Introduction to Akka

Mirko Adari
• Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM.
Why Akka?

• Complexity
  – Concurrency
  – Scalability
  – Fault-tolerance
CONCURRENCY
Status quo

- Threads
- Shared state
- State visibility
- Locks
- Concurrent collections
- Thread notifications
- ...
What is an actor?

• By Carl Hewitt:
  – Unit of computation
    • Processing
    • Storage
    • Communication
What can actors be used for?

• a thread
• an object instance or component
• a callback or listener
• a singleton or service
• a router load-balancer or pool
• a Finite State Machine
Core operations

- Define
- Create
- Send
- Become
Define

class HelloActor extends Actor {
    val log = Logging(context.system, this)
    def receive = {
        case "hello" = log.info("received hello")
        case _       = log.info("received unknown message")
    }
}
class HelloActor extends Actor {
    val log = Logging(context.system, this)
    def receive = {
        val byeActor = context.actorOf(Props[ByeActor],
            name = "byeActor")
    }
}
Send

class HelloActor extends Actor {
    val log = Logging(context.system, this)
    def receive = {
        case "hello" = sender ! "hello back"
        case _ = log.info("received unknown message")
    }
}

class HelloActor extends Actor {
  val log = Logging(context.system, this)
  def happy: Receive = {
    case "hello" = sender ! "I feel great."
    case "bye"   = become(sad)
  }
  def sad: Receive = {
    case "hello" = become(happy)
    case "sad"   = sender ! "I’m thinking of suicide."
  }
  def receive = {
    case "hello" = become(happy)
    case "bye"   = become(sad)
  }
}
Become

• Transform into a pool or a router
• Finite State Machines
• Adaptive worker processes
Key concepts

• Parallel
• Asynchronous
• Non-blocking
• Immutable
• Mailboxes
• What happens if actor sends a message to itself?
class HelloActor extends Actor {
    def receive = {
        val future = timeActor ? "time"
        future onSuccess {
            case time = sender ! time
        }
    }
}
SCALABILITY
Practices

• Stateless
• Datastore
• Replication
• SOA
Small footprint

• 2.7M actors per GB memory
akka.actor.deployment {
  /hello-pool {
    router = round-robin
    nr-of-instances = 5
  }
}

Routers
Remoting

akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider"
  }
  remote {
    transport = "akka.remote.netty.NettyRemoteTransport"
    netty {
      hostname = "127.0.0.1"
      port = 2552
    }
  }
}
FAULT-TOLERANCE
Inspiration

• Erlang
  – Designed by Ericsson in 1986
  – Telecommunications
  – Modern users
    • Facebook
    • Amazon
    • Yahoo
    • T-Mobile
Supervision

- Every actor has a supervisor

“the one who walks the bubbles of space-time”

```
/  
 / 
/  
```

“root guardian”

“guardian”

“system guardian”

user

system

your actor hierarchy

sys support hierarchy

shutdown order
• Who supervises system?
Supervision

• Every actor has a supervision strategy
  – Escalate
  – Stop
  – Restart
  – Resume
Default strategy

• ActorInitializationException => Stop
• ActorKilledException => Stop
• Exception => Restart
• Otherwise => Escalate
Custom strategy

class Supervisor extends Actor {
    override val supervisorStrategy =
        OneForOneStrategy(maxNrOfRetries = 10
                         withinTimeRange = 1 minute) {
            case _: ArithmeticException ⇒ Resume
            case _: NullPointerException ⇒ Restart
            case _: IllegalArgumentException ⇒ Stop
            case _: Exception ⇒ Escalate
        }
}
Example
Q&A