ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Supervision revisited

Programming Reactive Systems

Roland Kuhn

Supervision revisited

Supervision is the backbone of actor systems:

- each actor may fail, in which case its supervisor will step in
- the supervisor can react to failure, e.g. by restarting the actor

Supervision revisited

Supervision is the backbone of actor systems:

- each actor may fail, in which case its supervisor will step in
- the supervisor can react to failure, e.g. by restarting the actor

There are some issues with Akka untyped supervision:

- the failed actor is paused until the supervisor reacts
- the failure may need to travel across the network
- the failure notice contains too much information by default

Supervision in Akka Typed

Supervisor decorates the child actor’s behavior with a selective restart strategy.

Supervision in Akka Typed

Supervisor decorates the child actor’s behavior with a selective restart strategy.

The restart decision is taken locally, within the child actor’s context.

Supervision in Akka Typed

Supervisor decorates the child actor's behavior with a selective restart strategy.

The restart decision is taken locally, within the child actor's context.

Any unhandled failure will lead to the immediate termination of the failed actor.

Supervision in Akka Typed

Supervisor decorates the child actor's behavior with a selective restart strategy.

The restart decision is taken locally, within the child actor's context.

Any unhandled failure will lead to the immediate termination of the failed actor.

Remote supervision across the network is no longer supported.

Starting a supervised actor

The supervisor may add supervision to any behavior:

```
ctx.spawnAnonymous(
    Behaviors.supervise(actor)
        .onFailure[ArithmeticException](SupervisorStrategy.restart))
```

Starting a supervised actor

The supervisor may add supervision to any behavior:

```
ctx.spawnAnonymous(
    Behaviors.supervise(actor)
        .onFailure[ArithmeticException](SupervisorStrategy.restart))
```

This also allows the child actor to add its own supervision as desired.

Information flow from actor to supervisor

Akka Typed shields the supervisor from the failed actor’s state:

- an exception may reference any object for transporting information
- the failure may well be intrinsic to the request
- keeping the exception confined to its origin prevents mistakes

Information flow from actor to supervisor

Akka Typed shields the supervisor from the failed actor's state:

- an exception may reference any object for transporting information
- the failure may well be intrinsic to the request
- keeping the exception confined to its origin prevents mistakes

In case assistance is needed, regular messaging is the best choice.

Supervision implementation

Supervision does not need any special features, it is just a behavior decorator:

```
def supervise[T](behavior: Behavior[T]): Behavior[T] =
    new Restarter(behavior, behavior)

class Restarter[T](initial: Behavior[T], behavior: Behavior[T])
        extends ExtensibleBehavior[T] {
    import akka.actor.typed.ActorContext

    def receive(ctx: ActorContext[T], msg: T): Behavior[T] = ???

    def receiveSignal(ctx: ActorContext[T], msg: Signal): Behavior[T] = ???
}
```

Executing another behavior

```scala
def receive(ctx: ActorContext[T], msg: T): Behavior[T] = {
    import akka.actor.typed.Behavior.{ start, canonicalize, validateAsInitial,
                                       interpretMessage }
    try {
        val started = validateAsInitial(start(behavior, ctx))
        val next = interpretMessage(started, ctx, msg)
        new Restarter(initial, canonicalize(next, started, ctx))
    } catch {
        case _: ArithmeticException =>
            new Restarter(initial, validateAsInitial(start(initial, ctx)))
    }
}
```

Summary

In this video we have seen:

- how supervision in Akka Typed differs from untyped Akka
- that Akka Typed supervision can be implemented in user code
- how to write a behavior decorator that runs another behavior