`:gen_fsm` meets *elixir*

@gogogarrett

:gen_fsm

Generic Finite State Machine

GenServers' brother in crime

A finite-state machine (FSM)

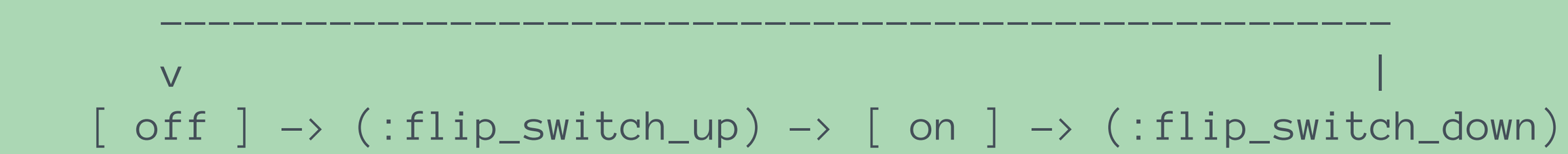
- ☞ can be in one of a finite number of states
- ☞ only can be in one state at a time
- ☞ change from one state to another by sending events

A simple state machine

States:
[on] [off]

Events:
(flip_switch_up), (flip_switch_down)

Fancy Chart:



Simple Workflow

- ☞ setup:
  - ☞ (client) `start_link`
  - ☞ (server) `init`
- ☞ usage:
  - ☞ (client) `get_state` [any function / event]
  - ☞ (server) `off(:get_state)` [current_state (event)]

Lets see it in action

```
defmodule LightSwitch.StateMachine do
  @name :LSFSM

  # Client api
  def start_link(inital_gen_state) do
    :gen_fsm.start_link({:local, @name}, __MODULE__, inital_gen_state, [])
  end

  def get_state do
    :gen_fsm.sync_send_event(@name, :get_state)
  end

  # Server api
  def init(gen_state) do
    {:ok, :off, gen_state}
  end

  def off(:get_state, _from, gen_state) do
    {:reply, :off, :off, gen_state}
  end
end
```

start_link

```
def start_link(inital_gen_state) do
  :gen_fsm.start_link({:local, @name}, __MODULE__, inital_gen_state, [])
end
```

☞ `{:local, :Something}`: how to name the gen_fsm

☞ `__MODULE__` which module the callbacks will go to

☞ `inital_gen_state` === `inital_gen_state`

☞ `[]` - other ops to gen_fsm which I have no idea what they do

init

```
def init(gen_state) do
  {:ok, :off, gen_state}
end
```

☞ `gen_state`: value that is specified in start_link

  ☞ similar to the `GenServer` state

☞ `{:ok, next_state_name, next_state}` if setup went well

☞ `{:stop, Reason}` if something goes wrong in setup

get_state

```
# Client api
def get_state do:
  # SYNC message send - aka: `GenServer.call`
  :gen_fsm.sync_send_event(@name, :get_state)
end

# Server api
# current_state is `off` specified in the `init` function
def off(:get_state, _from, gen_state) do
  # reply, what to reply, next state name, gen_state for gen_fsm
  {:reply, :off, :off, gen_state}
end
```

```elixir
defmodule LightSwitch.StateMachine do
  @name :LSFSM

  # Client api
  def start_link(inital_gen_state) do
    :gen_fsm.start_link({:local, @name}, __MODULE__, inital_gen_state, [])
  end

  def get_state, do: :gen_fsm.sync_send_event(@name, :get_state)

  # Server api
  def init(gen_state) do
    {:ok, :off, gen_state}
  end

  def off(:get_state, _from, gen_state) do
    {:reply, :off, :off, gen_state}
  end
end
```

```
iex(1)> LightSwitch.StateMachine.get_state
:off
```


That's all Folks!
kalilak

Great, we have a current state.

Let's change it.

:gen_fsm has two ways to send events

- ☞ `sync_send_event`
  - ☞ StateName/3 (sync)
- ☞ `send_event`
  - ☞ StateName/2 (async)

Sync

`sync_send_event` && StateName/3

```
:gen_fsm.sync_send_event(@name, :get_state)
:gen_fsm.sync_send_event(@name, {:player_joined, %{id: 1, name: "garrett"}})
:gen_fsm.sync_send_event(@name, {:setup_server, [config: true]})
```

☞ Gives you the return value of the function

☞ Blocks until a result is given

☞ The 2nd param is to be pattern matched in `StateName/3`

sync_send_event && **StateName/3**

```
# def off({:setup_server, keyword_config, _from, gen_state) do
# def off({:player_joined, %{id: 1}}, _from, gen_state) do
# def off({:player_joined, %{id: id}}, _from, gen_state) do

def off(:get_state, _from, gen_state) do
  # {:reply, Reply, NextStateName, NewStateData}
  #  +(Timeout, hibernate)
  # {:next_state, NextStateName, NewStateData}
  #  +(Timeout, hibernate)
  # {:stop, Reason, Reply, NewStateData}
  # {:stop, Reason, NewStateData}
  {:reply, "The current_state is :off", :off, gen_state}
end
```

Async

`send_event` && StateName/2

```
:gen_fsm.send_event(@name, :increase_counter)
:gen_fsm.send_event(@name, {:player_joined, %{id: 1, name: "garrett"}})
:gen_fsm.send_event(@name, {:save_event, {:lesson, %{id: 1}}})
```

☞ Fire and forget

☞ Continues to execute rest of the code

☞ The 2nd param is to be pattern matched in `StateName/2`

send_event && **StateName/2**

```elixir
# def off({:increase_counter, gen_state) do
# def off({:player_joined, %{id: id}}, gen_state) do
# def off({:save_event, {:activity, map}}, gen_state) do

def off({:save_event, {:lesson, map}}, gen_state) do
  # {:next_state, NextStateName, NewStateData}
  # {:next_state, NextStateName, NewStateData, Timeout}
  # {:next_state, NextStateName, NewStateData, hibernate}
  # {:stop, Reason, NewStateData}
  {:next_state, :on, gen_state}
end
```

Back to our lights

Two events

☞ `flip_switch_up`

☞ `flip_switch_down`

Event: `flip_switch_(up|down)`

Q:

☞ do we really care if this causes a state transition?

☞ do we need any return value from this?

A:

☞ No. This is a fire and forget event

```
# client
def flip_switch_up do
  :gen_fsm.send_event(@name, :flip_switch_up)
end

def flip_switch_down do
  :gen_fsm.send_event(@name, :flip_switch_down)
end

# server
def off(:flip_switch_up, gen_state) do
  {:next_state, :on, gen_state}
end

def on(:flip_switch_down, gen_state) do
  {:next_state, :off, gen_state}
end
```

```elixir
defmodule LightSwitch.StateMachine do
  @name :LSFSM
  # Client api
  def start_link(inital_gen_state) do
    :gen_fsm.start_link({:local, @name}, __MODULE__, inital_gen_state, [])
  end

  def get_state,        do: :gen_fsm.sync_send_event(@name, :get_state)

  def flip_switch_up,   do: :gen_fsm.send_event(@name, :flip_switch_up)
  def flip_switch_down, do: :gen_fsm.send_event(@name, :flip_switch_down)

  # Server api
  def init(gen_state), do: {:ok, :off, gen_state}

  def off(:get_state, _from, gen_state), do: {:reply, :off, :off, gen_state}
  def off(:flip_switch_up, gen_state),   do: {:next_state, :on, gen_state}

  def on(:get_state, _from, gen_state),  do: {:reply, :on, :on, gen_state}
  def on(:flip_switch_down, gen_state),  do: {:next_state, :off, gen_state}
end
```

:gen_fsm in action

```
iex(1)> LightSwitch.StateMachine.get_state
:off
iex(2)> LightSwitch.StateMachine.flip_switch_up
:ok
iex(3)> LightSwitch.StateMachine.get_state
:on
iex(4)> LightSwitch.StateMachine.flip_switch_down
:ok
iex(5)> LightSwitch.StateMachine.get_state
:off
```

Demo

Resources

- ☞ http://learnyousomeerlang.com/finite-state-machines
- ☞ http://erlang.org/doc/man/gen_fsm.html
- ☞ **https://github.com/gogogarrett/light_switch**
- ☞ https://pdincau.wordpress.com/2010/09/07/an-introduction-to-gen_fsm-behaviour/

Thanks

☞ @gogogarrett