Porting a slugger library to Elixir

Julius Beckmann

March 30, 2016

Agenda

- Why?
- Slugs
- Coding
  - Elixir Strings
  - Iterating List
  - Pattern Matching
  - “Generating Code”
- Protocol

Why?

Learn **Elixir** & know the **Ecosystem**

Why?

Learn **Elixir** & know the **Ecosystem**

Something *valueable* that will *stay*.

Why?

Learn **Elixir** & know the **Ecosystem**

Something *valueable* that will *stay*.

Easy and enjoyable fun! :)

Slugs

“Ecto Version 2.0 released”

`/post/ecto-version-2-0-released`

Elixir Strings

We need to know how strings work in Elixir/Erlang.

Binary or Char List

A Elixir string is a UTF-8 **binary** *or* a **list of chars**.

Elixir docs on Strings

In Elixir, the word string means a UTF-8 binary and there is a String module that works on such data. Elixir also expects your source files to be UTF-8 encoded. On the other hand, string in Erlang refers to char lists and there is a :string module, that's not UTF-8 aware and works mostly with char lists.

Binaries

```
iex> <<104, 101, 108, 108, 111>>
"hello"
```

Charlists

```
iex> [104, 101, 108, 108, 111]
'hello'
```

Single chars

```
iex> ?h
104
iex> ?e
101
iex> ?l
108
iex> ?l
108
iex> ?o
111
```

Source project

PHP: javiereguiluz/EasySlugger

```php
<?php
function slugify($str) {
  $sep = '-';

  $str = trim(strip_tags($str));

  // Replacing 'ä' with 'ae'.
  $str = transliterate($str);

  $str = preg_replace("/[^a-zA-Z0-9\/_|+ -]/", '', $str);
  $str = preg_replace("/[\/_|+ -]+/", $sep, $str);
  $str = strtolower($str);

  return trim($str, $sep);
}
```

Module: String

```
iex> String.downcase "Elixir is Cool!"
"elixir is cool!"

iex> String.strip " Elixir is Cool! "
"Elixir is Cool!"

iex> String.replace "Elixir is Cool!", "Cool", "Cooler"
"Elixir is Cooler!"
```

Elixir Pipes

```
iex> s = " Ecto Version 2.0 released "
" Ecto Version 2.0 released "

iex> s |> String.strip |> String.downcase
"ecto version 2.0 released"
```

Elixir Pipes

```
iex> s = " Ecto Version 2.0 released "
" Ecto Version 2.0 released "

iex> s |> String.strip |> String.downcase
"ecto version 2.0 released"

iex> s |> String.strip |> String.downcase
        |> String.replace(~r/([^a-z0-9])+/, "-")
"ecto-version-2-0-released"
```

Transliterate

```php
<?php
// Replacing 'ä' with 'ae'.
$str = transliterate($str);
```

Transliterate

```php
<?php
// Replacing 'ä' with 'ae'.
$str = transliterate($str);
```

Replacing single chars?

```
iex> "äpfel" |> String.replace("ä", "ae")
"aepfel"
```

Transliterate

```php
<?php
// Replacing 'ä' with 'ae'.
$str = transliterate($str);
```

Replacing single chars?

```
iex> "äpfel" |> String.replace("ä", "ae")
"aepfel"
```

String.replace inside a loop will be too slow . . .

Transliterate

```php
<?php
// Replacing 'ä' with 'ae'.
$str = transliterate($str);
```

Replacing single chars?

```
iex> "äpfel" |> String.replace("ä", "ae")
"aepfel"
```

String.replace inside a loop will be too slow . . .

Lets do it in **one** iteration!

Iterating List

```
defp iterate([head|tail]) do
    IO.puts "Head:" ++ head

    # tail is always a list with
    # remaining elements or empty list.
    iterate(tail)
end

defp iterate([]) do
    IO.puts "End of list."
end
```

Iterating List - Example

- `iterate 'hello' => [?h | 'ello'] // Head:  h`

Iterating List - Example

- `iterate 'hello' => [?h | 'ello'] // Head: h`

- `iterate 'ello' => [?e | 'llo'] // Head: e`
- `iterate 'llo' => [?l | 'lo'] // Head: l`
- `iterate 'lo' => [?l | 'o'] // Head: l`
- `iterate 'o' => [?o | ''] // Head: o`

Iterating List - Example

- `iterate 'hello' => [?h | 'ello'] // Head: h`

- `iterate 'ello' => [?e | 'llo'] // Head: e`
- `iterate 'llo' => [?l | 'lo'] // Head: l`
- `iterate 'lo' => [?l | 'o'] // Head: l`
- `iterate 'o' => [?o | ''] // Head: o`

- `iterate '' // End of list.`

Real code

Iterating through a charlist without changing it.

```
defp replace_chars([h|t]), do: [h] ++ replace_chars(t)

defp replace_chars([]), do: []
```

This is will replace single chars:

```
defp replace_chars([?ä|t]), do: "ae" ++ replace_chars(t)
defp replace_chars([?ö|t]), do: "oe" ++ replace_chars(t)
defp replace_chars([?ü|t]), do: "ue" ++ replace_chars(t)
defp replace_chars([?Ä|t]), do: "Ae" ++ replace_chars(t)
defp replace_chars([?Ö|t]), do: "Oe" ++ replace_chars(t)
defp replace_chars([?Ü|t]), do: "Ue" ++ replace_chars(t)
```

Replace definitions from a file

A file containing tuples of replacements:

```
# replacements.exs
[
    {?ä, 'ae'}, {?ö, 'oe'}, {?ü, 'ue'},
    {?Ä, 'Ae'}, {?Ö, 'Oe'}, {?Ü, 'Ue'}
]
```

Generate Code!

Elixir can *run code* at compile time!

```
{replacements, _} = Code.eval_file("replacements.exs", __DIR__)

for {search, replace} <- replacements do
    defp replace_chars([unquote(search)|t]) do
        unquote(replace) ++ replace_chars(t)
    end
end
```

Resulting Code

```
# Generated
defp replace_chars([?ä|t]), do: 'ae' ++ replace_chars(t)
defp replace_chars([?ö|t]), do: 'oe' ++ replace_chars(t)
defp replace_chars([?ü|t]), do: 'ue' ++ replace_chars(t)
defp replace_chars([?Ä|t]), do: 'Ae' ++ replace_chars(t)
defp replace_chars([?Ö|t]), do: 'Oe' ++ replace_chars(t)
defp replace_chars([?Ü|t]), do: 'Ue' ++ replace_chars(t)

# Static
defp replace_chars([h|t]),  do: [h]  ++ replace_chars(t)
defp replace_chars([]),     do: []
```

Protocol

Like an **Interface** but dependent on

- the *type of given argument*

instead of

- instance of *implementing class*.

PHP Example

```php
<?php
interface SluggifyInterface
{
    public function slugify($string);
}
```

Sluggify Protocol

```
defprotocol Slugify do
  @fallback_to_any true

  @doc "Returns the slug for the given data"
  def slugify(data)
end
```

Sluggify Protocol default implementation

```
defimpl Slugify, for: Any do

  @doc """
    Default handler using String.Chars Protocol.
  """
  def slugify(data) do
    data |> Kernel.to_string |> Slugger.slugify
  end
end
```

Extending Sluggify Protocol

Anybody else can implement that protocol for their own data.

```
defmodule BlogPost do
    defstruct title: "Ecto Version 2.0 released"
end
```

Extending Sluggify Protocol

Anybody else can implement that protocol for their own data.

```
defmodule BlogPost do
    defstruct title: "Ecto Version 2.0 released"
end

defimpl Slugify, for: BlogPost do
    def slugify(post) do
        # Create slug only from title of BlogPost
        post.title |> Slugger.slugify
    end
end
```

The End!

Thanks

:)