akka-examples (0.0.1) [Build Status]

Maksim Kostromin

Version 0.0.1, 2019-06-22 15:17:53 UTC

Table of Contents

Introduction

This documentation contains some help to examples from akka-examples repository. Akka playground projects

This repository contains Akka playground projects examples.

Documentation

- Basic actor
- Learning Akka projects
- ping-pong: yet another java gradle / maven docker akka jar...
- java (gradle / maven) starter in docker
- kotlin (gradle / maven) starter in docker
- scala (gradle / maven) starter in docker

TODO: - spring-boot + akka 1 and 2

Chapter 1. Implementation

Chapter 2. basic getting started (java 12)

```
@Value(staticConstructor = "withBody")
class Message {
  private final String body;
}

class HelloWorld extends AbstractLoggingActor {

  @Override
  public Receive createReceive() {
    return receiveBuilder().match(Message.class, message -> onMessage(message))
                           .build();
  }

  private void onMessage(Message message) {
    log().info("received {}", message);
  }
}

@Configuration
class AkkaCfg {

  @Bean
  ActorSystem helloSystem() {
    return ActorSystem.create("hello-world");
  }

  @Bean
  Props helloProps() {
    return Props.create(HelloWorld.class);
  }

  @Bean
  ActorRef helloRef(ActorSystem helloSystem, Props helloProps) {
    return helloSystem.actorOf(helloProps, "hello-actor");
  }
}

@Configuration
@ComponentScan(basePackageClasses = App.class)
public class App {

  public static void main(String[] args) {
    var ctx = new AnnotationConfigApplicationContext(App.class);
    var helloRef = ctx.getBean(ActorRef.class);

    helloRef.tell(Message.withBody("Max"), ActorRef.noSender());
    helloRef.tell(Message.withBody("Bob"), ActorRef.noSender());
```

```
    helloRef.tell(Message.withBody("Everyone"), ActorRef.noSender());

    Try.run(() -> TimeUnit.SECONDS.sleep(1))
       .andFinally(() -> ctx.getBean(ActorSystem.class).terminate());
  }
}
```

Chapter 3. others

3.1. scala akka sbt IDEA

build, test and run

```
./sbtw clean compile test assembly # run
java -jar target/scala-2.12/*-assembly-*.jar
```

Supervisor actors

```
package daggerok.supervisor

import akka.actor.SupervisorStrategy.{Escalate, Restart, Resume, Stop}
import akka.actor.{Actor, ActorRef, ActorSystem, OneForOneStrategy, Props,
SupervisorStrategy}
import daggerok.supervisor.API.{RestartError, ResumeError, StopError}

object API {
  sealed trait ErrorMessage extends RuntimeException
  case object StopError extends ErrorMessage
  case object ResumeError extends ErrorMessage
  case object RestartError extends ErrorMessage
}

class Child extends Actor {

  override def preStart(): Unit = {
    println("Child: preStart")
    Thread.sleep(100)
    super.preStart()
  }

  override def preRestart(reason: Throwable, message: Option[Any]): Unit = {
    println("Child: preRestart")
    Thread.sleep(100)
    super.preRestart(reason, message)
  }

  override def postRestart(reason: Throwable): Unit = {
    println("Child: postRestart")
    Thread.sleep(100)
    super.postRestart(reason)
  }

  override def postStop(): Unit = {
    println("Child: postStop")
    Thread.sleep(100)
    super.postStop()
```

```
  }

  override def receive: Receive = {
    case "Stop" => throw StopError
    case "Resume" => throw ResumeError
    case "Restart" => throw RestartError
    case _ => throw new RuntimeException
  }
}

class Parent extends Actor {

  var childRef: ActorRef = _

  import scala.concurrent.duration._ // seconds
  override def supervisorStrategy: SupervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 5, withinTimeRange = 1 second) {
      case StopError => Stop
      case ResumeError => Resume
      case RestartError => Restart
      case _: RuntimeException => Escalate
    }

  override def preStart(): Unit = {
    println("Parent: preStart")
    childRef = context.actorOf(Props[Child], "child")
    Thread.sleep(100)
    super.preStart()
  }

  override def receive: Receive = {
    case message =>
      println(s"Parent message: $message")
      childRef ! message
      Thread.sleep(100)
  }
}

object SupervisorApp {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem("supervisor-system")
    val parent = system.actorOf(Props[Parent], "parent")
//    parent! "Stop"
//    parent! "Resume"
    parent! "Restart"
    Thread.sleep(1000)
    system.terminate()
  }
}
```

Tell / Ask to actors

```
package daggerok.tellandaskactors

import akka.actor.{Actor, ActorRef, ActorSystem, Props}
import akka.util.Timeout
import daggerok.tellandaskactors.CheckerAPI.{BlacklistUserResponse, CheckUserRequest,
WhitelistUserResponse}
import daggerok.tellandaskactors.RecorderAPI.CreateUser
import daggerok.tellandaskactors.StorageAPI.AddUser

case class User(name: String)

object RecorderAPI {
  sealed trait RecorderMessage
  case class CreateUser(user: User) extends RecorderMessage
}

class Recorder(checker: ActorRef, storage: ActorRef) extends Actor {
  import scala.concurrent.duration._ // seconds
  implicit val timeout = Timeout(5 seconds)

  import akka.pattern.ask // actor ask map
  import scala.concurrent.ExecutionContext.Implicits.global
  override def receive: Receive = {
    case CreateUser(user) =>
      checker ? CheckUserRequest(user) map {
        case WhitelistUserResponse(user) =>
          storage ! AddUser(user)
        case BlacklistUserResponse(user) =>
          println(s"Recorder black user: $user")
      }
    case _ => println("Checker: unknown message received")
  }
}

object CheckerAPI {
  sealed trait CheckerRequest
  case class CheckUserRequest(user: User) extends CheckerRequest

  sealed trait CheckerResponse
  case class BlacklistUserResponse(user: User) extends CheckerResponse
  case class WhitelistUserResponse(user: User) extends CheckerResponse
}

class Checker extends Actor {
  val blackList = List("bad", "black", "evil")
  override def receive: Receive = {
    case CheckUserRequest(user) if (blackList.exists(name =>
user.name.contains(name))) =>
      sender() ! BlacklistUserResponse(user)
    case CheckUserRequest(user) =>
```

```
      sender() ! WhitelistUserResponse(user)
    case _ => println("Checker: unknown message received")
  }
}

object StorageAPI {
  sealed trait StorageMessage
  case class AddUser(user: User) extends StorageMessage
}

class Storage extends Actor {
  private var users = List.empty[User]
  override def receive: Receive = {
    case AddUser(user) =>
      println(s"user $user added")
      users = user :: users
    case _ => println("Storage: unknown message received")
  }
}

object TellAndAskApp {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem("tell-ask-system")
    val checker = system.actorOf(Props[Checker], "checker")
    val storage = system.actorOf(Props[Storage], "storage")
    val recorder = system.actorOf(Props(new Recorder(checker, storage)), "recorder")

    recorder ! CreateUser(User("white"))
    recorder ! CreateUser(User("black"))
    recorder ! CreateUser(User("one more white"))
    Thread.sleep(1000)
    system.terminate()
  }
}
```

Music Player

```
package daggerok.musicplayer

import akka.actor.{Actor, ActorSystem, Props}
import daggerok.musicplayer.MusicController.{PlayMsg, StopMsg}
import daggerok.musicplayer.MusicPlayer.{StartMusicMsg, StopMusicMsg}

object MusicController {
  sealed trait MusicControllerMessage
  case object PlayMsg extends MusicControllerMessage
  case object StopMsg extends MusicControllerMessage

  def props = Props[MusicControllerActor]
}

class MusicControllerActor extends Actor {
  override def receive: Receive = {
    case PlayMsg => println("playing music...")
    case StopMsg => println("music is stopped")
  }
}

object MusicPlayer {
  sealed trait MusicPlayerMessage
  case object StopMusicMsg extends MusicPlayerMessage
  case object StartMusicMsg extends MusicPlayerMessage
}

class MusicPlayerActor extends Actor {
  override def receive: Receive = {
    case StopMusicMsg => println("I don't wanna stop!")
    case StartMusicMsg =>
      val ctrlActor = context.actorOf(MusicController.props, "ctrl-actor")
      ctrlActor ! PlayMsg
    case _ => println("received unknown message.")
  }
}

object MusicPlayerApp {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem("music-system")
    val actor = system.actorOf(Props[MusicPlayerActor], "mp-actor")

    actor ! StartMusicMsg
    actor ! StopMusicMsg
    Thread.sleep(1000)
    system.terminate()
  }
}
```

Hello World

```
package daggerok.helloworld

import akka.actor.{Actor, ActorSystem, Props}

case class HelloMessage(name: String)

class HelloActor extends Actor {
  override def receive: Receive = {
    case HelloMessage(name) => println(s"Hello, $name!")
  }
}

object HelloWorldApp {
  def main(args: Array[String]): Unit = {
    val helloSystem = ActorSystem("hello-actor-system")
    val helloActor = helloSystem.actorOf(Props[HelloActor], "hello-actor")

    helloActor ! HelloMessage("Максимко")
    Thread.sleep(1000)
    helloSystem.terminate()
  }
}
```

3.2. Java Gradle Maven Docker Starter

This is just an Akka java gradle / maven / docker / starter project...

build and test

```
docker-compose down -v; ./mvnw clean package; ./gradlew clean build; docker-compose up
--build --force-recreate --remove-orphans

# or
docker-compose down -v
./mvnw clean package
./gradlew clean build
docker-compose up --build --force-recreate --remove-orphans
```

links:

- Akka docs

3.3. Kotlin Gradle Maven Docker Starter

This is just an Akka kotlin gradle / maven / docker / starter project...

build and test

```
docker-compose down -v; ./mvnw clean package; ./gradlew clean build; docker-compose up
--build --force-recreate --remove-orphans

# or
docker-compose down -v
./mvnw clean package
./gradlew clean build
docker-compose up --build --force-recreate --remove-orphans
```

links:

- kotlin with maven
- Akka docs

3.4. akka

generated using jvm yeoman generator

build

```
./mvnw clean package com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:up
./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:down

./gradlew clean build composeUp
./gradlew composeDown
```

3.5. another-akka-try

test

```
./gradlew clean installDist
bash build/install/another-akka-try/bin/another-akka-try

./gradlew clean distZip
unzip -o build/distributions/*.zip -d /tmp
bash /tmp/another-akka-try-0.0.1/bin/another-akka-try

./mvnw
java -jar target/*-all.jar
```

build

```
./mvnw clean package com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:up
./mvnw com.dkanejs.maven.plugins:docker-compose-maven-plugin:1.0.1:down

./gradlew clean build composeUp
./gradlew composeDown
```

generated using jvm yeoman generator

Chapter 4. Links

- Learning Akka
- Videos: Intro to Akka
- Reactive Programming with Akka
- Akka and the Zen of Reactive System Design

Chapter 5. Enjoy! :)