Los atributos del módulo en Elixir tienen tres propósitos

- Sirven para anotar el módulo, a menudo con información para ser utilizada por el usuario o la VM.
- Trabajan como constantes.
- Funcionan como un módulo de almacenamiento temporal para ser utilizado durante la compilación.

Veamos cada caso, uno por uno.

Anotaciones

Elixir trae el concepto de atributos de módulo de Erlang. Por ejemplo:

```
defmodule MyServer do
  @vsn 2
end
```

En el ejemplo anterior, estamos configurando explícitamente el atributo de versión para ese módulo. @vsn es utilizado por el mecanismo de recarga de código en Erlang VM para verificar si un módulo se ha actualizado o no. Si no se especifica ninguna versión, la versión se establece en la suma de comprobación MD5 de las funciones del módulo.

Elixir tiene un puñado de atributos reservados. Aquí hay algunos de ellos, los más utilizados:

- @moduledoc: proporciona documentación para el módulo actual.
- @doc: proporciona documentación para la función o macro que sigue al atributo.
- @behaviour: (observe la ortografía británica) utilizada para especificar un OTP o un comportamiento definido por el usuario.
- @before_compile: proporciona un enlace que se invocará antes de compilar el módulo. Esto

hace posible inyectar funciones dentro del módulo exactamente antes de la compilación.

@moduledoc y @doc son los atributos más utilizados. Elixir trata la documentación como de primera clase y proporciona muchas funciones para acceder a la documentación. Puede leer más en la documentación en Elixir.

Volvamos al módulo Math definido en los capítulos anteriores, agreguemos documentación y guárdelo en el archivo math.ex:

```
defmodule Math do
  @moduledoc """
  Provides math-related functions.

  ## Examples

      iex> Math.sum(1, 2)
      3

  """

  @doc """
  Calculates the sum of two numbers.
  """
  def sum(a, b), do: a + b
end
```

Elixir promueve el uso de Markdown con heredocs para escribir documentación legible. Los heredocs son cadenas de varias líneas, comienzan y terminan con comillas dobles triples, manteniendo el formato del texto interno. Podemos acceder a la documentación de cualquier módulo compilado directamente desde IEx:

```
$ elixirc math.ex
$ iex
```

```
iex> h Math # Access the docs for the module Math
...
iex> h Math.sum # Access the docs for the sum function
```

```
...
```

También tenemos una herramienta llamada ExDoc que se utiliza para generar páginas HTML a partir de la documentación.

Puede consultar los documentos de Módulo para obtener una lista completa de los atributos admitidos. Elixir también usa atributos para definir typepecs.

Esta sección cubre los atributos integrados. Sin embargo, los desarrolladores también pueden usar los atributos o extenderlos las bibliotecas para admitir un comportamiento personalizado.

Constantes

Los desarrolladores de Elixir a menudo usan atributos de módulo cuando desean hacer que un valor sea más visible o reutilizable:

```
defmodule MyServer do
  @initial_state %{host: "127.0.0.1", port: 3456}
  IO.inspect @initial_state
end
```

Intentar acceder a un atributo que no se definió imprimirá una advertencia:

```
defmodule MyServer do
  @unknown
end
warning: undefined module attribute @unknown, please remove access to @unknown or explicitly set it before
access
```

Los atributos también se pueden leer dentro de las funciones:

```
defmodule MyServer do
  @my_data 14
  def first_data, do: @my_data
  @my_data 13
  def second_data, do: @my_data
end

MyServer.first_data #=> 14
MyServer.second_data #=> 13
```

Cada vez que se lee un atributo dentro de una función, se toma una instantánea de su valor actual. En otras palabras, el valor se lee en tiempo de compilación y no en tiempo de ejecución. Como veremos, esto también hace que los atributos sean útiles como almacenamiento durante la compilación del módulo.

Normalmente, repetir un atributo del módulo hará que su valor se reasigne, pero hay circunstancias en las que es posible que desee configurar el atributo del módulo para que se acumulen sus valores:

```
defmodule Foo do
  Module.register_attribute __MODULE__, :param, accumulate: true

  @param :foo
  @param :bar
  # here @param == [:foo, :bar]
end
```

Se pueden invocar funciones al definir un atributo del módulo:

```
defmodule MyApp.Notification do
  @service Application.get_env(:my_app, :email_service)
  @message Application.get_env(:my_app, :welcome_email)
  def welcome(email), do: @service.send_welcome_message(email, @message)
end
```

Sin embargo, tenga cuidado: las funciones definidas en el mismo módulo que el atributo en sí no se pueden invocar porque aún no se han compilado cuando se está definiendo el atributo.

Al definir un atributo, no deje un salto de línea entre el nombre del atributo y su valor.

Almacenamiento temporal

Uno de los proyectos en la organización Elixir es el proyecto Plug, que pretende ser una base común para construir bibliotecas web y marcos en Elixir.

La biblioteca Plug permite a los desarrolladores definir sus propios plugins que se pueden ejecutar en un servidor web:

```
defmodule MyPlug do
  use Plug.Builder

  plug :set_header
  plug :send_ok

  def set_header(conn, _opts) do
    put_resp_header(conn, "x-header", "set")
  end

  def send_ok(conn, _opts) do
    send_resp(conn, 200, "ok")
  end
end

IO.puts "Running MyPlug with Cowboy on http://localhost:4000"
Plug.Adapters.Cowboy.http MyPlug, []
```

En el ejemplo anterior, hemos utilizado la macro plug/1 para conectar funciones que se invocarán cuando haya una solicitud web. Internamente, cada vez que llama a plug/1, la biblioteca Plug almacena el argumento dado en un atributo @plugs. Justo antes de compilar el módulo, Plug ejecuta una devolución de llamada que define una función (call/2) que maneja las solicitudes HTTP. Esta función ejecutará todos los enchufes dentro de @plugs en orden.

Para comprender el código subyacente, necesitaríamos macros, por lo que volveremos a visitar este patrón en la guía de metaprogramación. Sin embargo, el enfoque aquí está en cómo usar los atributos del módulo como almacenamiento permite a los desarrolladores crear DSL.

Otro ejemplo proviene del marco ExUnit que utiliza los atributos del módulo como anotación y almacenamiento:

```
defmodule MyTest do
  use ExUnit.Case

  @tag :external
  test "contacts external service" do
    # ...
  end
end
```

Las etiquetas en ExUnit se utilizan para anotar pruebas. Las etiquetas se pueden usar más tarde para filtrar las pruebas. Por ejemplo, puede evitar ejecutar pruebas externas en su máquina porque son lentas y dependen de otros servicios, mientras que aún pueden habilitarse en su sistema de compilación.

Esperamos que esta sección arroje algo de luz sobre cómo Elixir admite la metaprogramación y cómo los atributos del módulo juegan un papel importante al hacerlo.