Route and Response

online.pragmaticstudio.com/courses/elixir/steps/10

Notes

Accessing Map Values with Atoms vs. Strings

Suppose we have the following map defined in an `iex` session:

```
iex> conv = %{ method: "GET", path: "/wildthings" }
```

The keys are atoms, so to get the values associated with the keys we can use the square-bracket syntax:

```
iex> conv[:method]
"GET"

iex> conv[:path]
"/wildthings"
```

The square-bracket syntax works with all maps. However, when the keys are atoms, you can also use the dot notation:

```
iex> conv.method
"GET"

iex> conv.path
"/wildthings"
```

Now suppose we use strings instead of atoms as the map keys:

```
iex> conv = %{ "method" => "GET", "path" => "/wildthings" }
```

To access the same values as before we must use strings with the square-bracket syntax:

```
iex> conv["method"]
"GET"

iex> conv["path"]
"/wildthings"
```

Trying to use an atom returns `nil` because no such key exists:

```
iex> conv[:method]
nil
```

It's also worth noting that the dot notation only works for keys that are atoms:

```
iex> conv.method
** (KeyError) key :method not found in: %{"method" => "GET", "path" =>
"/wildthings"}
```

Gotcha: String Lengths

A sequence of bytes in Elixir is referred to as a *binary*. You rarely need to grovel around at the byte level, but it's important to understand that double-quoted Elixir strings are represented internally as a sequence of bytes. Thus, double-quoted strings are binaries. Here's one reason that's important...

In an `iex` session, suppose we have a string variable representing a response body:

```
iex> resp_body = "Bears, Lions, Tigers"
"Bears, Lions, Tigers"
```

In the video, we got the length of the string using the `String.length/1` function:

```
iex> String.length(resp_body)
20
```

Since the string is a binary, we can get the number of bytes in the string using the built-in `byte_size/1` function:

```
iex> byte_size(resp_body)
20
```

No surprise, both functions return the same number: 20.

But now suppose we change "Lions" to "Liöns" (notice the umlaut):

```
iex> resp_body = "Bears, Liöns, Tigers"
"Bears, Liöns, Tigers"
```

The length of the string is still 20:

```
iex> String.length(resp_body)
20
```

But the number of bytes has increased from 20 to 21:

```
iex> byte_size(resp_body)
21
```

Why? Well, a string is a **UTF-8 encoded binary** and the character "ö" takes 2 bytes to be represented in UTF-8. So even though the string has 20 characters, the number of bytes in that string is 21.

Now this isn't a problem for the response body strings we'll use in the videos. However, using `String.length/1` to set the value of the `Content-Length` header in the response won't be correct for all possible UTF-8 encoded strings. The `Content-Length` header must indicate the size of the body in *bytes*. Therefore, it's better to use the `byte_size/1` function to get the number of bytes:

```
Content-Length: #{byte_size(conv.resp_body)}
```

wööt!

Code So Far

The code for this video is in the `route-response` directory found within the `video-code` directory of the code bundle.

Go To Next Video