schema.org/>
  CONSTRUCT
    { ?x schema:name ?name ;
         schema:email ?mbox }
  WHERE
    { ?x foaf:name ?name ;
         foaf:mbox ?mbox }
  """)
```

```
#RDF.Graph<name: nil
  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
  @prefix schema: <http://schema.org/> .
  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

  [
      schema:email <mailto:peter@example.org> ;
      schema:name "Peter Goodguy"
  ] .

  [
      schema:email <mailto:jlow@example.com> ;
      schema:name "Johnny Lee Outlaw"
  ] .
>
```

SPARQL.Client

```
# Places with free wi-fi from Wikidata
"""
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX bd: <http://www.bigdata.com/rdf#>

SELECT ?item ?itemLabel (SAMPLE(?coord) AS ?coord)
WHERE {
    ?item wdt:P2848 wd:Q1543615 ;  # wi-fi gratis
          wdt:P625 ?coord .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en" }
} GROUP BY ?item ?itemLabel
LIMIT 100
"""
|> SPARQL.Client.query("https://query.wikidata.org/bigdata/namespace/wdq/sparql")

{:ok,
 %SPARQL.Query.Result{
   variables: ["item", "itemLabel", "coord"],
   results: [
     %{
       "coord" => %RDF.Literal{literal: %RDF.Literal.Generic{datatype: "http://www.opengis.net/ont/geosparql#wktLiteral", value: "Point(99.032395 7.576717)"}, valid: true},
       "item" => RDF.iri("http://www.wikidata.org/entity/Q59156498"),
       "itemLabel" => %RDF.Literal{literal: %RDF.XSD.String{value: "Q59156498", lexical: "Q59156498"}, valid: true}
     },
     ...
   ]
 }}
```

ShEx.ex

```
graph =
  RDF.Turtle.read_string!("""
    PREFIX ex: <http://ex.example/#>
    PREFIX inst: <http://example.com/users/>
    PREFIX school: <http://school.example/#>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>

    inst:Alice foaf:age 13 ;
      ex:hasGuardian inst:Person2, inst:Person3 .

    inst:Bob foaf:age 15 ;
      ex:hasGuardian inst:Person4 .

    inst:Claire foaf:age 12 ;
      ex:hasGuardian inst:Person5 .

    inst:Don foaf:age 14 .
    """)

shape =
  ShEx.ShExC.decode!("""
    PREFIX ex: <http://ex.example/#>
    PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
    PREFIX school: <http://school.example/#>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>

    school:enrolleeAge xsd:integer MinInclusive 13 MaxInclusive 20

    school:Enrollee {
      foaf:age @school:enrolleeAge ;
      ex:hasGuardian IRI {1,2}
    }
    """)
```

```
graph
|> ShEx.validate(shape,
  %{
    ~I<http://example.com/users/Alice>
      => ~I<http://school.example/#Enrollee>,
    ~I<http://example.com/users/Bob>
      => ~I<http://school.example/#Enrollee>,
    ~I<http://example.com/users/Claire>
      => ~I<http://school.example/#Enrollee>,
    ~I<http://example.com/users/Don>
      => ~I<http://school.example/#Enrollee>,
  }
)
|> Enum.each(fn association ->
  IO.puts("#{inspect association.node} is #{association.status}")
end)
```

```
~I<http://example.com/users/Alice> is conformant
~I<http://example.com/users/Bob> is conformant
~I<http://example.com/users/Claire> is nonconformant
~I<http://example.com/users/Don> is nonconformant
```

Grax

```elixir
defmodule User do
  use Grax.Schema

  alias NS.{SchemaOrg, FOAF}

  schema SchemaOrg.Person do
    property :name, SchemaOrg.name, type: :string, required: true
    property :emails, SchemaOrg.email, type: list_of(:string)
    property :age, FOAF.age, type: :integer

    field :password

    link :friends, FOAF.friend, type: list_of(User)
    link :posts, -SchemaOrg.author, type: list_of(Post)
  end
end
```

Grax

```
User.load(graph, EX.User1)

{:ok,
 %User{
   __id__: ~I<http://example.com/User1>,
   age: 42,
   email: ["jane@example.com", "jane@work.com"],
   friends: [],
   name: "Jane",
   password: nil,
   posts: [
     %Post{
       __id__: ~I<http://example.com/Post1>,
       author: #Grax.Link.NotLoaded<link :author is not loaded>,
       content: "Lorem ipsum ...",
       title: "Lorem"
     }
   ]
 }}
```

Elixir Appetizer

- Phoenix
- Phoenix LiveView
- Nerves
- Flow & Broadway
- Nx

Fragen?

BEAM

Weitere Sprachen auf der BEAM:

- Erlog (Prolog)
- LFE (Lisp)
- Luerl (Lua)
- Gleam

Weitere Implementierungen der BEAM:

- Lumen
- Erlscripten