Akka 2.0 and the Actor Model

Thomas Lockney
thomas@lockney.net
@tlockney

What is the actor model?

A brief, high-level look.

The Actor Model

- An actor is defined by three traits:
  - ability to compute things
  - a state model
  - ability to send and receive messages

Actor Systems

- It's rare that it makes sense to use a single actor alone (*there are exceptions!*)
- Systems are how you model computation in your problem domain

Why does this matter?

- Lets you avoid dealing with locks explicitly
- Provides a lot of flexibility for scaling (up *and* down)
- Maps well to a large domain of problems

Actors & Threads

- Actors are not threads
- Actors are not guaranteed to run on any single thread
- Actors can block
- Blocking is bad!

Akka 2.0

- Stable release in March, 2012
- Builds on lessons learned in Akka 1.x, but adds a lot of enhancements:
  - Supervision is built in at the core (more on this later)
  - Actor hierarchy is fundamental
  - Location transparency
  - New dispatch strategies (default uses JSR-166 based Fork-Join impl.)

Core concepts in Akka

- ActorSystem
- ActorContext
- ActorRef
- Supervisors
- Dispatchers
- Transparent remoting
- STM
- ... lots of other things... *we won't be covering all of these anyway*.

ActorSystem

- As the name implies...
- Among other things, this is where you can:
  - create new actors
  - lookup actors by path (I'll explain shortly)
  - schedule tasks
  - shutdown actors or the actor system

ActorContext

- Provides the actor with a view to its context
- Use this to change your behavior
- Find other actors by path or create new ones

ActorRef

An actor ref is an immutable reference to an actor cell

Supervisors

Allow you to define what happens when something goes wrong (e.g., an actor throws an exception)

Not for today...

- STM and Transactors
- Remote actors (unless I can whip together an example in time)

How do I use it?

First things, first:

```
import akka.actor._
val system = ActorSystem("basicSystem")
```

Multiple systems are allowed, but require unique configurations.

Creating your first actor

```
import akka.actor._

object SimpleExample extends App {
  val system = ActorSystem("basicSystem")
  system.actorOf(Props(new Actor {
    def receive = {
      case _ => println("Message received!")
    }
  }))
}
```

What's this Props thing?

- It's a way to provide immutable configuration for actor instances. What does this mean?
- Almost the only thing you should care about: setting routers and dispatchers. E.g.,

```
system.actorOf(Props[FancyActor]
  .withDispatcher("my-fancy-dispatcher")
  .withRouter(RoundRobinRouter()))
```

- We'll come back to this

Isn't there more to it?

- Not really.
- But...

Interacting actors

- Actors can send message to each other, of course
- Off to do some live coding... (watch for flying debris!)

Ping-pong!

```
   import akka.actor._

val system = ActorSystem(“pingPong”)

case class Start(p: ActorRef)

val pinger = system.actorOf(Props(new Actor {
  def receive = {
    case Start(ponger) => ponger ! “ping”
    case _ => sender ! “ping”
  }
}))

val ponger = system.actorOf(Props(new Actor {
  def receive = { case _ => sender ! “pong” }
}))

pinger ! Start(ponger)
```

Routing

- Routing lets you specify where to send messages
- Available routers:
  - RoundRobinRouter
  - SmallestMailboxRouter
  - RandomRouter
  - BroadcastRouter
  - ScatterGatherFirstCompletedRouter

A simple routing example

```
import akka.actor._
import akka.routing._

val system = ActorSystem("mySystem")

class DumbActor extends Actor {
  def receive = {
    case msg => println(msg)
  }
}

val router = system.actorOf(Props[DumbActor]
  .withRouter(RoundRobinRouter(nrOfInstance = 5)))

1 to 50 foreach { i => router ! i }
```

Things to be aware of

A.k.a., best-practices, lessons learned or whatever you want to call it

Create a single, top-level actor

- This gives you a single point of management for your supervisor hierarchy.
- Creating top-level actors is single-threaded.
- If you need more, a handful of top-level actors is fine, but keep the number small.

Don’t block

If you have a large chunk of work to be done, break it into smaller pieces using either actors or futures.

“Tracing”

- Use ActorLogging and LoggingReceive
- turn on logging in the configuration:

```
akka {
        loglevel = “DEBUG”
        event-handlers = [“akka.event.Logging$DefaultLogger”]
        actor {
            debug {
                receive = on
                autoreceive = on
                lifecycle = on
            }
        }
}
```

Other resources

- Akka Homepage: http://akka.io
- Akka-user mailing list:
  https://groups.google.com/forum/?fromgroups#!forum/akka-user
- IRC: irc://irc.freenode.net/%23akka
- Akka team blog: http://letitcrash.com/

Special thanks to...

Harvey for helping me write this.

And more thanks...

To Maggie for helping us review.