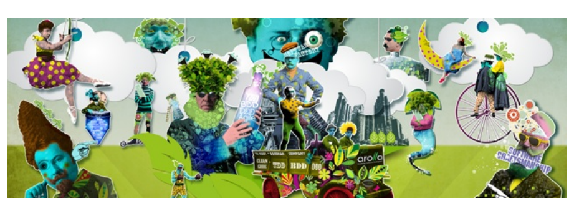


FONCTIONNEL

Listes Scala: méthodes foldLeft et foldRight

by Nouhoum • 27 octobre 2011

Dans ce billet je souhaite vous parler rapidement des méthodes foldLeft et foldRight de l'API des listes en Scala.

foldLeft

foldLeft est une méthode de la classe scala.collection.immutable.List et voici ce que le scaladoc nous en dit:
"*Applies a binary operator to a start value and all elements of this list, going left to right.*"
Scaladoc nous dit que foldLeft applique une fonction prenant deux paramètres (opérateur binaire) à une valeur initiale (appelée **accumulateur** dans le jargon fonctionnel) et à tous les éléments de la liste en partant de la gauche. Voyons attentivement la signature de la méthode pour rendre les choses un peu plus claires!

```
def foldLeft [B] (z : B)(f : (B, A) ⇒ B) : B
```

Ah tiens une chose transparaît dans cette signature: *foldLeft* est une méthode currifiée. Si vous n'avez jamais fait de programmation fonctionnelle auparavant cette notion ne doit pas vous parler. En fait malgré les apparences la méthode foldLeft ne prend pas deux arguments! Voici comment elle fonctionne: elle prend un paramètre z de type B et renvoie une fonction qui prend à son tour un paramètre qui est une fonction de type (B, A) => B. z représente la valeur initiale de l'accumulateur et est du même type que la valeur de retour de foldLeft. A chaque étape la fonction f est appliquée à l'accumulateur et à l'élément courant de la liste. La valeur de l'accumulateur peut changer tout au long du déroulement de l'opération. Prenez une minute pour bien visualiser la chose ce n'est pas si compliqué que cela une fois qu'on a compris la notion. La méthode fold est d'une puissance incroyable! Elle vous permet de faire quasiment de faire toutes les opérations imaginables avec les listes.

Voici un usage simple de *foldLeft*: Calculer la somme des éléments d'une liste d'entiers.

L'accumulateur (la valeur initiale) est égal à zéro et l'opérateur binaire est une fonction qui prend deux entiers et fait leur somme:

```
 val nums = List( 1 , 2 , 3 , 4 )
 val sum = nums.foldLeft( 0 ){
   (acc, num) = > acc + num
 
```

En action dans le REPL:

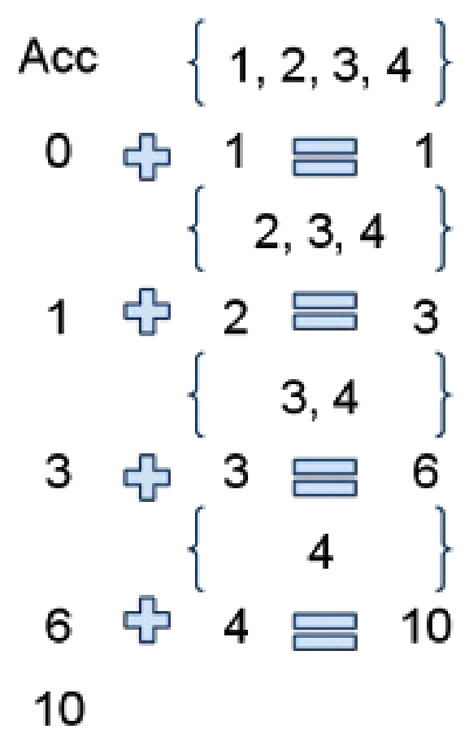
```
scala> val nums = List( 1 , 2 , 3 , 4 )
nums : List[Int] = List( 1 , 2 , 3 , 4 )

scala> val sum = nums.foldLeft( 0 ){
 (acc, num) = > acc + num
 }
sum : Int = 10
```

Le déroulé de l'opération foldLeft est le suivant :

```
(((( 0 + 1 ) + 2 ) + 3 ) + 4 )
(((( 1 ) + 2 ) + 3 ) + 4 )
((( 3 ) + 3 ) + 4 )
(( 6 ) + 4 )
( 10 )
```

En image cela donne:



foldRight

foldRight est assez similaire à *foldLeft* mais il parcourt la liste en partant de la droite de la liste. Ainsi c'est le dernier élément de la liste qui est d'abord traité. Voyons l'implémentation de la somme des éléments d'une liste avec *foldRight*:

```
scala> val nums = List( 1 , 2 , 3 , 4 )
nums : List[Int] = List( 1 , 2 , 3 , 4 )
scala> val sum = nums.foldRight( 0 ) {(num, acc) = >
  num + acc
}
sum : Int = 10
```

Le déroulé de l'opération foldLeft est le suivant :

```
( 1 + ( 2 + ( 3 + ( 4 + 0 ))))
( 1 + ( 2 + ( 3 + ( 4 ))))
( 1 + ( 2 + ( 7 )))
( 1 + ( 9 ))
( 10 )
```

Conclusion

Prenez le temps de digérer ce contenu, si jamais l'envie me prend dans l'avenir je reviendrais sur une comparaison plus avancée entre ces deux méthodes.

Tw eeter

Tags: fold foldLeft foldRight fonctionnel liste scala

← Impératif vs. fonctionnel ou le comment vs. le quoi

The Arolla « ancient world map » of software development →

1 comment for "Listes Scala: méthodes foldLeft et foldRight"

Eric Le Goff

29 avril 2015 at 16 h 14 min

Le deuxieme deroule s'intitule :

« Le déroulé de l'opération foldLeft est le suivant : »

il faudrait plutot
« Le déroulé de l'opération foldRight est le suivant : »

Laisser un commentaire

Votre adresse de messagerie ne sera pas publiée. Les champs obligatoires sont indiqués avec *

Nom *

Adresse de contact *

Site web

hzL0bwDmCjin *** Copy This Password ***

*** Type Or Paste Password Here ***

Commentaire

Laisser un commentaire

Arolla en bref

Tweets Suivre

brunoboucard @brunoboucard 14h
Tomorrow night we'll code in synchronized mode in different languages. Thanks guys @parisjug @tpierrain @dlresende pic.twitter.com/7nU1ea7irB
Retweeté par Arolla

Étendre

Arolla @ArollaFr 22h
Craft your Skills par @CoulasFedou, c'est jeudi chez Arolla! meetu.ps/2TX63w #CodingDojos #SoftwareCraftsmanship #FooBarQix
Afficher le Résumé

Benjamin Reitzammer 7 Févr

Tweeter à @ArollaFr

Manifeste d'Arolla

Suivre @ArollaFr

Inscrivez-vous à notre newsletter!

Recherche…

Rechercher

Principaux contributeurs

Cyrille Martraire
(30 articles)
Pierre Irrmann
(24 articles)
Nouhoum Traore
(13 articles)
Mathieu Laurent
(12 articles)
Yakhya Dabo
(11 articles)

Catégories

- Actu (53)
- Agilité (18)
- Bonnes pratiques de dév (52)
- Evénements (45)
- Fonctionnel (25)
- Outils (25)
- Programmation (102)
- Revues de presse (14)
- Traduction (2)

Étiquettes

agile architecture AST bdd C# clean code cqrs craftsmanship DDD design devops devoxxFR domain développement editeur enum F# fonctionnel GIT html5 immutable java Java 8 java8 javascript jvm lambda LINQ Linux maven mongodb MVC nodeJS outils polymorphism programming quality scala software software craftsmanship TDD TechDays testing visualization web

Archives

Sélectionner un mois

Liens

- Arolla.fr, le site web d'Arolla
- Le blog de Cyrille Martraire, directeur technique d'Arolla
- Le Tumblr d'Arolla