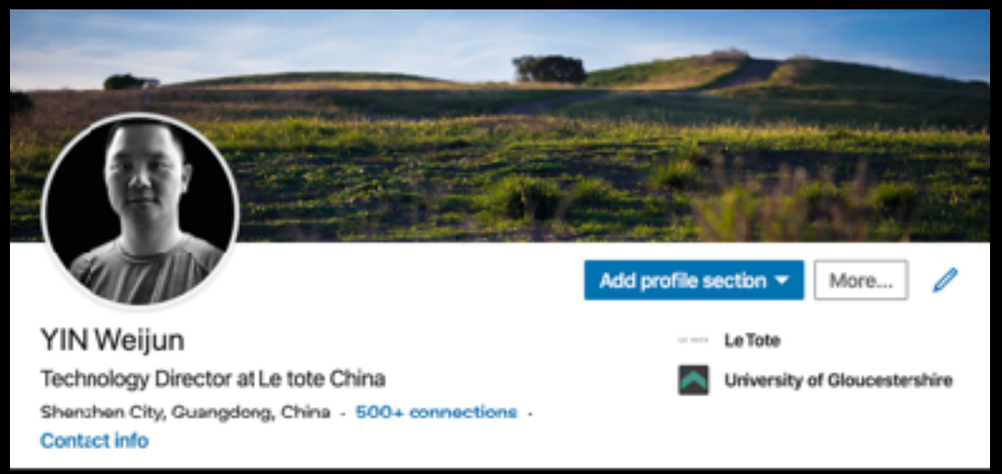
My Elixir Metaprogramming Journey

Two Practical Suggestions



尹伟君

linkedin.com/in/yin-weijun-93a08816/

Agenda

- When needs DSL then thinks deguard macro
- Macro from data perspective
- unquote

在Elixir 1.6 defguard macro出现前，如果你要定义使用到guard的DSL, 可以这么做:

```
  @doc false
  defmacro is_outofchina(lat, lng) do
    quote do
      (is_float(unquote(lng)) and (unquote(lng) < 72.004 or unquote(lng) > 137.8347)) or (is_floa
t(unquote(lat)) and (unquote(lat) < 0.8293 or unquote(lat) > 55.8271))
    end
  end

  @doc """
  Return true if given lat/lng within China "square" territory.

  ## Example


    iex> EvilTransform.Geo.outOfChina?(22.596828,114.120043)
    false
    iex> EvilTransform.Geo.outOfChina?(35.652832,139.839478)
    true


  """
  def outOfChina?(lat, lng) when is_outofchina(lat, lng) do
    true
  end
  @doc false
  def outOfChina?(lat, lng) when is_float(lat) and is_float(lng), do: false
```

Reference: github.com/weij/eviltransform

合理使用defguard能令业务判断更为清晰，容易理解容易维护

```elixir
defmodule UglyOrange.InventoryTransfer.Guards do

  defguard is_shipped(sent_at, sent_by) when not is_nil(sent_at) and not is_nil(sent_by)

  defguard is_received(received_at, received_by) when not is_nil(received_at) and not is_nil(received_by)

  defguard is_really_damaged_item?(item_state, input_opt) when item_state in ["damaged", "unmendable"] and input_opt == "damaged"

  defguard is_really_new_item?(item_state, input_opt) when item_state !== "damaged" and input_opt == "new"
end


  defp validate_item(item_barcode, input_opt, source_fc, destination_fc) do
    case Catalog.API.fetch_product_item(item_barcode) do
      {:ok, nil} ->
        {:error, "该条码无效"}

      {:ok, %{state: state, id: item_id}} when is_really_damaged_item?(state, input_opt) ->
        check_product_item(item_barcode, item_id, source_fc, destination_fc)

      {:ok, %{state: state, id: item_id}} when is_really_new_item?(state, input_opt) ->
        check_product_item(item_barcode, item_id, source_fc, destination_fc)

      {:ok, _} ->
        {:error, "该商品#{lookup(input_opt)}"}

      {:error, reason} ->
        {:error, reason}
    end
  end
```

数据驱动的编程方法 = 数据模型的使用

1. Import external file

```
@external_resource bin_number_path = Path.join([__DIR__, "../..", "bin_number.txt"])
```

2. Pattern match into variables

```
for line <- File.stream!(bin_number_path) do
  [weight, group, bin_num] =
    line
    |> String.split("|")
    |> Enum.map(&String.trim(&1))

  weight =
    case weight do
      "_" -> quote do: _
      _others -> weight
    end
```

3. Deal with special value

4. Combine into your functions

```
  def bin_number(unquote(weight), unquote(group)), do: unquote(String.to_integer(bin_num))
end

def bin_number(_, _), do: 0
```

5. Don’t forget catch-all clause

优点:

1. 不需要开发人员将业务部门的逻辑转变成为代码，业务部门的逻辑直接便是代码；
2. 代码和逻辑都在同一个文件内，便于传递和沟通交流；
3. 由于是Stateless方式，方便单元测试，提高安全性，运算性能高；
4. 适合不常改变其中逻辑的场景;

缺点：

1. 不合适经常改变参数的场景
2. 不支持通过调整某个参数值，而改变输出结果，需要做代码整体部署

What is Unquote?

“Unquotes the given expression inside a quoted expression.

This function expects a valid Elixir AST, also known as quoted expression, as argument. If you would like to unquote any value, such as a map or a four-element tuple, you should call Macro.escape/1 before unquoting.”

Unquote会解析传入的expression，然后inject valid quoted expression

```elixir
value =
  quote do
    13
  end

quote do
  sum(1, value, 3)
end
# => {:sum, [], [1, {:value, [], Elixir}, 3]}

value =
  quote do
    13
  end

  quote do
    sum(1, unquote(value), 3)
  end
  # => {:sum, [], [1, 13, 3]}
```

```elixir
# same as
# value = IO.puts(42)
expr = quote do
    unquote(IO.puts(42))
  end
  # => 42
  # => :ok

Code.eval_quoted(expr)
# => {:ok, []}
```

```elixir
quote do: IO.puts(42)

expr = quote do
  unquote({{:., [], [{:__aliases__, [alias: false], [:IO]}, :puts]}, [], '*'})
end

Code.eval_quoted(expr)
# => 42
```

1. Given expression in terms of two kinds - quoted expression, normal expression
2. Example 1,2,3

传入Macro的所有参数都是quoted expression 数据结构. 如果包含unquote会是怎样?

```elixir
defmodule Foo do

  # Example 1:
  # reuiqre Foo
  # Foo.bar(unquote(x))
  # => {:unquote, [line: 3], [{:x, [line: 3], nil}]}
  #
  # Example 2:
  # Foo.bar do: unquote(y)
  # => do: {:unquote, [line: 4], [{:y, [line: 4], nil}]}]
  defmacro bar(opts) do
    IO.inspect(opts)
    nil
  end

  # iex(8)> Foo.bar(unquote(x)) do
  #   ...(8)> unquote(y)
  #   ...(8)> end
  #   {:unquote, [line: 8], [{:x, [line: 8], nil}]}
  #   [do: {:unquote, [line: 9], [{:y, [line: 9], nil}]}]
  defmacro bar(head, body) do
    IO.inspect head
    IO.inspect body
    nil
  end

end
```

传入Macro的所有参数都是quoted expression 数据结构. 如果包含unquote会是怎样?

```elixir
defmodule M do
  defmacro m(x) do
    IO.inspect(x, label: __MODULE__)

    quote bind_quoted: [x: Macro.escape(x, unquote: true)] do
      def unquote(x)() do
        unquote(x)
      end
    end
    |> IO.inspect(label: "end of module M")
  end
end

# T.a => :a
# T.b => :b
# T.c => :c
defmodule T do
  import M

  for x <- ~w(a b c)a do
    m(unquote(x))
    #  Or, you don't need M module, and instead code follows
    #
    # def unquote(x)() do
    #   unquote(x)
    # end
  end
end
```

During code expansion...

```elixir
Elixir.M: {:unquote, [line: 21], [{:x, [line: 21], nil}]}
end of module M: {:__block__, [],
 [
  {:=, [], [{:x, [], M}, {:x, [line: 21], nil}]},
  {:def, [context: M, import: Kernel],
   [
    {{:unquote, [], [{:x, [], M}]}, [context: M], []},
    [do: {:unquote, [], [{:x, [], M}]}]
   ]}
 ]}
```

Reference: https://elixirforum.com/t/using-unquote-outside-of-quote-block/6179

https://elixirforum.com/t/using-unquote-outside-of-quote-block/6179

unquote vs Macro.escape/1

- Quote returns AST of passed in code block
- Macro.escape returns AST of passed in value

“ :unquote - when true, this function leaves unquote/1 and unquote_splicing/1 statements unescaped, effectively unquoting the contents on escape. This option is useful only when escaping ASTs which may have quoted fragments in them. Defaults to false. “

```
iex(4)> Macro.escape(
...>  {:unquote, [line: 21], [{:x, [line: 21], nil}]},
...>   unquote: true)
...>
...>  {:x, [line: 21], nil}
```

https://www.bignerdranch.com/blog/getting-started-with-elixir-metaprogramming/

Thank You