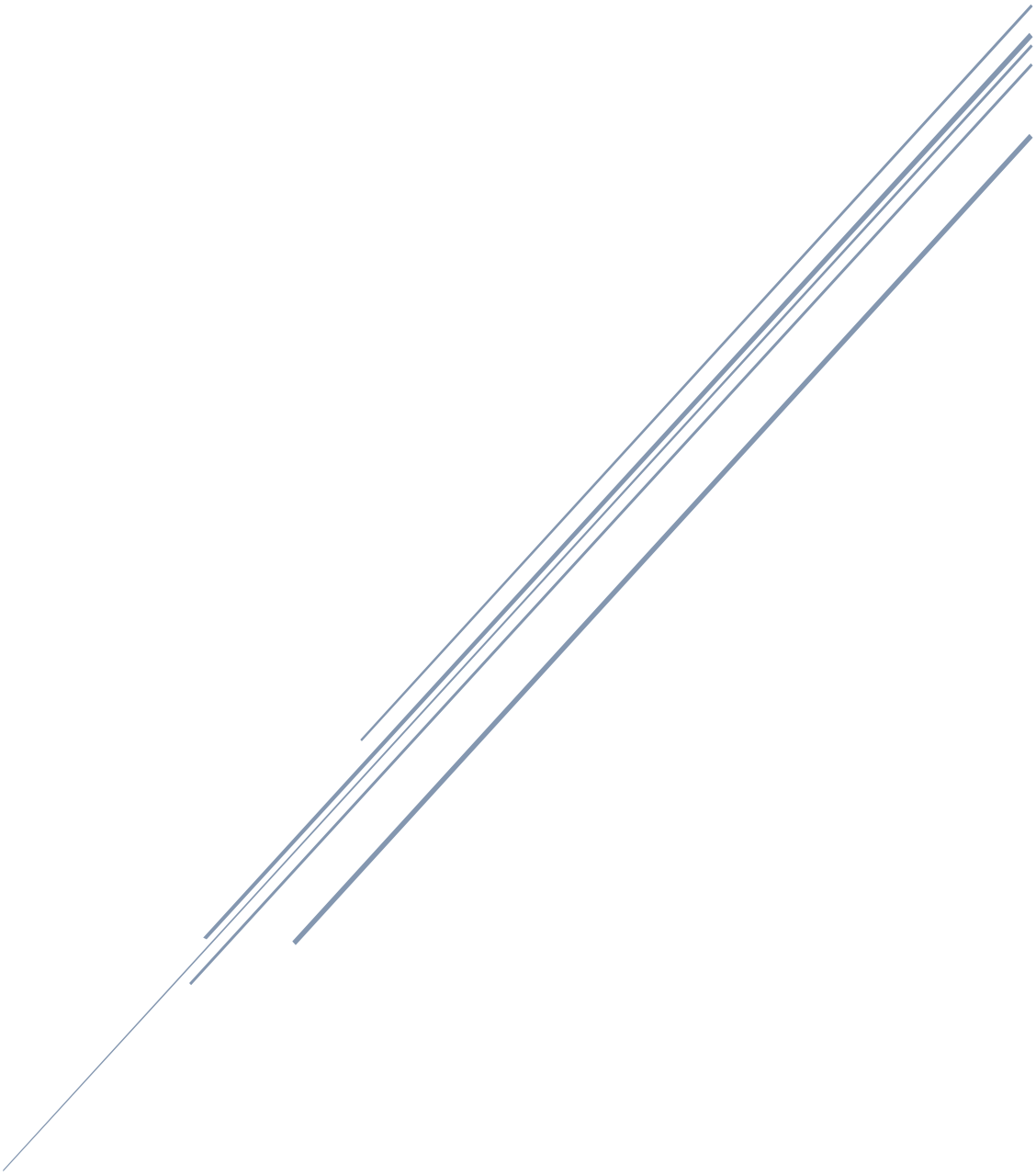
ANTLR COMPACT

Le vital pour comprendre ANTLR d'après la documentation de ANTLR



Vivian LOSCIALE

Table des matières

Disclaimer

La documentation de ANTLR4 est dépréciée, et certaines méthodes évoquées dans la documentation officielles ne sont tout simplement pas disponibles. Par conséquent, voici une explication de « Comment marche ANTLR ? », en essayant d'avoir expliqué au maximum ce que j'ai pu découvrir

Lexique de la grammaire

Commentaires :

Marche de la même façon qu'en JAVA, on peut donc utiliser « // », « /* */ » et « /** */ »

Identificateurs :

Les noms de ces mots clés utilisé pour le lexical commencent toujours par une majuscule « STARTER », bien qu'il faille que ce soit uniquement la première lettre en majuscule, je vous recommande de le mettre entièrement en majuscule.

Les noms des règles utilisés pour le parser doivent commencer par une minuscule « statement », bien qu'il faille que ce soit uniquement la première lettre en minuscule, je vous recommande de le mettre entièrement en minuscule.

String :

Pour écrire des strings dans le langage, utilisez le '' (simple quote), par exemple 'Hello', même si ce n'est pas un charactère, ANTLR ne fait pas la différence entre le string et le char, donc utilisez le ''.

Actions :

Sont les actions que doit effectuer le compilateur lorsqu'il reconnait une règle, un exemple sera donné par la suite.
Pour spécifier ces actions, il faut utiliser les { }.

Pour signifier la fin de ligne, utiliser EOF. (Comme ça, pas la peine de mettre des quote)

Structure de la grammaire

Une grammaire est définie par un ensemble de règle, mais suivant une construction précise.
Votre première ligne doit forcément être :

grammar NAME ;

où NAME est le nom de votre langage, à noter que celui-ci doit être le nom de votre fichier .g4.

Ensuite, il vous faut rentrer vos règles lexicales et syntaxiques (parser)

Les règles syntaxiques (parser)

Consiste en un ensemble de règle syntaxique ou d'une grammaire combinée. Pour rappel, les règles syntaxiques doivent commencer par une minuscule. Voici un exemple :

```
bloc :
    bloc_si bloc_alors ';'

bloc_si :
    'si'
    ;

bloc_alors :
    'alors'
    |
    ;

```

Ainsi, si l'on regarde la règle bloc, on voit qu'elle est constituée de deux règles, bloc_si et bloc_alors suivi d'un ';'. De même pour les deux autres blocs, il y a des ''.
SI l'on réfléchit à ce que peut produire cette règle bloc, elle peut produire :

si alors ;

si ;

Voici les phrases qui seront acceptés par cette règle. En effet, le bloc alors autorise plusieurs formulations, on peut soit écrire 'alors' ou bien ne rien mettre.

Les règles lexicales

Permet de généraliser une ou plusieurs expressions sous le même nom. Un exemple sera sans doute plus explicite. Pour rappel, les règles lexicales commencent toujours par une majuscule.

Imaginons que l'on veut faire une règle de calcul, on aurait donc le suivant :

```
INT : [0-9]+ ;
Calcul :
    INT '+' INT
    | INT '-' INT
    ;

```

Comme vous pouvez le voir, cela permet de faire une abstraction et de manipuler une entité qui correspond à ce que vous avez écrit. De cette manière, on peut écrire n'importe quelle soustraction et addition avec des nombres entier.

Cet exemple est assez simple, mais vous voyez ce qu'on peut faire séparément.

Dans le cas des commandes lexicales, il est intéressant de voir que l'on peut aussi spécifier certaines actions, comme le fait de ne pas les prendre en compte. Exemple :

```
WS : [ \t]+ -> skip ; // on s'en fout des espace
```

Et vous pouvez donc combiner les 2 pour avoir une grammaire importante !

```
BONJOUR : 'hola'
    | 'bonjour'
    | ' Buongiorno'
    ;

NAME : [a-z]+
    ;

PONCTUATION : ' !'
    | '…'
    ;

phrase : salutations PONCTUATION adieu
    ;

salutations : BONJOUR
    | BONJOUR NAME
    ;

adieu : 'rip'
    ;
```

Je pense que vous l'avez remarqué, mais la structure d'une règle est donc :

NOM : règle1

| règle 2

| …

;

Le saut à la ligne n'est pas obligatoire mais cela permet d'avoir une meilleure visibilité et de spécifier mieux les actions que doivent faire les règles. En parlant d'action…

Les actions dans la grammaire

Si vous voulez faire remonter une information globale, il vous faudra utiliser le '@members'

@members {
}

Vous spécifiez donc le/les variables que vous voulez avoir ainsi, vous pouvez les utiliser dans le champ des actions d'une règle spécifique. Exemple :

```
@members {
int compteur = 0;
}

list
@after {System.out.println(compteur+" ints vu dans la chaine");}
: (INT {count++;})*
;

INT : [0-9]+ ;
WS : [ \r\t\n]+ -> skip ;

```

Ici, lors que l'on voit la chaine '30 254 2 54', on va donc avoir notre compteur qui va donner la valeur 4. Le @after permet de dire que l'action qui suit (ici le print) doit se faire lorsqu'on a fini d'analyser cette règle.
Cette variable sera partagée avec toutes les règles, mais il est possible de remonter uniquement la valeur d'une règle, et c'est ce que l'on va voir.

Définition règle d'attribut

Il est possible de renvoyer directement une valeur d'une règle si cela est utile. Pour cela, utilisez [ ] :

```
rulename[args] returns [retvals] locals [localvars] : ... ;
```

Un exemple serait :

```
add[int x] returns [int result] : '+=' INT {$result = $x + $INT.int;} ;
```

Dans l'exemple donné plus haut, nous n'avons pas de variable locale, donc la partie local n'est pas nécessaire.
Concernant le INT.int, ANTLR permet d'accéder aux attributs des règles lexicales, donc ici, le INT correspond au [0-9]+, donc INT.int permettra de récupérer la valeur en type int, on pourrait aussi le demander en string, tout simplement en écrivant INT.text.

Les arbres

ANTLR construit par défaut un arbre syntaxique, un ParseTree.

Avec la commande mvn antlr4:antlr4, ANTLR génère une interface Listener qui étend de ParseTreeListener

Il génère aussi une classe NAMEbaseListener.java, où vous pouvez spécifier l'appel à certaines méthodes lorsqu'il rentre dans une règle ou bien lorsqu'il sort de celui-ci.

En gros tout se fait via l'override de ces fonctions de ce listener.

https://stackoverflow.com/questions/15050137/once-grammar-is-complete-whats-the-best-way-to-walk-an-antlr-v4-tree