akka.net in a nutshell

@profesor79pl

Overview

1. What is an actor system?
2. A bit of intro (and history)...
3. Coding demo/exercises
4. Summary

Why do JAVA developers wear glasses?

Because they don’t **C#…**

A bit of history

- The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation. In response to a message that it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify their own private state, but can only affect each other through messages (avoiding the need for any locks).

- The actor model originated in 1973

So what is “THE ACTOR”?

- Unit of code
- Helps with separation of ‘policy decisions’ and business logic
- A class in our code that can be run in same system or distributed
- A thread
- An object instance or component
- A callback or listener
- A singleton or a service
- A router, load balancer or pool
- A Finite State Machine (FSM)

Carl Hewitt’s definition

- The fundamental unit of computation that embodies:
  - Processing
  - Storage
  - Communication
- 3 Axioms - When an Actor receives a message it can:
  - Create new Actors
  - Send messages to Actors it knows
  - Designate how it should handle the next message it receives

4 core Actor operations

- 0 Define
- 1 Create
- 2 Send
- 3 Become
- 4 Supervise

0 - DEFINE

- What is the actor responsibility?
- How it need to handle own state?
- What level of interaction with other components is needed?
- Shall it delegate “RISKY” tasks?
- Proper name: MyFirstActor

1 - CREATE

- Props or DI ?
- Local or remote deployment?
- One actor or group (routing strategy)?
- What is planned lifespan?
- Access to resources?
- Do it need any pre-start actions?

2 - SEND

- An Actor is passive until a message is sent to it, which triggers something within the Actor
- Asynchronous and Non-blocking - Fire-forget
- EVERYTHING is asynchronous and lockless
- Send -> tell don’t ASK (more details to follow)

3 - BECOME

- Dynamically redefines Actor’s behavior
- Triggered reactively by receive of message
- Will now react differently to the messages it receives
- Implement an FSM (Finite State Machine)

4 - SUPERVISE

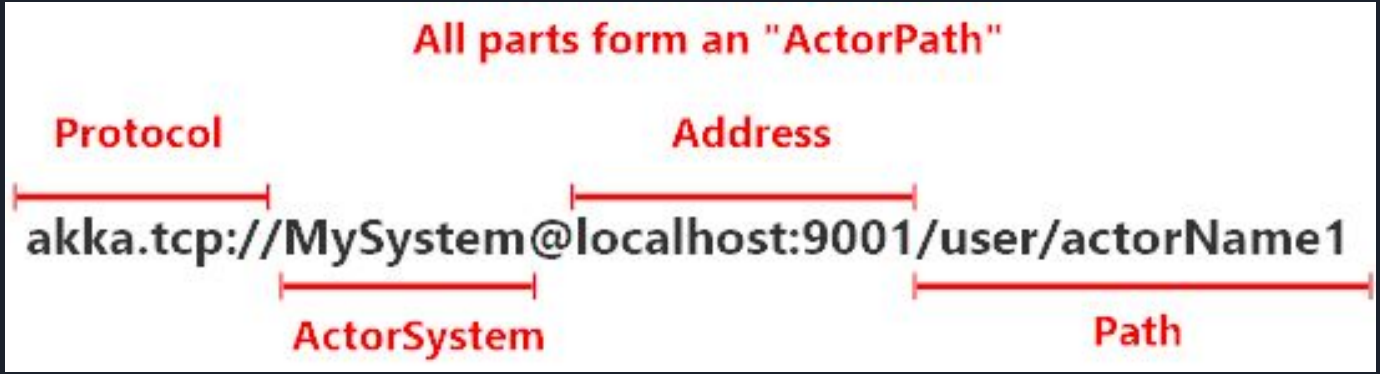
- How to handle exceptions not using try...catch :)
- One for one strategy
- One for all strategy
- What is decider

TALK IS CHEAP - SHOW ME THE CODE (LT)

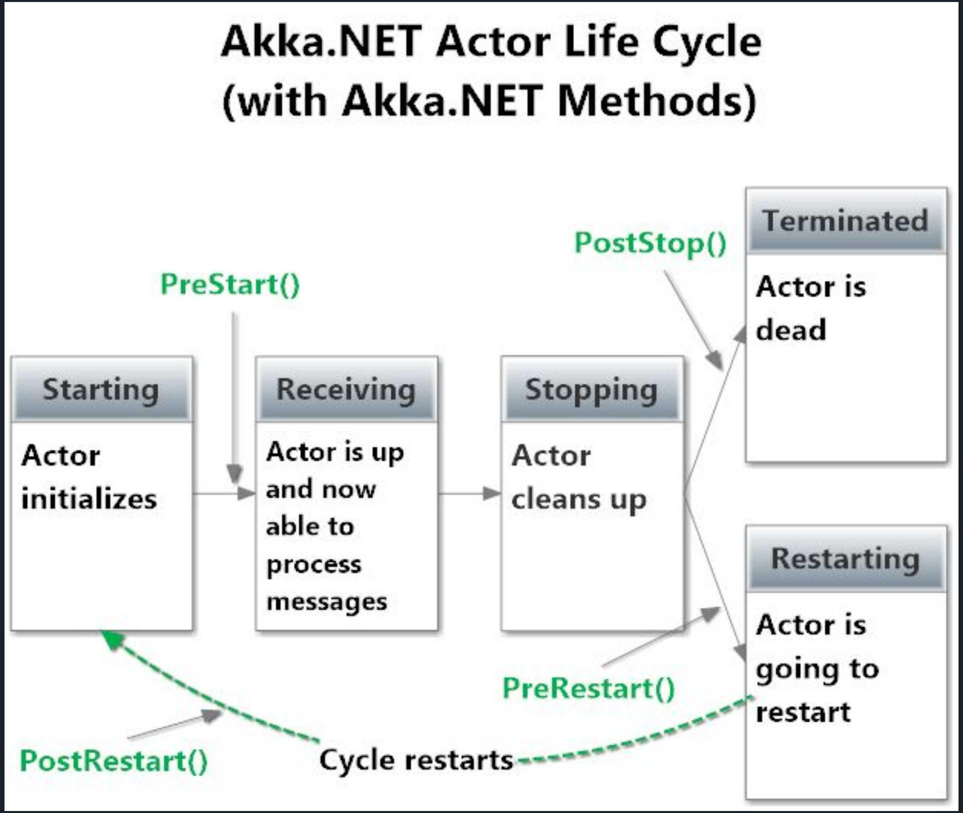
TASK LIST :)

1. Send - receive (what is your phone number)
2. Using a timer (ctor is not my whole life)
3. A finite state machine (light controller)
4. Supervision strategy in action (is there a life after death?)
5. Routing/Balancing (work smart not hard)
6. Can I get your signature, Sir? (at least one delivery)
7. How to deploy an actor in a remote system? (do we need that)
8. A simple actor cluster

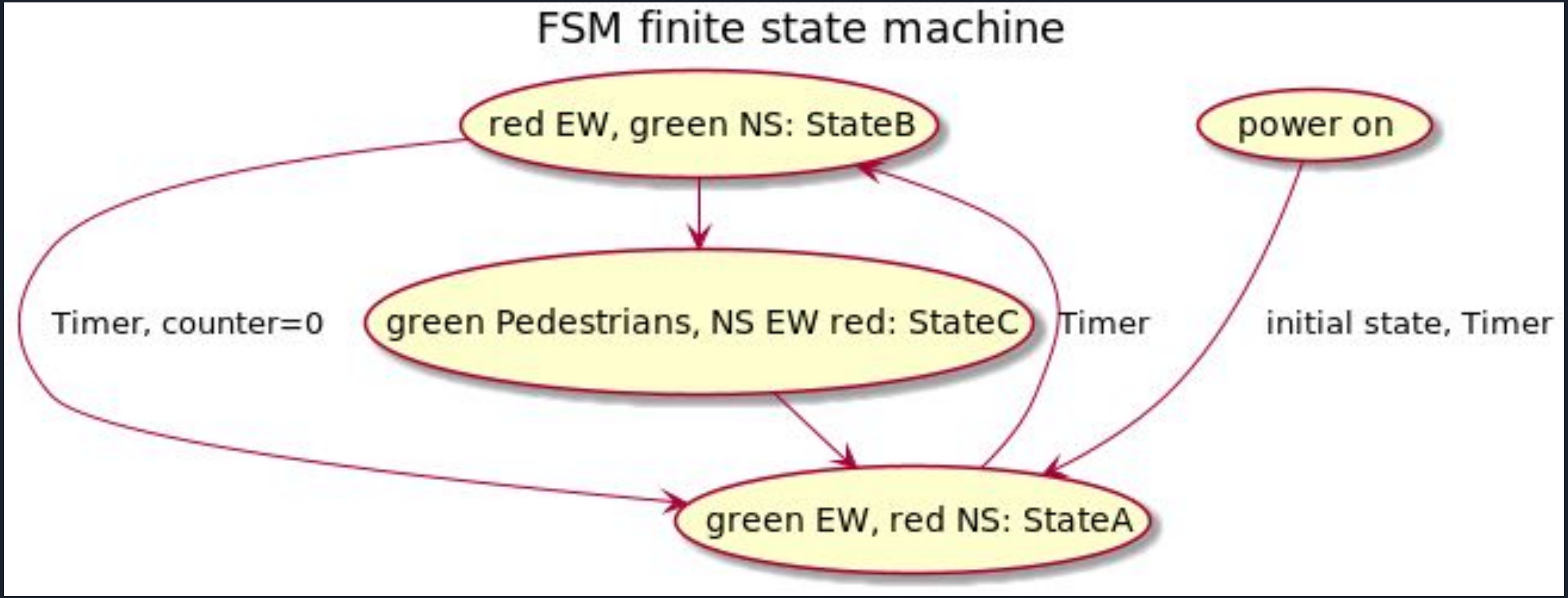
1. Send - receive (what is your phone number)


All parts form an "ActorPath"
Protocol
Address
akka.tcp://MySystem@localhost:9001/user/actorName1
ActorSystem
Path

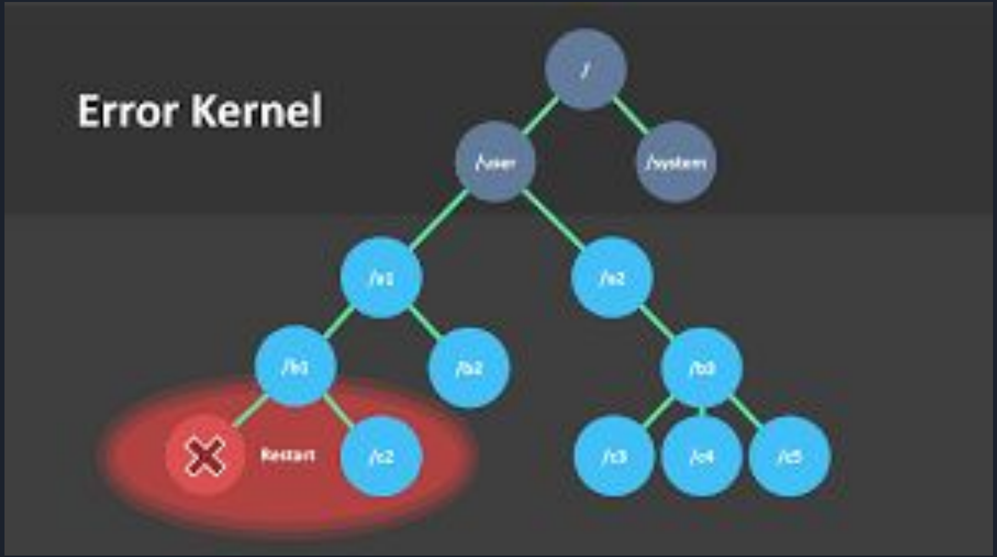
2. Using a timer (ctor is not my whole life)



3. A finite state machine (light controller)



4. Supervision strategy in action (is there a life after death?)



5. Routing/Balancing (work smart not hard)

- Pools vs. Groups
- Round-Robin
- Scatter-Gather
- Consistent Hashing

6. Can I get your signature, Sir? (at least one delivery)

7. How to deploy an actor in a remote system? (do we need that?)

- What we need to change in an actor?
- Is that complicated?
- Can we update actor remotely?

8. A simple actor cluster

Online resources

- getakka.net/articles/intro/what-is-akka.html
- www.slideshare.net/jboner/introducing-akka
- github.com/profesor79/devconf-lk-20190223

Questions?

Thank you!