Workshop on Play and Akka (Verizon VDSI)

Nirmalya Sengupta

Akka: Actors' behaviour can be configured

- Configuration is a necessity
  - Usual name: application.conf
- Convention is to store it in src/{java,scala}/resources directory
- Follows HOCON structure
- Akka provides a configuration at the root of the jar
- Application overrides the fields and values (fallback)
- An ActorSystem depends on it at the time of creations
- org.training.nirmalya.**sampleCodeSeven**

Akka: Scheduling

- An Actor can schedule occurrences of events / messages
- ActorSystem provides the Scheduler
- Once / Repetitive
- Cancellable interface: why do we need to cancel
- Remember: we are supposed to take an action
- org.training.nirmalya.**sampleCodeEight**

Actors: Logging

- Configuration
- EventStream
- Providers: STDOUT, SL4J

Akka: Exercise

1. Modify the Thesauras actor so that this time, at every 10 seconds, it prints a randomly chosen English word of the day, along with one Hindi synonym and one English synonym.

Akka: Futures (your own thread)

- Actors allow us to divert into the world of threads
- Future: the data structure that allows us accomplish time-consuming tasks
- Based on Promise: a contract of eventual fulfilment
- **ask()/pipeTo()**: alternative to tell()
  - An unseen Actor is managing the **tell()** conversation
- Mark the presence of dispatcher
  - Actor's receive() is freed for upcoming requests
- org.training.nirmalya.**sampleCodeNine**

Akka: Exercise

1. Modify the Thesauras actor so that this time, it uses ask/pipe pattern to interact with the Linguist actors.
2. Modify the Thesauras actor, so that it keeps track of the most popular English word searched for so far.
3. Add a message to the protocol. On receipt of this message, the Thesauras actor should provide the inquirer with the most popular English word so far)

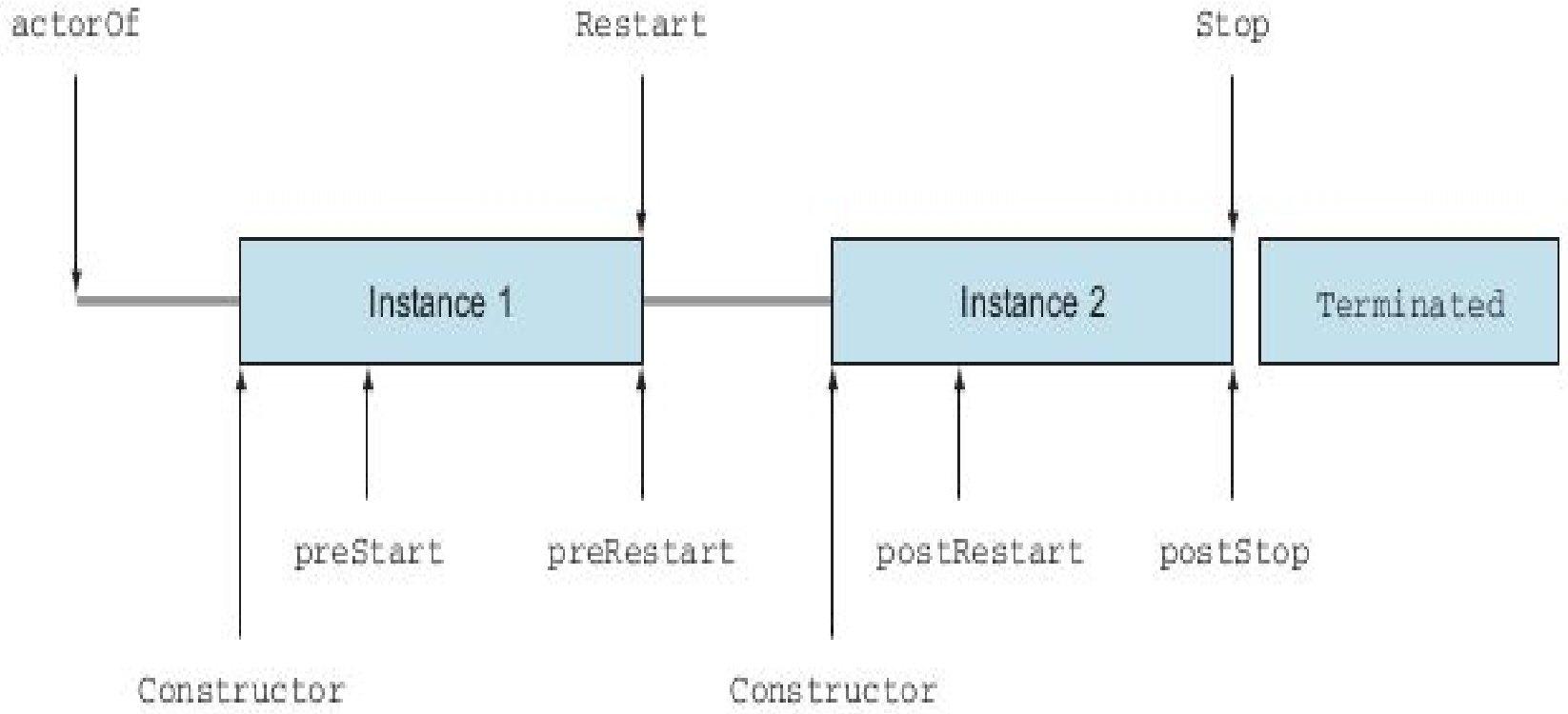
Actors: Reference and Path

- References are created for *referring* to an Actor as an object
- ActorSystem.actorOf(.....)
- Reference is applicable to the local ActorSystem
- Can also be identified with an ActorPath
- ActorPath can refer to the hierarchy
- ActorPath can yield ActorRef
- Equality to actors => equality of ActorPath and ActorRef

Actors: Remote Actors

- Two or more ActorSystems
  - Same IP, different ports
  - Different IP, (same | different) ports
- ActorSelection: helps in accessing the remote Actor
- Remote Actor’s parent hierarchy is important
- Sharable Configuration
- Serialization: an important consideration
- org.training.nirmalya.sampleCodeEleven.*

Actors: Life Cycle



Actors: Stopping an actor

- <ActorSystem>.stop(actorRef)
- context.stop(actorRef) // Children actor
- context.stop(getSelf()) // **Possible**
- actorRef ! **PoisonPill** // pill supplied by Akka library
- DeathWatch

Actors: Stopping an actor

- <ActorSystem>.stop(actorRef)
  - Actor receives the Stop message and suspends the actor's Mailbox.
  - Actor tells all its children to Stop. Stop messages propagate down the hierarchy below the actor.
  - Actor waits for all children to stop.
  - Actor calls PostStop lifecycle hook method for resource cleanup.
  - Actor shuts down

Actors: Poisoning an actor

- <ActorRef>.tell(PoisonPill.getInstance)
  - PoisonPill message is an auto-received, system-level message.
  - The actor treats a PoisonPill like an ordinary message.
  - The PoisonPill goes to the back of the actor's mailbox; preceding messages are processed.
  - PoisonPill is picked up and the Actor asks itself to stop

Actors: Killing an actor

- <ActorRef>.tell(akka.actor.Kill.getInstance,ActorReft.noSender)
  - Actor throws ActorKilledException and suspends operation
  - Supervisor (Parent) decides how to handle the exception
    - Resume (as if nothing happened)
    - Restart (start from scratch)
    - Terminate

Actors: Exercise

- A PlayerRankingInfoActor offers ranking of players
- It uses the services of a CricketRankingActor and a SoccerRankingActor
- Controllers use CompletionStage to return from actions
- People may want to vote to increase or decrease the rank of a player (rank > 0, always)
- Watchers query the rank of a Cricket or Soccer player (or of both)
- Admin may stop the voting for a given player, anytime.

Actors: Clustering

- A collection of ActorSystems (and of course, Actors inside)
- Usually, same host/different ports (but not necessarily)
- Seed Nodes, Joinee Nodes
- Configuration holds a very important position
- GOSSIP protocol
- Coordination: combination of Configuration and User Responsibility

Actors: Circuit Breakers

- Multi-layered applications depends on correctness of intervening modules
- Cascaded failures are a drain on resources, recovery and support
- CircuitBreakers help us in ensuring that responsiveness is not compromised

Actors: Circuit Breakers

- Begin in a CLOSED state (calls are flowing)
- Observe failures for a certain number of times (configurable)
- Sets itself to a OPEN state (no call can go through)
- Brings itself back to HALF_OPEN state after some time (configurable)
- If first call succeeds, back to OPEN state
- Callbacks available: onOpen, onClose, onHalfOpen

Actors: Routing

- Router and Routees
- Routing Strategies: several available
  - RoundRobinRoutingLogic
  - RandomRoutingLogic
  - SmallestMailboxRoutingLogic
  - Our own
- Routers can form a group or a pool
- Router.route(routee,sender())
- Supervision by Router in a Group

Actors: Reactive Systems

- React to demand on the computing facilities
  - Amadahl’s law / Gunther’s law
- Prefer Events, to Messages
- Embrace asynchronous natures; deals with failures
  - Coordination structure given by programming API
- Help imagine applications as a bunch of workflows

Actors: Reactive Systems

- Events are generated, components choose to react
- Programming model: Erlang
  - Temporal boundary
  - Isolation
- Message delivery guarantee
  - At most once
  - At least once
- Passing messages is the foundation

Actors: Reactive Systems

- Fault tolerance
- Resilience
  - Self-healing
    - Errors must become Messages
    - Helps in Isolation and Recovery
    - Responsibility of components higher-up
- Elasticity
- Replication

Actors: Reactive Systems

- Programming model requires
  - Addressability
  - Location transparency (here or there?)
  - Failure detection and recovery
  - Temporal nature (when? For how long?)
- System architecture issues still need attention
  - Consistency of data
  - Cross-node communication
  - Versioning