

Frontend Interface



Methoden zur Minimierung von blockenden Aufrufen im Routing

Matthias Jahn, Frederick Schäfer, André Wolski

Scala Futures

- Konstrukt zur nebenläufigen Ausführung von Code
- abhängige Berechnungen einfach Erstellbar

Beispiel:

```
import scala.concurrent._
import scala.concurrent.duration._
import scala.concurrent.ExecutionContext.Implicits.global

val piFuture = Future { calcPI() }
val resultFuture = piFuture map { pi => 2 * pi * r }
val result = Await.result(resultFuture, 5 seconds)
```

For Comprehension

Verschachtelung zweier Futures mit flatMap:

```
val futureA = Future { calcA() }  
val futureB = Future { calcB() }  
val result = futureA flatMap {  
  a => futureB map { b => a * b }  
}
```

Verschachtelung mit For:

```
val futureA = Future { calcA() }  
val futureB = Future { calcB() }  
val result = for(  
  a <- futureA  
  b <- futureB  
) yield a * b
```

```
val result = for(  
  a <- Future { calcA() }  
  b <- Future { calcB() }  
) yield a * b
```

nebenläufig

sequenziell

Futures reduzieren

`Seq[Future] ~> Future[Seq]`

```
val f1: Seq[Future[Foo]] = for .. yield Future {...}
val f2: Future[Seq[Foo]] = Future.sequence(f1)
```

`Future.reduce`

```
val f1: Seq[Future[Foo]] = for .. yield Future {...}
val f2: Future[Foo] = Future.reduce(f1){
  (a, b) => Foo(a.x + b.x)
}
```

`Future.fold`

```
val f1: Seq[Future[Foo]] = for .. yield Future {...}
val f2: Future[Double] = Future.fold(f1)(42.13){
  (sum, a) => sum + a.x
}
```

- Akka Ask („?“) Methode liefert ein Future[Any]
- Weiterleitung eines Ergebnisses mit *pipeTo* möglich
- **Achtung:** Kapselung des internen Aktorzustandes beachten!
 - <http://doc.akka.io/docs/akka/current/general/jmm.html#jmm-shared-state>
 - "sender" ist ein *def* und könnte sich zwischenzeitlich bei Eintreffen einer anderen Nachricht ändern
- <http://doc.akka.io/docs/akka/current/scala/futures.html>

Futures und Akka

Beispiel:

```
import akka.pattern.{ask, pipe}

val from = sender

val result: Future[Info] = (actorA ? GetInfo).mapTo[Info]

val wrapped: Future[InfoRequestResult] = result.map(
  info => InfoRequestResult(info, from)
)

wrapped pipeTo actorB
```

Futures und Spray-Routing

- direkte Unterstützung für Futures in complete()
- seit Version 1.2-M8 zusätzlich mit onSuccess, onFailure und onComplete Direktiven

Vorher:

```
val future = (subjectProviderManager ? GetAllProcessInstances(userId))  
    .mapTo[AllProcessInstancesAnswer]  
val result = Await.result(future, timeout.duration)  
complete(result.processInstanceInfo)
```

Nachher:

```
val future = (subjectProviderManager ? GetAllProcessInstances(userId))  
    .mapTo[AllProcessInstancesAnswer]  
complete(future.map(result => result.processInstanceInfo))
```

Beispiel für onSuccess

Vorher:

```
val userFuture = (persistenceActor ? Users.Read.ByIdWithIdentities(id)).mapTo[...]  
val user = Await.result(userFuture, timeout.duration)  
if (user.isDefined) {  
    complete(UserWithMail(...))  
} else {  
    complete(StatusCodes.NotFound)  
}
```

Nachher:

```
val userFuture = (persistenceActor ? Users.Read.ByIdWithIdentities(id)).mapTo[...]  
onSuccess(userFuture) {  
    user =>  
        if (user.isDefined) {  
            complete(UserWithMail(...))  
        } else {  
            complete(StatusCodes.NotFound)  
        }  
}
```


Anweisungen innerhalb des Routingpfades

- Anweisungen können bei der Erstellung der Route oder pro Anfrage ausgeführt werden

```
val route: Route = get {  
  val time = currentTime()  
  complete("Current time: " + time)  
}
```

bedeutet in etwa:

```
val route: Route = {  
  val inner = {  
    val time = currentTime()  
    { ctx => ctx.complete("Current time: " + time) }  
  }  
  get.apply(inner)  
}
```

Wird bei der Erstellung der Route ausgeführt

<http://spray.io/documentation/1.2-M8/spray-routing/advanced-topics/understanding-dsl-structure/>

Aufrufe innerhalb des Routingpfades

Lösungsmöglichkeit 1: alle Anweisungen innerhalb von *complete*

```
get {  
  complete {  
    val time = currentTime()  
    "Current time: " + time  
  }  
}
```

Lösungsmöglichkeit 2: Nutzung der *dynamic* Direktive

```
get {  
  dynamic {  
    val time = currentTime()  
    complete("Current time: " + time)  
  }  
}
```