Testing Akka Typed Behaviors

Programming Reactive Systems

Roland Kuhn

Testing Actors

Testing *actors* is hard:

- all messaging is asynchronous, preventing the use of normal synchronous testing tools
- asynchronous expectations introduce the concept of a timeout
- test procedures become non-deterministic

Testing Actors

Testing *actors* is hard:

- all messaging is asynchronous, preventing the use of normal synchronous testing tools
- asynchronous expectations introduce the concept of a timeout
- test procedures become non-deterministic

Testing *actor behaviors* should be easy:

- functions from input message to next behavior
- any effects are preformed during computation of the function
- full arsenal of testing for functions is available

Akka Typed BehaviorTestKit

Place the behavior into a container that emulates an `ActorContext`:

```
val guardianKit = BehaviorTestKit(guardian)
```

Akka Typed BehaviorTestKit

Place the behavior into a container that emulates an `ActorContext`:

```
val guardianKit = BehaviorTestKit(guardian)
```

Inject a message:

```
guardianKit.ref ! <some message>
guardianKit.runOne()
```

Effect tracking

`BehaviorTestKit` tracks the following internal effects:

- child actor creation and forced termination
- DeathWatch: watching and unwatching
- setting receive timeout
- scheduling a message

Effect tracking

`BehaviorTestKit` tracks the following internal effects:

- child actor creation and forced termination
- DeathWatch: watching and unwatching
- setting receive timeout
- scheduling a message

Inspect the effects:

```
guardianKit.isAlive must be(true)
guardianKit.retrieveAllEffects() must be(Seq(Stopped("Bob")))
```

Effect tracking

`BehaviorTestKit` tracks the following internal effects:

- child actor creation and forced termination
- DeathWatch: watching and unwatching
- setting receive timeout
- scheduling a message

Inspect the effects:

```
guardianKit.isAlive must be(true)
guardianKit.retrieveAllEffects() must be(Seq(Stopped("Bob")))
```

Inspecting effects should be considered whitebox testing!

The standard approach is to observe only messages.

Akka Typed TestInbox

Create a receptacle for messages from the behavior under test:

```
val sessionInbox = TestInbox[ActorRef[Command]]()
```

Akka Typed TestInbox

Create a receptacle for messages from the behavior under test:

```
val sessionInbox = TestInbox[ActorRef[Command]]()
```

Exchange messages between test procedure and actor behavior:

```
guardianKit.ref ! NewGreeter(sessionInbox.ref)
guardianKit.runOne()
val greeterRef = sessionInbox.receiveMessage()
sessionInbox.hasMessages must be(false)
```

Testing child actor behaviors

```
// retrieve child actor's behavior testkit
val greeterKit = guardianKit.childTestKit(greeterRef)
```

Testing child actor behaviors

```
// retrieve child actor's behavior testkit
val greeterKit = guardianKit.childTestKit(greeterRef)

// test a greeting
val doneInbox = TestInbox[Done]()
greeterRef ! Greet("World", doneInbox.ref)
greeterKit.runOne()
doneInbox.receiveAll() must be(Seq(Done))
```

Testing child actor behaviors

```
// retrieve child actor's behavior testkit
val greeterKit = guardianKit.childTestKit(greeterRef)

// test a greeting
val doneInbox = TestInbox[Done]()
greeterRef ! Greet("World", doneInbox.ref)
greeterKit.runOne()
doneInbox.receiveAll() must be(Seq(Done))

// test shutting down the greeter
greeterRef ! Stop
greeterKit.runOne()
greeterKit.isAlive must be(false)
```

Summary

In this video we have seen:

- how to synchronously and deterministically test actor behaviors
- that internal effects are observable, but using them is reserved for whitebox testing