The Akka Framework

Simpler Scalability, Fault-Tolerance, Concurrency and Remoting through Actors

Dean Wampler

dean@deanwampler.com
@deanwampler

What Is Akka

- http://akkasource.org
  - Source at http://github.com/jboner/akka
  - Documentation at http://doc.akkasource.org
- A Scala and Java framework for scalability, fault-tolerance, concurrency and remoting through actors.
- Inspired by Erlang OTP.
- Started and led by Jonas Bonér.
  - Co-creator of AspectWerkz.
  - Worked on JRocket and Terracotta.
- At V0.6. V0.7 released next week?

Highlights (1/2)

(Adapted from the http://akkasource.org web site.)

- **Simpler Concurrency**: Write simpler concurrent (yet correct) applications using Actors, STM & Transactors (transactional actors).
- **Event-driven Architecture**: The perfect platform for asynchronous, event-driven architectures. Never block.
- **True Scalability**: Scale out on multi-core or multiple nodes using asynchronous message passing.

Highlights (2/2)

- **Fault-tolerance**: Embrace failure. Write applications that self-heal using Erlang-style Actor supervisor hierarchies.
- **Transparent Remoting**: Remote Actors give you a high-performance, transparent-distributed programming model.
- **Scala and Java APIs**: Scala and Java APIs, as well as Spring and Guice integration. Deploy in your application server or run it stand-alone.

Organization: Core Modules

The “core” modules at the heart of Akka.

MODULE	DESCRIPTION
Core	Actors, remote actors, transactors and STM modules.
Util	Configuration, logging, UUID and other utilities.
Java Util	Java utilities.

Organization: Add-on Modules (1/2):

Optional modules for other features.

MODULE	DESCRIPTION

Microkernel	A standalone kernel for running Akka-based applications.
Security	Adapted from Lift's authentication module.
Persistence	Redis, Mongo, Cassandra, and common (shared) modules.
REST	Module for REST-ful web services.
Comet	Module for Comet "persistent" connections.

Organization: Add-on Modules (2/2):

MODULE	DESCRIPTION
Camel	Module for the Apache Camel library, a DSL for routing definitions, Spring configuration definitions, *etc*.
AMQP	Module for AMQP messaging.
Scheduler	For scheduling tasks in the future.
Spring, Guice, and Lift	Integration modules for these frameworks.
Samples	For Security, REST, Chat, and Lift.

(The list is organized slightly differently on the http://doc.akkasource.org site.)

Akka Core: Actors

Most of the Scala's standard actor API is supported.

- Message-passing abstraction for concurrency.
- Originally developed by Hewitt in 1973.
- A clean-room implementation of Scala actors.
  - Improvements and simplifications.
  - (Scala actor syntax patterned after Erlang actors)

Most of the Scala's standard actor API is supported. Here are some examples that use Akka-specific syntax.

An Akka Actor with an Initialization Body and Message Handler

This example uses the `Actor` object's `init` method to create an actor.

```
1. import se.scalablesolutions.akka.actor.Actor._
2.
3. val a = Actor.init {
4.   ... // initializer block (optional)
5. } receive {
6.   // Usual Scala PartialFunction stuff
7.   case msg => ... // handle message
8. }
```

An Akka Actor with a LifeCycle

The `LifeCycle` specification allows this actor to be managed by an Erlang-style supervisor.

```
1. import se.scalablesolutions.akka.actor.Actor._
2.
3. val a = actor(LifeCycle(Temporary)) {
4.   ... // init stuff
5. } receive {
6.   case msg => ... // handle message
7. }
```

An Akka Actor with a LifeCycle

This argument to `LifeCycle` is a `se.scalablesolutions.akka.config.ScalaConfig.Scope` object, one of the following:

- `Temporary`: Do not restart the actor if it crashes.
- `Permanent`: Restart the actor when necessary; keep it running permanently.

`LifeCycle` can also take a second argument of callbacks to invoke when a supervisor restarts a (dead) `Permanent` actor.

Spawn a Worker Task

When you don't need to send it messages...

```
import se.scalablesolutions.akka.actor.Actor._

spawn {
  ... // do stuff
}
```

Send a Message to an Actor (1/2)

```
// Import Self when the current sender isn't an actor.
import Actor.Sender.Self
...
actor ! "Hello"              // Fire and forget
```

Send a Message to an Actor (2/2)

```
...
(actor !! "Hello") match {  // fire and wait for reply...
  case Some(reply) => ...   // handle reply
  case None =>        ...   // got a timeout, handle that
}

val future = actor !!! "Hello"   // fire and return "future"
```

Forward a Message

Retains the original sender.

```
actor.forward("Hello")
```

"HotSwap" - Replace an Actor's Message Loop

Send a message `HotSwap` with the new `PartialFunction` defining the new message loop for the Actor.

```
actor ! HotSwap(Some({
  case message => println("Hotswapped body...")
}))
```

"HotSwap" - Rollback to Previous Version

The old version is kept so you can rollback by sending `HotSwap` with `None`.

```
actor ! HotSwap(None)
```

The original actor theory included this idea!

STM

Software Transactional Memory (STM) is the application of ACID (minus the D) transactional semantics to memory. It has been popularized by Clojure, of course.

STM: Based on 2 Concepts

- **Managed References**: Memory cells, holding an immutable value, that implement CAS (Compare-And-Swap) semantics and are managed and enforced by the STM for coordinated changes across many References.
- **Persistent Data Structures**: Immutable but with constant time access and modification. The use of structural sharing and an insert or update does not ruin the old structure, hence "persistent".

STM: Implementation

Akka's persistent Map and Vector are ports of Clojure's Map and Vector. The Managed References are implemented using the excellent Java Multiverse STM.

Transactors: Transactions + Actors

Transactors combine Actors, which provide concurrency and asynchronous event-based programming, and STM, which provide compositional, transactional, shared state. Hence, transactors provide transactional, compositional, asynchronous, event-based message flows.

If you need durability (full ACID), use a persistent store, not one of the in-memory data structures.

There are several ways to impose "transaction required" semantics:

Extend the `Transactor` Trait

```
1. import se.scalablesolutions.akka.actor.Transactor
2.
3. class MyActor extends Transactor {
4.   ...
5. }
```

Mix in the `TransactionRequired` Trait

```
1. class MyActor extends Actor with TransactionRequired {
2.   ...
3. }
```

Call the `makeTransactionRequired` Method

```
1. class MyActor extends Actor {
2.   makeTransactionRequired
3.   ...
4. }
```

Creating Persistent Data Structures within a Transactor

If you create a transactional Ref, Vector or Map in a Transactor constructor, either declare it `lazy`, to ensure that it is created within the scope of a transaction, or create the data structure inside an `atomic { ... }` block.

Supervision and Fault Handling

Akka subscribes to the *Let It Fail* strategy:

- Don't build complicated logic to attempt recovery when things go wrong.
- Distribute behavior among many actors
- When one of them gets into trouble, kill it and restart it.

Supervisory Actors

Actors are organized into supervisory hierarchies.

A supervisor starts, monitors, and stops other actors. There are two restart strategies:

- **One for One**: Only restart the actor that has crashed. Used when the actors are autonomous.
- **All for One**: Restart all the actors the supervisor manages. Used when the actors depend on each other.

Supervisor Example

```
01. object factory extends SupervisorFactory(
02.   SupervisorConfig(
03.     RestartStrategy(AllForOne, 3, 1000, List(classOf[Exception])),
04.     Supervise(
05.       new MyActor1,
06.       LifeCycle(Permanent))
07.     ::
08.     Supervise(
09.       new MyActor2,
10.       LifeCycle(Permanent))
11.     :: Nil))
12
```

```
    }
13. val supervisor = factory.newInstance
14. supervisor.start // link and start up all actors
```

Other Core Modules

MODULE	DESCRIPTION
Active Objects	How Java is supported; instrumented POJOs using the AspectWerkz weaver.
Remote Actors	Mostly-transparent remote message passing.
Dispatcher	Configure and optimize messaging infrastructure, *etc*.
Serialization	JSON, Protocol Buffers, and SBinary.
Configuration	Conventional text-based configuration files.
Cluster Membership	JGroups (P2P), Shoal (also P2P + JXTA), ZooKeeper (planned).

Microkernel Module

- To start the kernel invoke:
  - `java -jar $AKKA_HOME/dist/akka-<version>.jar`
- Wait for all services to start up.
- Profit!

Configuration

The kernel will look for a configuration file `akka.conf`, *e.g.*, in `$AKKA_HOME/config/akka.conf` or on the `CLASSPATH`.

Some of the things you can configure:

- Logging.
- Actor and STM default properties.
- Persistence store properties.
- Server and client hostnames and ports.
- …

Persistence Module

Akka provides persistent data structures using the STM API. They are really intended to provide the "D" for ACID persistence, as they back STM Maps, Vectors and References. The API doesn't provide a general-purpose DSL for persistence.

DATA STRUCTURE	DESCRIPTION
`PersistentMap`	Implements `scala.collection.mutable.Map`
`PersistentVector`	Implements `scala.RandomAccessSeq`
`PersistentRef`	Compare and Set (CAS) cell

Persistence Module

They are backed by one of the three (at this time) supported *NoSQL* datastores:

- Cassandra
- MongoDB
- Redis

Cassandra Example (1/5)

Requires an Akka configuration file, `akka-reference.conf`, for details like the host and port, and a Cassandra configuration file `storage-conf.xml`.

Create one of the persistent data structures:

```
val map =    CassandraStorage.newMap(id)     # omit (id) => will be autogenerated.
val vector = CassandraStorage.newVector(id)
val ref =    CassandraStorage.newRef(id)
```

Cassandra Example (2/5)

Retrieve a persistent data structure based on an explicit id (will be created if it doesn't exist):

```
val map =    CassandraStorage.getMap(id)
val vector = CassandraStorage.getVector(id)
val ref =    CassandraStorage.getRef(id)
```

Cassandra Example (3/5)

Akka also provides a direct Cassandra API. It abstracts away session pooling, protocol *etc.* and lets us by-pass the STM persistence abstractions (Map, Vector and Ref).

```
import se.scalablesolutions.akka.state.CassandraSessionPool

val sessions = new CassandraSessionPool(
  keyspace,
  StackPool(SocketProvider(hostname, port)),
  protocol,
  ConsistencyLevel.QUORUM)
...
```

Cassandra Example (4/5)

```
...

// insert a column
sessions.withSession { session =>
  session ++| (key, new ColumnPath(columnFamily, null, columnName),
               serializer.out(element), System.currentTimeMillis)
}
...
```

Cassandra Example (5/5)

```
...
// retrieve a column
val column: Option[ColumnOrSuperColumn] = sessions.withSession { session =>
  session | (key, new ColumnPath(columnFamily, null, columnName))
}

// get a range of colums
val columns: List[ColumnOrSuperColumn] = sessions.withSession { session =>
  session / (key, columnParent, slicePredicate, isAscending, count)
}
```

Persistence Module: MongoDB and Redis

There are similar APIs for MongoDB and Redis.

(But I don't think there are the same sorts of column "operators".)

Demo!

- `akka-sample-rest-scala`: Simple REST'ful Service.
- `akka-sample-chat`: Simple Chat Service.

Recap (1/2)

- **Simpler Concurrency**: Write simpler concurrent (yet correct) applications using Actors, STM & Transactors (transactional actors).
- **Event-driven Architecture**: The perfect platform for asynchronous, event-driven architectures. Never block.
- **True Scalability**: Scale out on multi-core or multiple nodes using asynchronous message passing.

Recap (2/2)

- **Fault-tolerance**: Embrace failure. Write applications that self-heal using Erlang-style Actor supervisor hierarchies.
- **Transparent Remoting**: Remote Actors give you a high-performance, transparent-distributed programming model.
- **Scala and Java APIs**: Scala and Java APIs, as well as Spring and Guice integration. Deploy in your application server or run it stand-alone.

Thanks!

- http://akkasource.org
- dean@deanwampler.com
- @deanwampler (Twitter)
- polyglotprogramming.com