ETS, DETS, Mnesia, and things

Julian Doherty
madlep.com
@madlep

State

Try not to. But you probably will though

Agents

Simple, easy, relatively inflexible

Possibly 💩 depending on who you talk to

```
{:ok, agent} = Agent.start( fn() -> 0 end)
# {:ok, #PID<0.91.0>}

Agent.get(agent, fn(state) -> state end)
# 0

Agent.update(agent, fn(state) -> state + 1 end)
# :ok

Agent.get(agent, fn(state) -> state end)
# 1
```

Agents

So?... What's the big deal?

Agents

- Agent is a separate process
- Can read/write from multiple places
- Functions are executed in process that maintains agent data
- Isolated, serialized, transactional

Agents

Can be limiting. You'll probably grow out of them

And end up using...

GenServer

```elixir
defmodule IncrementServer do
  use GenServer

  def start_link(number \\ 0) do
    GenServer.start_link(IncrementServer, number)
  end

  def handle_call(:get_number, _from, state) do
    {:reply, state, state}
  end

  def handle_cast(:increment, state) do
    {:noreply, state + 1}
  end
end

{:ok, my_incr} = IncrementServer.start_link()

GenServer.call(my_incr, :get_number)
# 0

GenServer.cast(my_incr, :increment)
# :ok

GenServer.call(my_incr, :get_number)
# 1
```

GenServer

— more complicated
— more flexible
— you'll probably end up here

Agents and GenServer

Both can be bottle necks

Pure Erlang/Elixir can't do fast mutable or shared data

Need some

Ets, Dets, Mnesia

Implemented in native code, built into the VM

Can do things Erlang/Elixir code isn't allowed to

Ets

Erlang

Term

Storage

(In memory data store)

```
people_tid = :ets.new(:people, [])
# 135191

:ets.insert(people_tid, {
  "madlep",
  %{name: "Julian", twitter: "@madlep"}
})
# true

:ets.lookup(people_tid, "madlep")
# [{"madlep", %{name: "Julian", twitter: "@madlep"}}]
```

Ets

— fast

— in memory

— atomic, serializable

— transient

— need to understand error handling

Dets

Disk

Erlang

Term

Storage

(Ets, but slower and saved to disk)

```
{:ok, people_tid} = :dets.open_file(
  :people,
  [file: 'my_people_file.dets']
)

:dets.insert(people_tid, {
  "madlep",
  %{name: "Julian Doherty", twitter: "@madlep"}
})
# :ok

:dets.lookup(people_tid, "madlep")
# [{"madlep", %{name: "Julian Doherty", twitter: "@madlep"}}]

:dets.close(people_tid)
# :ok
```

Dets

- almost the same as Ets
- durable, survives VM restarts
- slower, disk based vs memory based
- slower, like EVERYTHING is a disk operation
- no in memory caching

Dets is slow, but durable. Ets is fast, but not durable...

Why don't I just cache Dets with Ets?

Erlang is way ahead of you

Mnesia

(HAHA get it! Mnesia… Amnesia… Programmer joke!… 🙄)

Mnesia

RDBMS built into Erlang stdlib

Built on top of Ets and Dets

Transactional, distributable, replicatable

Can be disk and/or memory based

```
:mnesia.create_schema([node()])
# :ok

:mnesia.start()
# :ok

:mnesia.create_table(Person, [attributes: [:id, :name, :twitter]])
# {:atomic, :ok)

:mnesia.transaction(fn ->
  :mnesia.write({Person, "madlep", "Julian", "@madlep"})
  :mnesia.write({Person, "trump", "Donald", "@realDonaldTrump"})
end)
# {:atomic, :ok}

:mnesia.transaction(fn ->
  :mnesia.read({Person, "madlep"})
end)
# {:atomic, [{Person, "madlep", "Julian", "@madlep"}]}
```

Mnesia

— closer to MySql/Postgresql (but not as good)
— full transactions
— indexes
— richer querying
— distributed
— replicated
— need to understand split brain

In summary

Ets
- You'll probably use this. A lot

Dets
- Useful for occasionally accessed, persistent data

Mnesia
- More complex
- Usually MySQL or Postgresql is a better option
- Has it's uses

Thank you! Questions?

Julian Doherty
madlep.com
@madlep