Akka et le modèle d'acteur

Jacob Tardieu

21 février 2014

1 Le language Scala

Scala [3] est un language multi-paradigme qui a vu le jour en 2003 à l'École Polytechnique Fédérale de Lausanne. C'est un langage à la fois orienté objet (tel que Java) et mais également fonctionnel. La syntaxe est fortement inspirée de de celle de Java et il est possible d'avoir accès à toutes les bibliothèques Java. Scala est prévu pour être compilé en bytecode Java, donc exécutable sur la JVM. L'intéropérabilité avec Java est facilitée par le fait qu'il est possible d'utiliser dans un même projet les deux langages (du code Scala peut être invoqué à partir de code Java). Pour plus de détails sur le langage Scala, voir [5] et [3].

2 Le framework Akka

Le framework Akka [2] est un outil open-source dont le but est de faciliter la contruction d'applications concurrentes et réparties sur la JVM. Akka permet d'utiliser un modèle d'acteurs pour gérer la concurrence.

2.1 La notion de `Future`

Un outil important apporté par Akka au langage Scala est la notion de `Future`. Une `Future` est un objet qui contient une valeur qui n'est pas disponible tout de suite mais qui le sera à un moment donné. Cela permet d'écrire du code non bloquant et d'utiliser des *callbacks* afin de traiter l'information au moment où elle est reçue sans avoir à l'attendre pour exécuter le reste du code.

Les `Future`s permettent donc une parallélisation de manière efficace et non-bloquante. Elles sont maintenant intégrées au langage Scala. Voir [1] pour plus de détails.

2.2 Les acteurs Akka

Les acteurs Akka sont des entités légères qui traitent les messages reçus de manière asynchrone au moment de leur réception (c'est un modèle *event-*

driven). La concurrence est donc uniquement gérée par *message passing*, c'est à dire uniquement par les messages : aucune donnée mutable ou mécanisme de synchronisation ne sont utilisés par Akka.

De plus, les acteurs interagissent de la même manière qu'ils soient sur une même machine ou sur des machines séparées, cela permet très facilement soit d'utiliser plus d'acteurs sur la même machine (*scale up*) soit d'utiliser plus de machines afin de répartir la charge (*scale out*).

Enfin, Akka utilise une hiérarchie entre les acteurs (acteurs parents et acteurs enfants) ce qui permet une supervision et une meilleure tolérance aux pannes.

2.3 Utilisation des acteurs

Un acteur reçoit les messages via une méthode `receive` et le *pattern matching* permet de les différencier.

```
class GreetingActor extends Actor with ActorLogging {
  def receive = {
        case Greeting(who) => log.info("Hello " + who)
  }
}
```

Deux méthodes sont utilisées pour envoyer des messages à des acteurs :

- `send` ou `!` qui envoie un message et n'attend pas de réponse (*fire and forget*). Cette méthode est non bloquante : une fois utilisée le programme continue son cours sans se soucier de l'effet de celle-ci.
- `ask` ou `?` qui envoie un message mais récupère la réponse sous forme de `Future`. Le fait que cette méthode retourne une `Future` en fait une méthode non bloquante, mais elle donne la possiblité de récupérer une information directement suite à l'envoi du message.

Akka encourage cependant l'utilisation de la méthode `send` qui correspond parfaitement à la manière de communiquer entre les acteurs.

Références

[1] Scala documentation. Futures and promises. `http://docs.scala-lang.org/overviews/core/futures.html`, 2014. [Online ; accessed 21-February-2014].

[2] Akka website. Akka. `http://akka.io/`, 2014. [Online ; accessed 21-February-2014].

[3] Scala website. The scala programming language. `http://www.scala-lang.org/`, 2014. [Online ; accessed 21-February-2014].

[4] Wikipedia. Akka (toolkit) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Akka_(toolkit)&oldid=593098346, 2014. [Online ; accessed 21-February-2014].

[5] Wikipedia. Scala (programming language) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Scala_(programming_language)&oldid=596058129, 2014. [Online ; accessed 21-February-2014].