

Macros in Elixir

by Mieszko Wawrzyniak

What are macros?

The elixir macros are used to generate code during compilation

Abstract Syntax Tree

```
quote do: foo(arg1, arg2)
{:foo, [], [
  {:arg1, [], Elixir},
  {:arg2, [], Elixir}
]}
```

```
quote do: %{a: 1, b: 2}
{::%{}, [], [a: 1, b: 2]}
```

```
quote do: Enum.to_list(%{a: 1, b: 2})
{
  {
    :.,
    [],
    [
      {:__aliases__, [alias: false], [:Enum]},
      :to_list
    ]
  },
  [],
  [{:%{}, [], [a: 1, b: 2]}]
}
```

Our first macro

```
quote do: %{a: 1, b: 2}
{: %{}}, [], [a: 1, b: 2]
```

```
defmodule First do
  defmacro to_map(keyword) do
    {: %{}}, [], keyword
  end
end
```

```
iex(1)> require First
First
iex(2)> [a: 1, b: 2] \
... (2)>   ▷ First.to_map() \
... (2)>   ▷ Macro.to_string() \
... (2)>   ▷ IO.puts()
%{a: 1, b: 2}
```

Macros can generate macros

```
defmodule AssertionGenerator do
  defmacro defassert(operator, err) do
    quote do
      defmacro assert({unquote(operator), _, [lhs, rhs]} = expr) do
        err = unquote(Macro.escape(err))

        quote do
          unquote(expr) ||
            unquote(err).(unquote(lhs), unquote(rhs))
          ▷ IO.puts()
        end
      end
    end
  end
end
end
end
```

```

defmodule Assertion do
  import Assertion.Generator

  defassert(:=, &"#{&1} is not equal to #{&2}")
  defassert(:≠, &"#{&1} is equal to #{&2}")
  defassert(:≤, &"#{&1} is greater than #{&2}")
end

```

```

defmodule Second do
  import Assertion

  def main do
    assert 2 + 2 = 4
    assert 2 + 2 = 5
    assert 2 + 2 ≠ 5
    assert 2 + 2 ≠ 4
    assert 2 + 2 ≤ 4
    assert 2 + 3 ≤ 3
  end
end

```

```

iex(1)> Second.main
4 is not equal to 5
4 is equal to 4
5 is greater than 3

```

But not always everything will work as expected


```
iex(1)> "a" <> b <> "c" = "some string"
```

```
** (ArgumentError) the left argument of <> operator inside a match should be  
always a literal binary as its size can't be verified, got: b
```

```
(elixir) lib/kernel.ex:1759: Kernel.invalid_concat_left_argument_error/1
```

```
(elixir) lib/kernel.ex:1731: Kernel.wrap_concatenation/3
```

```
(elixir) lib/kernel.ex:1710: Kernel.extract_concatenations/2
```

```
(elixir) lib/kernel.ex:1710: Kernel.extract_concatenations/2
```

```
(elixir) expanding macro: Kernel.<>/2
```

```
iex:1: (file)
```

```
defmodule Fail do
  defmacro matcher(args) do
    head =
      Enum.reduce(args, nil, fn
        text, nil when is_binary(text) → text
        param, nil → {param, [], Elixir}
        text, acc when is_binary(text) → quote do: unquote(acc) <> unquote(text)
        param, acc → quote do: unquote(acc) <> unquote({param, [], Elixir})
      end)

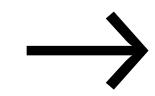
    quote do
      def match(unquote(head)), do: :ok
    end
  end
end
```

```
defmodule Third do
```

```
  import Fail
```

```
  matcher(["a", :b, "c"])
```

```
end
```



```
def(match(("a" <> b) <> "c")) do
```

```
  :ok
```

```
end
```

Domain Specific Language

```
defmodule MyAppWeb.Router do
  use Phoenix.Router

  pipeline :browser do
    plug :accepts, ["html"]
  end

  scope "/" do
    pipe_through :browser
    # browser related routes and resources
  end
end
```

```
defmodule DcmWeb.Eipa.Dictionary.ConnectorInterface do
  use Absinthe.Schema.Notation

  object :eipa_dictionary_connector_interface do
    field :id, non_null(:id)
    field :name, non_null(:string)
    field :description, non_null(:string)
  end
end
```

```
from(s in Suite,
      join: r in Reservation,
      on: s.id = r.suite_id,
      where: r.public_token = ^public_token,
      preload: [
        :address,
        :photos
      ]
    )
```

Have Fun 🎉

Code samples at Github

<https://github.com/kaaboaye/elixir-wroclaw-macro>



Mieszko Wawrzyniak
mieszkowaw@gmail.com
wawrzyniak.dev

