

# ELIXIR PLUGS

**SIMPLE**  
**(MOSTLY)**

PART OF ELIXIR CORE  
[GITHUB.COM/ELIXIR-LANG/PLUG](https://github.com/elixir-lang/plug)

WHAT IS A  
PLUG?

SPECIFICATION FOR  
**COMPOSABLE MODULES**  
BETWEEN WEB APPS

# CONNECTION ADAPTER

## FOR WEB SERVERS

## IN THE ERLANG VM

SO A LOT LIKE  
**RACK**  
REALLY

**PHOENIX** IS BASED  
HEAVILY ON PLUGS



**EVERYTHING IS A PLUG  
(ALMOST)**

# 2 TYPES

- > FUNCTION
- > MODULE

# FUNCTION PLUG

- > ANY FUNCTION
- > TAKES CONNECTION + OPTIONS
- > RETURNS A CONNECTION

```
(Plug.Conn.t, Plug.opts) :: Plug.Conn.t
```

```
def p(conn, _opts) do  
  conn |> do_something  
end
```

# WHAT IS Plug.Conn?

```
%Plug.Conn{  
  host: "www.example.com",  
  path_info: ["bar", "baz"],  
  ...}
```

**DIRECT CONNECTION TO UNDERLYING WEBSERVER**

**HOLDS ALL HEADER, REQUEST AND RESPONSE INFO**

**PASSED ALL THE WAY THROUGH THE PLUG PIPELINE**

```
iex(9)> %Plug.Conn{}
%Plug.Conn{adapter: {Plug.Conn, :...}, assigns: %{}, before_send: [],
  body_params: %Plug.Conn.Unfetched{aspect: :body_params},
  cookies: %Plug.Conn.Unfetched{aspect: :cookies}, halted: false,
  host: "www.example.com", method: "GET", owner: nil,
  params: %Plug.Conn.Unfetched{aspect: :params}, path_info: [], peer: nil,
  port: 0, private: %{},
  query_params: %Plug.Conn.Unfetched{aspect: :query_params}, query_string: "",
  remote_ip: nil, req_cookies: %Plug.Conn.Unfetched{aspect: :cookies},
  req_headers: [], request_path: "", resp_body: nil, resp_cookies: %{},
  resp_headers: [{"cache-control", "max-age=0, private, must-revalidate"}],
  scheme: :http, script_name: [], secret_key_base: nil, state: :unset,
  status: nil}
```



# MODULE PLUG

- `init(options)` INITIALISES OPTIONS
- `call(conn, options)` SAME AS A FUNCTION PLUG

THE RESULT OF `init/1` IS PASSED TO `call/2`

`init/1` MAY BE CALLED DURING COMPILATION

# FUNCTION PLUG EXAMPLE

```
def json_header_plug(conn, _opts) do
  conn |> put_resp_content_type("application/json")
end
```

# MODULE TRANSFORMATION

```
defmodule JSONHeaderPlug do
  def init(opts) do
    opts
  end

  def call(conn, _opts) do
    conn |> put_resp_content_type("application/json")
  end
end
```

# MODULE RESPONDER

```
defmodule MyPlug do

  @behaviour Plug

  def init(options) do
    options
  end

  def call(conn, _opts) do
    conn
    |> put_resp_content_type("text/plain")
    |> send_resp(200, "Hello world")
  end
end
```

# TESTING A PLUG

```
defmodule MyPlugTest do
  use ExUnit.Case, async: true
  use Plug.Test

  test "returns hello world" do
    # Create a test connection
    conn = conn(:get, "/hello")

    # Invoke the plug
    conn = MyPlug.call(conn, [])

    # Assert the response and status
    assert conn.state == :sent
    assert conn.status == 200
    assert conn.resp_body == "Hello world"
  end
end
```

Plug.Builder

**HELPS BUILD PLUG PIPELINES**

# PIPELINE EXAMPLE

```
defmodule MyPlugPipeline do
  use Plug.Builder

  plug Plug.Logger
  plug :hello, upper: true

  # A function from another module can be plugged too, provided it's
  # imported into the current module first.
  import AnotherModule, only: [interesting_plug: 2]
  plug :interesting_plug

  def hello(conn, opts) do
    body = if opts[:upper], do: "WORLD", else: "world"
    send_resp(conn, 200, body)
  end
end
```

# PLUGS ALL THE WAY DOWN

