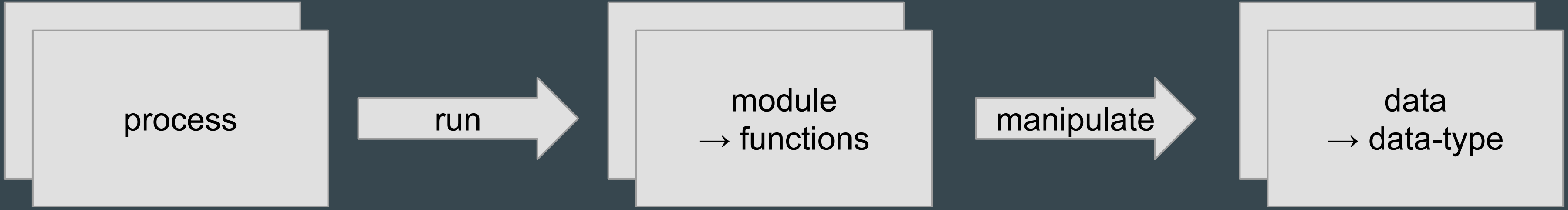
Elixir Meetup Aachen

Protocol <> Behaviour

by
Niklas Möller
(Sportograf Gmbh Co KG)

Elixir’s core elements



“[…] they are all interconnected: **processes** run the code defined in **modules** that manipulate the **data types** […]”

José Valim

“Polymorphism”

Elixir provides different “polymorphism” mechanisms for each core element

Processes	Modules	Data
● Process can send messages to every other processes ● Messages are the common interface and not written in code ● **Implicit polymorphism**	● Modules contain functions ● Caller only wants to know a module contains a function which accepts specific parameters ● e.g. a Parser has the parse/1 function	● Often you only need to know you can do something with a data-type, not how it is done ● e.g. read the size of a data-type

Protocol - for data-types

- Define what you can do to a data-type
  - boolean, String, map, tuple, …
- Functions ask for data-types providing a specific protocol

NO compile error

Typical scenario

Functions of module Enum (e.g. Enum.map) requires you to pass a data-type which has the Enumerable-Protocol functions defined

Protocol - What about some coding...?

→ Fallback-Implementation with “Any” possible

Protocol - working with structs

- extended maps
- protocols match structs independent

$$
\begin{array}{rl}
\%User\{\} & \rightarrow \text{defimpl Size, for: User} \\
\%Foo\{\} & \rightarrow \text{defimpl Size, for: Foo} \\
\text{not: } \%Bar\{\} & \rightarrow \text{defimpl Size, for: Map}
\end{array}
$$

Protocol - Some more coding...

Behaviour - for modules

- Comparable to interfaces in OO
- Define an abstract set of functions
- Compiler-warning if not defined

Typical scenario

Plug-Behaviour - Provides an exchangeable interface

  - Requires definition of functions init/1 and call/2
  - A caller only needs to know your module has the Plug-Behaviour to call it

Behaviour - What about some coding...?

@behaviour vs. use

- “use” instead of “@behaviour”
- “use” has nothing to do with behaviours but is used for convenience

- @behaviour
  - the actual behaviour-implementation
- use
  - the modules “__using__” function or macro will be called
  - often used to implement a behaviour and additional default code
  - that way you can implement default functionality

Behaviour - Some more coding...

Protocol vs. Behaviour

Protocol	Behaviour
● Can be defined anywhere. ● On each call the program has to check for existing implementations anywhere in the code ● Compiler like mix optimize for this situation (protocol consolidation) ● Can be used outside of the library or application they are defined in ● No validation. When a definition is missing, you get a RUNTIME error.	● Behaviours need to define their functions in the module ● Behaviours are explicit. You get a compiler-warning if a required function is missing.

The winner is... everybody!

- Protocols and behaviours are made for different things.
- You need to know when to use what.

When to use what

Protocol	Behaviour
● Define how a data-type handles a specific function	● Define what a module is able to do ● “use” instead of “@behaviour”
● e.g. Size / Enumerable	● e.g. Parser → JSONParser.parse/1 → YAMLParser.parse/1

When to use what

more flexible
more extensible
more reusable

and that's what everybody loves

Read more...

Protocols vs. Behaviours

- https://www.djm.org.uk/posts/elixir-behaviours-vs-protocols-what-is-the-difference/

Behaviours

- https://elixir-lang.org/getting-started/typespecs-and-behaviours.html#behaviours
- https://www.djm.org.uk/posts/writing-extensible-elixir-with-behaviours-adapters-pluggable-backends/

Protocols

- https://elixir-lang.org/getting-started/protocols.html
- https://medium.com/everydayhero-engineering/extensibility-in-elixir-using-protocols-2e8fb0a35c48

Code Examples @ GIT

- https://github.com/LordZedDE/elixir_meetup_aachen_protocolbehaviour

That’s it!

Thank you for listening.