Kapitel 1

Reaktive Systemarchitekturen nach dem Actors-Modell mit Akka

…

1.1 Router

Der Zweck von Routern ist es, Nachrichten zwischen einer Gruppe von möglichen Actors (*«Routees»*) ähnlich einem *Load-Balancer* zu verteilen. Router ähneln insofern Actoren, dass Sie Nachrichten empfangen und an andere Actors weiterleiten. Allerdings sind sie dabei effizienter, weil Sie nicht die Standardstrukturen bspw. von *Mailboxes* verwenden.

1.1.1 Eingebaute Router-Typen

`RoundRobinRoutingLogic` Nachrichten werden «reihum» an die zur Verfügung stehenden *Routees* gesendet. Man kann dabei nicht beeinflussen, welcher *Routee* der erste in der Reihenfolge ist.

Dieser Router ist besonders geeignet, wenn die zu verteilenden Aufgaben, die hinter den weitergeleiteten Nachrichten stehen, mit einem ähnlichen Aufwand gelöst werden und wenn die *Routees* vergleichbare oder gemeinsam geteilte Ressourcen zur Verfügung haben. Falls die Aufgaben oder die Ressourcen jedoch ungleichmäßig sind, entsteht eine «Unwucht» bei der Verarbeitung.

`RandomRoutingLogic` Nachrichten werden an einen zufällig ausgewählten Routee gesendet.

`SmallestMailboxRoutingLogic` Nachrichten werden zufällig an einen der *Routees* mit der geringsten Anzahl von wartenden (unverarbeiteten) Nachrichten in der Mailbox gesendet. Allerdings kann solch ein Router nicht auf die Mailbox-Größe entfernter *Routees* zugreifen. Deshalb haben *remote* Actors bei dieser Routing-Logik immer die geringste Priorität.

> Selbst wenn eingehende Nachrichten immer an die kleinste Mailbox gesendet werden, gibt es keine Gewähr dafür, ob die Aufgaben hinter den Nachrichten einer volleren Mailbox nicht in Summe schneller abgearbeitet sind.

`BroadcastRoutingLogic` Nachrichten werden vervielfältigt und jeder *Routee* erhält ein Exemplar der zu routenden Nachricht.

`ScatterGatherFirstCompletedRoutingLogic` Wie bei der `BroadcastRouterLogic` werden Nachrichten an alle *Routees* weitergeleitet. Hier wird aber speziell das «`ask`»-Kommunikationsmuster unterstützt: Die zurückgegebene Antwort ist die des ersten der *Routees*, der eine Antwort zurückgibt. Der Anwendungszweck ist bei zeitkritischen Aufgaben, wenn mehrere Actors parallel dasselbe Problem zu lösen versuchen. Das am schnellsten erzeugt Ergebnis wird benutzt, die anderen werden ignoriert.

Die folgenden Gründe können dazu führen, dass die *Routees* unterschiedlich lange für die Bearbeitung derselben Aufgabe brauchen:

- Jeder Routee verwendet einen anderen Lösungsweg oder Algorithmus.
- Eine Zufallskomponente ist bei der Lösung beteiligt.
- Verteilte Ressourcen mit unterschiedlicher Auslastung oder Leistungsfähigkeit werden zur Lösung (also zur Ausführung der Routees) benutzt.

1.1.2 Router-Verwendung

Listing 1.1 zeigt exemplarisch, wie ein Router mit `RoundRobinRoutingLogic`-Verhalten Nachrichten zwischen *Routees* vom Typ `Worker` aufteilt: Statt Nachrichten mit «`tell`» oder «`ask`» zu senden, werden sie mit «`route`» über den Router `router` verteilt.

Listing 1.1: Programmtische Verwendung eines `RoundRobinRoutingLogic`-Routers (*Routees* vom Typ `Worker`)

```
1 List<Routee> routees = new ArrayList<Routee>();
2 for (int i = 0; i < 5; i++) {
3     ActorRef r = getContext().actorOf(Props.create(Worker.class));
4     getContext().watch(r);
```

```
5     routees.add(new ActorRefRoutee(r));
6 }
7 router = new Router(new RoundRobinRoutingLogic(), routees);
8 // ...
9 router.route(message, getSender());
```

Neben der programmatischen Verwendung von Routern kann auch der konfigurative Ansatz verwendet werden. Listing 1.2 zeigt einen Teil der Konfiguration für einen zum Listing 1.1 vergleichbaren Router.

Listing 1.2: Konfiguration eines `RoundRobinRoutingLogic`-Routers (*Routees* vom Typ `Worker`)

```
1 akka.actor.deployment {
2   /parent/routerGroup {
3     router = round-robin-group
4     nr-of-instances = 5
5   }
6 }
```

Aus solch einer Konfiguration kann der Router zusammen mit seinen *Routees* programmatisch erzeugt werden:

```
ActorRef router = getContext().actorOf(FromConfig.getInstance().props
    (), "routerGroup");
```

über den dann wieder Nachrichten wie in Listing 1.1 versendet werden können:

```
router.route(message, getSender());
```

Router für verteilte *Routees*

Exkurs

Der konfigurative Ansatz ermöglicht zudem analog zur Definition eines Routers mit lokalen Routees die effektive Nutzung verteilter Ressourcen. In Listing 1.3 ist die Konfiguration eines `RoundRobinRoutingLogic`-Routers und einer verteilten Gruppe von drei *Routees* vom Typ `Worker` auf den Hosts `host1`, `host2` und `host3` dargestellt.

Listing 1.3: Konfiguration eines Routers und einer verteilten Gruppe von drei *Routees*

```
1 akka.actor.deployment {
2   /parent/remoteGroup {
3     router = round-robin-group
4     routees.paths = [
5       "akka.tcp://test@host1:4711/user/workers/w1",
6       "akka.tcp://test@host2:4711/user/workers/w1",
7       "akka.tcp://test@host3:4711/user/workers/w1"]
8   }
9 }
```

Listing 1.4: Konfiguration eines Routers und zehn *Routees* gleichmäßig auf `host2` und `host3` verteilt

```
1 akka.actor.deployment {
2   /parent/remotePool {
3     router = round-robin-pool
4     nr-of-instances = 10
5     target.nodes = ["akka.tcp://app@host2:4711", "akka.tcp://
          app@host3:4711"]
6   }
7 }
```

In Listing 1.4 ist die Konfiguration eines `RoundRobinRoutingLogic`-Routers und einer verteilten Gruppe von zehn *Routees* gleichmäßig verteilt auf die Hosts `host2` und `host3` gezeigt