eBook Gratuit

APPRENEZ ANTLR

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#antlr

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: antlr

It is an unofficial and free ANTLR ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ANTLR.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec ANTLR

Remarques

ANTLR (ANother Tool for Language Recognition) est un puissant générateur d'analyseurs pour la lecture, le traitement, l'exécution ou la traduction de textes structurés ou de fichiers binaires. Il est largement utilisé pour créer des langages, des outils et des frameworks. À partir d'une grammaire, ANTLR génère un analyseur qui peut créer et parcourir des arbres d'analyse.

- Site officiel antlr (pointe toujours sur la dernière version)

Antlr Versions
Antlr est séparé en deux grandes parties, la grammaire (fichiers de grammaire) et les fichiers de code générés, qui dérivent de la grammaire basée sur le langage cible. Les versions antlr sont au format de V1.V2.V3:

- V1: Changer dans V1 signifie que la nouvelle syntaxe des fonctionnalités a été introduite dans les fichiers de grammaire
- V2: Change in V2 signifie que de nouvelles fonctionnalités ou corrections majeures ont été introduites dans les fichiers générés (par exemple, l'ajout de nouvelles fonctions)
- V3: correspond à des corrections de bogues ou à des améliorations mineures

Bibliothèques d'exécution et cibles de génération de code
L'outil Antlr est écrit en Java, mais il est capable de générer des analyseurs et des lexers dans différents langages. Pour exécuter l'analyseur et le lexer, vous devez également disposer de la bibliothèque d'exécution d'antlr avec le code analyseur et lexer. Le langage cible pris en charge (et les bibliothèques d'exécution) sont les suivants:

- Java
- C #
- Python (2 et 3)
- JavaScript

Versions

Version	Date de sortie
2.0	1997-05-01
3.0	2011-01-19
4.0	2013-01-21
4.1	2013-07-01

Version	Date de sortie
4.2	2014-02-05
4.2.1	2014-03-25
4.2.2	2014-04-07
4.3	2014-06-19
4.4	2014-07-16
4.5	2015-01-23
4.5.1	2016-07-16
4.5.2	2016-01-30
4.5.3	2016-03-31
4.6	2016-12-15
4.7	2017-03-30

Examples

Bonjour le monde

Un simple grammaire mondiale peut être trouvé ici :

```
// define a grammar called Hello
grammar Hello;
r   : 'hello' ID;
ID  : [a-z]+ ;
WS  : [ \t\r\n]+ -> skip ;
```

Pour générer cet exemple .g4, vous pouvez exécuter la commande suivante à partir du terminal / de la ligne de commande de votre système d'exploitation:

```
Java -jar antlr-4.5.3-complete.jar Hello.g4

//OR if you have setup an alias or use the recommended batch file

antlr4 Hello.g4
```

La construction de cet exemple doit aboutir à la sortie suivante dans le répertoire de fichiers Hello.g4:

1. Hello.tokens
2. HelloBaseListener.java

3. HelloLexer.java
4. HelloLexer.tokens
5. HelloListener.java
6. Bonjour.java

Lorsque vous utilisez ces fichiers dans votre propre projet, veillez à inclure le fichier jar ANTLR. Pour compiler tous ces fichiers à l'aide de Java, dans le même répertoire d'exploitation ou par chemin, exécutez la commande suivante:

```
javac *.java
```

Lire Démarrer avec ANTLR en ligne: https://riptutorial.com/fr/antlr/topic/4453/demarrer-avec-antlr

Chapitre 2: ANTLR Cibles / Runtimes linguistiques

Examples

Support linguistique

ANTLR est capable de générer des analyseurs pour un certain nombre de langages de programmation:

1. C # Cible
2. Cible Python
3. Cible JavaScript
4. Cible Java

Par défaut, ANTLR générera un analyseur à partir de la ligne de commande dans le langage de programmation Java:

```
Java -jar antlr-4.5.3-complete.jar yourGrammar.g4 //Will output a
    java parser
```

Pour changer la langue cible, vous pouvez exécuter la commande suivante à partir du terminal OS / ligne de commande:

```
    antlr4 -Dlanguage=Python3 yourGrammar.g4
//with alias
    java -jar antlr-4.5.3-complete.jar -Dlanguage=Python3 yourGrammar.g4
//without alias
```

Plutôt que d'utiliser le paramètre '-Dlanguage' sur la ligne de commande / terminal chaque fois que vous créez l'analyseur souhaité pour une langue spécifique, vous pouvez sélectionner la cible dans votre fichier de grammaire .g4 en incluant la cible dans la section globale:

```
options {
    language  = "CSharp";
}
//or
options {
    language="Python";
}
```

Pour utiliser la sortie d'analyse analysée, assurez-vous que vous disposez du moteur d'exécution ANTLR pour la langue spécifiée:

1. CSharp runtime
2. Python 2 runtime
3. python 3 runtime

Instructions complètes et informations sur les bibliothèques d'exécution ANTLR

Configuration de l'analyseur Python

Après avoir exécuté votre fichier .g4 avec ANTLR.jar, vous devez avoir généré un certain nombre de fichiers tels que:

```
1.yourGrammarNameListener.py
2.yourGrammarNameParser.py
3.yourGrammarName.tokens
...
```

Pour les utiliser dans un projet python, incluez le runtime Python dans votre espace de travail afin que toute application que vous développez puisse accéder à la bibliothèque ANTLR. Cela peut être fait en extrayant le runtime dans votre dossier de projet actuel ou en l'important dans votre IDE dans les dépendances de votre projet.

```
#main.py
import yourGrammarNameParser
import sys

#main method and entry point of application

def main(argv):
    """Main method calling a single debugger for an input script"""
    parser = yourGrammarNameParser
    parser.parse(argv)

if __name__ == '__main__':
    main(sys.argv)
```

Cette configuration inclut votre analyseur et accepte l'entrée de la ligne de commande pour permettre le traitement d'un fichier passé en paramètre.

```
#yourGrammarNameParser.py
from yourGrammarNameLexer import yourGrammarNameLexer
from yourGrammarNameListener import yourGrammarNameListener
from yourGrammarNameParser import yourGrammarNameParser
from antlr4 import *
import sys

class yourGrammarNameParser(object):
    """
    Debugger class - accepts a single input script and processes
    all subsequent requirements
    """
def __init__(self): # this method creates the class object.
    pass


#function used to parse an input file
def parse(argv):
    if len(sys.argv) > 1:
        input = FileStream(argv[1]) #read the first argument as a filestream
        lexer = yourGrammarNameLexer(input) #call your lexer
```

```
        stream = CommonTokenStream(lexer)
        parser = yourGrammarNameParser(stream)
        tree = parser.program() #start from the parser rule, however should be changed to your
entry rule for your specific grammar.
        printer = yourGrammarNameListener(tree,input)
        walker = ParseTreeWalker()
        walker.walk(printer, tree)
    else:
        print('Error : Expected a valid file')
```

Ces fichiers, associés à l'environnement d'exécution ANTLR et à vos fichiers générés à partir de votre fichier de grammaire, accepteront un seul nom de fichier comme argument et liront et analyseront vos règles de grammaire.

Pour étendre les fonctionnalités de base, vous devez également développer l'écouteur par défaut pour gérer les événements pertinents pour les jetons rencontrés lors de l'exécution.

Lire ANTLR Cibles / Runtimes linguistiques en ligne: https://riptutorial.com/fr/antlr/topic/3414/antlr-cibles---runtimes-linguistiques

Chapitre 3: Introduction à ANTLR v3

Examples

Installation et configuration

Comment installer ANTLR dans Eclipse

(Dernier test sur Indigo et ANTLR IDE 2.1.2)

1. Installez Eclipse.
2. Téléchargez le fichier binaire complet ANTLR qui inclut ANTLR v2. Extraire dans un répertoire temporaire. Copiez le dossier antlr-nn vers un emplacement permanent approprié, par exemple le même dossier dans lequel Eclipse est installé.
3. Ajoutez le site de mise à jour IDE ANTLR à Eclipse.
    - Dans Eclipse, cliquez sur Aide et sélectionnez Installer un nouveau logiciel.
    - Cliquez sur le bouton Ajouter….
    - Dans la fenêtre Ajouter un référentiel, pour Emplacement, tapez http://antlrv3ide.sourceforge.net/updates et tapez quelque chose comme ANTLR IDE pour le nom et cliquez sur OK pour revenir à la fenêtre Logiciels disponibles.
    - Cochez la case pour ANTLR IDE vn.nn et cliquez dessus jusqu'à ce qu'il soit installé. Eclipse va probablement redémarrer.
4. Configurez l'IDE ANTLR.
    - Dans la fenêtre principale d'Eclipse, cliquez sur Window puis sur Preferences.
    - Dans le volet gauche, développez ANTLR et sélectionnez Générateur.
    - Dans le volet de droite, cliquez sur le bouton Ajouter….
    - Dans la fenêtre Ajouter un package ANTLR, cliquez sur Répertoire… et accédez à l'emplacement du dossier antlr-nn, puis cliquez sur OK.
    - Cliquez sur OK pour fermer la fenêtre Ajouter un package ANTLR.
    - Sélectionnez Générateur de code dans le volet gauche et cliquez sur Dossier relatif au projet dans le volet droit. Tapez un nom de dossier. Exemples: antlr-java ou antlr généré.
    - Sélectionnez d'autres paramètres de configuration, mais NE vérifiez PAS –nfa ou –dfa dans la sous-fenêtre Général de la fenêtre Building. Si cette option est cochée, cela entraînera des erreurs ANTLR empêchant la génération de fichiers Java dans le dossier de sortie.
    - Cliquez sur OK pour fermer la fenêtre Préférences.
5. Créez un nouveau projet Java et activez le support ANTLR.
    - Dans la fenêtre principale d'Eclipse, accédez à Fichier, Nouveau, Projet Java. Cliquez sur Suivant, tapez un nom de projet et cliquez sur Terminer.
    - Pour activer le support ANTLR pour le projet, dans la fenêtre Explorateur de packages (volet gauche), cliquez avec le bouton droit sur le projet que vous venez de créer et sélectionnez Configurer, convertir en projet ANTLR.
    - Ajoutez le fichier JAR complet ANTLR au projet: cliquez avec le bouton droit sur le

projet et sélectionnez Propriétés, Chemin de génération Java, cliquez sur Ajouter des fichiers JAR externes, accédez au fichier JAR ANTLR, sélectionnez-le et cliquez sur OK. Cliquez sur OK pour fermer la fenêtre Propriétés du projet.

6. Créez une grammaire ANTLR.
    - Créez une nouvelle grammaire ANTLR: cliquez avec le bouton droit sur le dossier src du projet, puis cliquez sur Fichier, Nouveau, Autre, développez ANTLR et sélectionnez Grammaire combinée. Cliquez sur Suivant, tapez nom de la grammaire, sélectionnez une option de langue et cliquez sur Terminer.
    - Un fichier «.g» est créé avec les options sélectionnées et une règle vide. Ajoutez les options language = Java, @header, @lexer :: header et @members en haut (voir exemple). L'achèvement automatique est le moyen le plus simple de les ajouter (appuyez sur CTRL-espace pour afficher la liste de saisie automatique).
7. Enregistrez la grammaire.
    - Une fois enregistré, un dossier contenant le code Java généré pour la grammaire doit apparaître dans l'explorateur de projet. Si ce n'est pas le cas, assurez-vous que les options –nfa ou –dfa ne sont pas cochées dans les préférences ANTLR sous Général dans la fenêtre du bâtiment (étape 4g). [Confirmez si nécessaire: vérifiez que la variable d'environnement CLASSPATH pointe vers Java7 correspondant à votre installation Eclipse (32 ou 64 bits) et que la variable d'environnement Windows Path était dotée du SDK Java7.]
    - Pour éviter les erreurs Java «ne peuvent pas être résolues en un type», cliquez avec le bouton droit sur le dossier contenant le code Java généré, puis créez un chemin, utilisez-le comme dossier source.

ÉCHANTILLON COMBINÉ GRAMMAIRE

```
grammar test; //must match filename.g

options {
    language = Java;
}

@header { //parser
    package pkgName; //optional
    import java.<whatever you need>.*;
}

@members { //parser
    // java code here
}

@lexer::header { //lexer
    package pkgName; //optional
    import java.<whatever you need>.*;
}

@lexer::members {
    // java code here
}
/*------------------------------------------------------------------
 * PARSER RULES (convention is all lowercase)
 *------------------------------------------------------------------*/
parserule: LEXRULE;
```

```
/*------------------------------------------------------------------
 * LEXER RULES (convention is all uppercase)
 *------------------------------------------------------------------*/
LEXRULE: 'a'..'z';
```

Lire Introduction à ANTLR v3 en ligne: https://riptutorial.com/fr/antlr/topic/6629/introduction-a-antlr-v3

Chapitre 4: Introduction à ANTLR v4

Remarques

ANTLR v4 est un outil puissant permettant de créer de nouveaux langages de programmation et de traiter / traduire du texte structuré ou des fichiers binaires. ANTLR utilise une grammaire que vous créez pour générer un analyseur capable de créer et de parcourir un arbre d'analyse syntaxique (ou arbre syntaxique abstrait, AST). L'analyseur se compose de fichiers de sortie dans une langue cible que vous spécifiez. ANTLR v4 prend en charge plusieurs cibles, notamment: Java, C #, JavaScript, Python2 et Python3. Le support de C ++ est en cours de développement. Pour travailler dans des IDE GUI, il existe des plug-ins pour Visual Studio, Intellij, NetBeans et Eclipse.

Pour des informations générales, visitez le site Web d'ANTLR . Pour prendre au sérieux ANTLR, consultez le livre hautement recommandé écrit par Terrence Parr (le gars qui a créé ANTLR) The Definitive ANTLR 4 Reference .

Informations importantes sur la version

- 4.5: 22/01/15 - Ajout d'une cible JavaScript et mise à niveau de la cible C #. 4.5 Notes de version
- 4.4: 16/07/14 - Ajout de Python2 et Python3 comme cibles. 4.4 Notes de version
- 4.3: 18/06/14 - Correction de bugs majeurs; préparé pour l'ajout de nouvelles cibles. 4.3 Notes de version
- 4.2: 02/04/14 - Amélioration de la syntaxe pour la sélection / correspondance des arbres d'analyse. 4.2 Notes de version
- 4.1: 30/06/13 - Amélioration des performances d'analyse; exporter des AST vers PNG. 4.1 Notes de publication
- 4.0: 21/01/13 - Version initiale.

Examples

Installation pour utilisation en ligne de commande

ANTLR est distribué sous forme de fichier Java Jar. Il peut être téléchargé ici . Comme ANTLR est compilé en tant que fichier jar, il faut ensuite que l'environnement d'exécution Java fonctionne, si vous ne le possédez pas. Il peut être téléchargé ici .

Une fois le fichier ANTLR JAR téléchargé, vous pouvez exécuter ANTLR à partir de la ligne de commande de la même manière que tout autre fichier JAR:

```
Java -jar antlr-4.5.3-complete.jar
```

(En supposant que vous travaillez dans le même répertoire que le fichier antlr-4.5.3-complete.jar).

Cela devrait produire quelque chose de similaire à ceci:

```
ANTLR Parser Generator  Version 4.5.3
 -o ___              specify output directory where all output is generated
 -lib ___            specify location of grammars, tokens files
 -atn                generate rule augmented transition network diagrams
 -encoding ___       specify grammar file encoding; e.g., euc-jp
 -message-format ___ specify output style for messages in antlr, gnu, vs2005
 -long-messages      show exception details when available for errors and warnings
 -listener           generate parse tree listener (default)
 -no-listener        don't generate parse tree listener
 -visitor            generate parse tree visitor
 -no-visitor         don't generate parse tree visitor (default)
 -package ___        specify a package/namespace for the generated code
 -depend             generate file dependencies
 -D<option>=value    set/override a grammar-level option
 -Werror             treat warnings as errors
 -XdbgST             launch StringTemplate visualizer on generated code
 -XdbgSTWait         wait for STViz to close before continuing
 -Xforce-atn         use the ATN simulator for all predictions
 -Xlog               dump lots of logging info to antlr-timestamp.log
```

Les autres actions recommandées pour la configuration incluent:

```
     1. Add antlr4-complete.jar to CLASSPATH, either: Permanently:
Using System Properties dialog > Environment variables > Create or append to CLASSPATH
variable Temporarily, at command line: SET CLASSPATH=.;C:\Javalib\antlr4-
complete.jar;%CLASSPATH%
     3.Create batch commands for ANTLR Tool, TestRig in dir in PATH
         antlr4.bat: java org.antlr.v4.Tool %*
         grun.bat:   java org.antlr.v4.gui.TestRig %*
```

Après l'installation, vous pouvez créer une application en utilisant votre fichier de grammaire .g4:

```
Java -jar antlr-4.5.3-complete.jar yourGrammar.g4
```

Vous pouvez également créer une application dans d'autres langages avec le paramètre -Dlanguage. Par exemple, pour générer des fichiers C #, vous feriez quelque chose comme ceci:

```
java -jar antlr-4.5.3-complete.jar yourGrammar.g4 -Dlanguage=CSharp
```

Voir ici pour la liste complète des grammaires pré-établies pour les langages de programmation courants.

Installation à l'aide d'outils Build Automation

Téléchargez la dernière version d'ANTLR et extrayez-la dans un dossier.

Vous pouvez également utiliser Maven, Gradle ou un autre outil de génération pour dépendre de son `org.antlr:antlr4-runtime` exécution (les classes utilisées par les grammaires): `org.antlr:antlr4-runtime` .

Pour générer automatiquement l'analyseur dans un projet Maven , utilisez le plugin Maven :

`org.antlr:antlr4` .

Installer dans Eclipse et construire Hello World

(Testé avec ANTLR 4.5.3, Eclipse Neon, ANTLR 4 IDE 0.3.5 et Java 1.8)

1. Téléchargez le dernier ANTLR . Assurez-vous d'obtenir le fichier binaire complet ANTLR Java. Enregistrez dans n'importe quel emplacement approprié, par exemple le dossier où sont stockées les autres bibliothèques Java. Peu importe où, rappelez-vous simplement l'emplacement.

2. Installez l'IDE ANTLR dans Eclipse.

    - Dans le menu Eclipse, cliquez sur Aide et sélectionnez Marché Eclipse.
    - Dans la zone Rechercher: tapez antlr et cliquez sur OK.
    - Cliquez sur Installer pour ANTLR 4 IDE.
    - Cliquez sur Terminer dans la fenêtre Confirmation des entités sélectionnées.
    - Si une fenêtre d'avertissement de sécurité apparaît, cliquez sur OK.
    - Redémarrez Eclipse.

3. Contournez l'erreur «Impossible de créer un injecteur…».

    - Lors de l'accès aux préférences ANTLR 4 dans Eclipse ou lorsque la variable d'environnement HOME n'est pas définie, l'erreur suivante se produit: Échec de la création de l'injecteur pour com.github.jknack.antlr-4ide.Antlr4 pour com.github.jknack.antlr-4ide.Antlr4 .
    - Assurez-vous que la variable d'environnement HOME est définie. Sinon, configurez-le selon vos besoins.
    - Téléchargez Xtext 2.7.3 au même endroit que antlr-nnn-complete.jar.
    - Dans Eclipse, cliquez sur Aide et sélectionnez Installer un nouveau logiciel.
    - Cliquez sur Ajouter… pour accéder à la fenêtre Ajouter un référentiel.
    - Tapez un nom, xtext 2.7.3 par exemple, puis cliquez sur Archive…, accédez au fichier Xtext 2.7.3 et sélectionnez-le, puis cliquez sur OK.
    - Dans la fenêtre d'installation, cliquez sur le bouton Sélectionner tout, puis sur Suivant> deux fois, acceptez le contrat de licence. et cliquez sur Terminer.
    - Redémarrez Eclipse.

4. Dites à Eclipse / Java où est ANTLR.

    - Dans Eclipse, cliquez sur Fenêtre et sélectionnez Préférences.
    - Dans le volet gauche, développez Java et Build Path, puis sélectionnez Classpath Variables.
    - Dans le volet droit, cliquez sur Nouveau…, entrez un nom, puis cliquez sur Fichier… et accédez à votre emplacement antlr-nnn-complete.jar. Cliquez sur OK pour revenir à la fenêtre Variables du chemin de classes.
    - Cliquez sur OK pour quitter les Préférences.

5. (Facultatif) Configurez le répertoire des sources générées par ANTLR IDE.

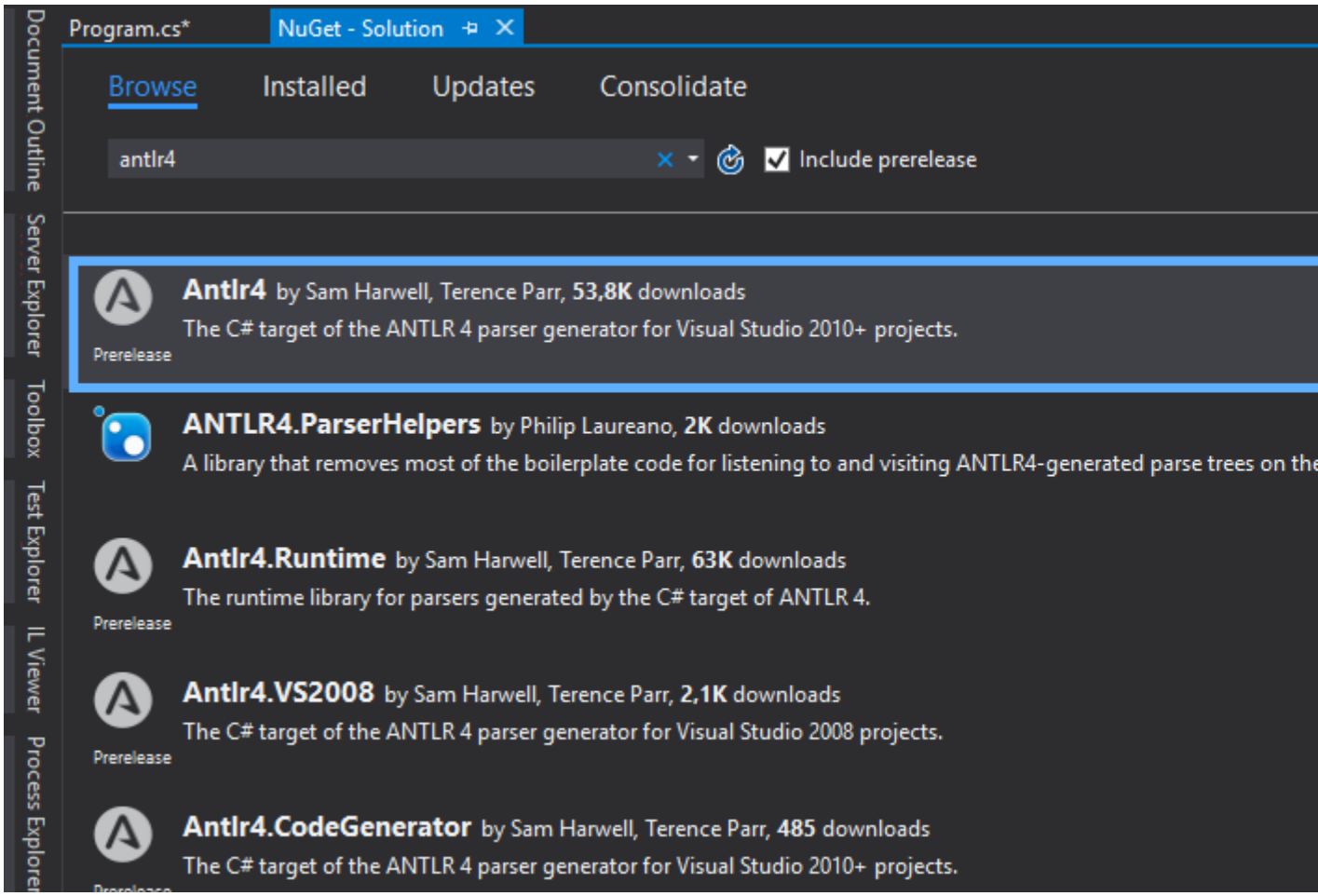
- Dans la fenêtre principale d'Eclipse, cliquez sur Window puis sur Preferences.
- Dans le volet gauche, développez ANTLR 4 et sélectionnez Outil.
- Sous Options, modifiez le répertoire si vous le souhaitez. Par exemple, java est ma langue cible, donc j'utilise ./antlr-java.
- Cliquez sur OK pour fermer la fenêtre Préférences.

6. Créez un projet ANTLR 4.

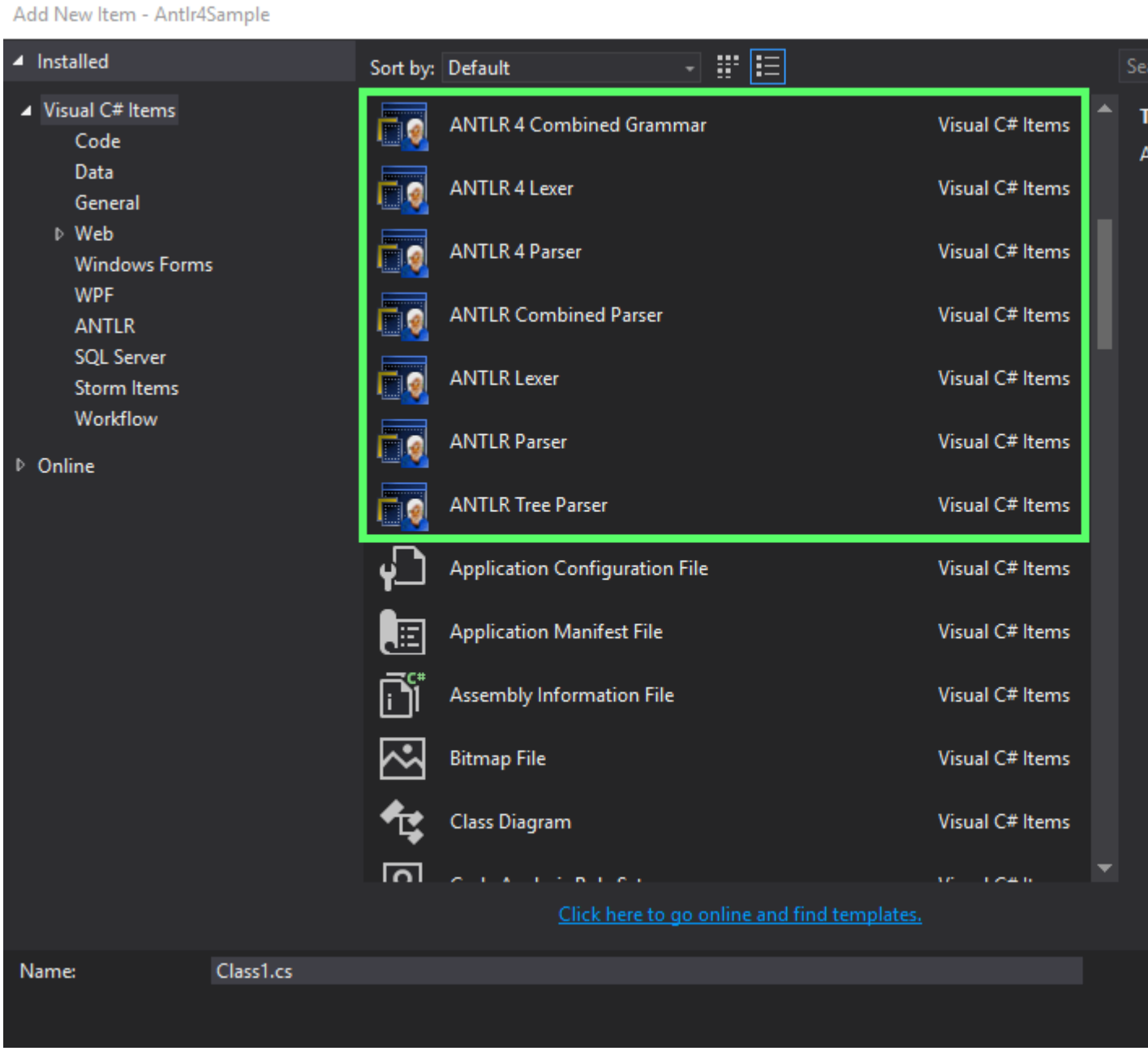
   - Dans la fenêtre principale d'Eclipse, accédez à Fichier, Nouveau, Projet.
   - Dans la fenêtre Nouveau projet, développez Général et sélectionnez Projet ANTLR 4.
   - Cliquez sur Suivant, tapez un nom de projet et cliquez sur Terminer.
   - Le nouveau projet par défaut contient un fichier Hello.g4 et générera automatiquement le programme standard "Hello World".
   - Dans l'explorateur de packages, développez le nouveau dossier de projet pour afficher le fichier g4 et un dossier nommé target (ou le nom que vous lui avez donné à l'étape 5) contenant les fichiers source cibles.

Installation d'ANTLR dans Visual Studio 2015 (à l'aide de Nuget)

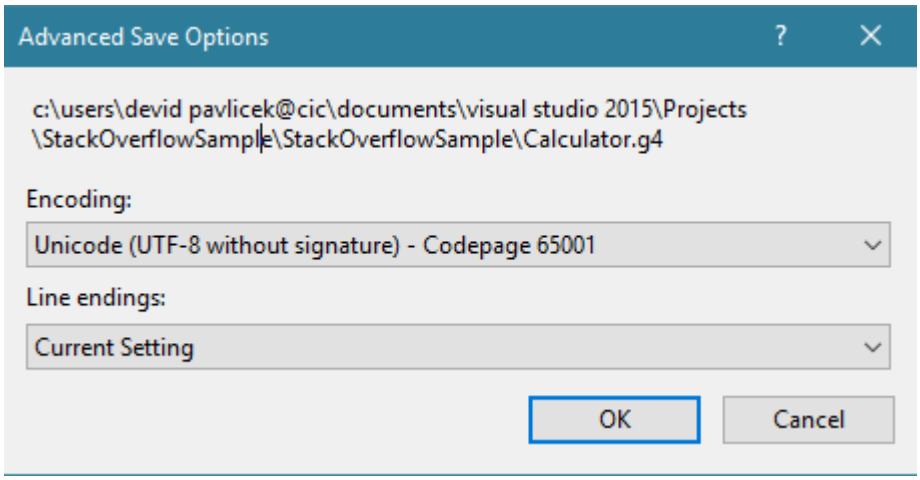
1. Ouvrez Visual Studio 2015, accédez à Outils → Extensions → En ligne et recherchez Antlr. Téléchargez l'extension ANTLR Language Support (créée par Sam Harwell) et redémarrez Visual Studio.
2. Créez un nouveau projet d'application de console. Faites un clic droit sur la solution → Gérer les paquets Nuget pour la solution → Parcourir (Tab) et recherchez Antlr4 et installez-le.



3. Ajoutez un nouvel élément à votre projet en cliquant dessus avec le bouton droit de la souris. Et recherchez les modèles ANTLR4.

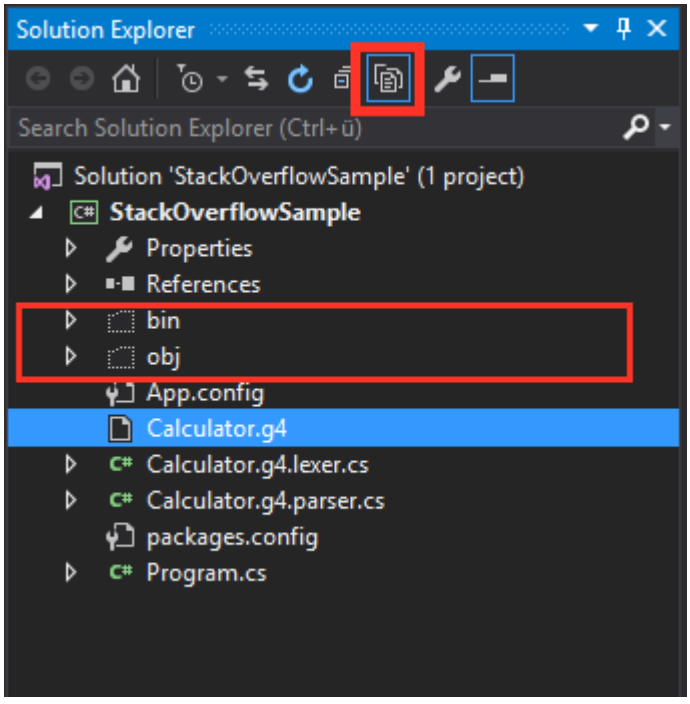


4. À partir de votre fichier ANTLR (se terminant par .g4), accédez à Fichier → Options de sauvegarde avancées et recherchez Unicode (UTF-8 **sans signature**) - Codepage 65001 et cliquez sur OK. C'est tout.

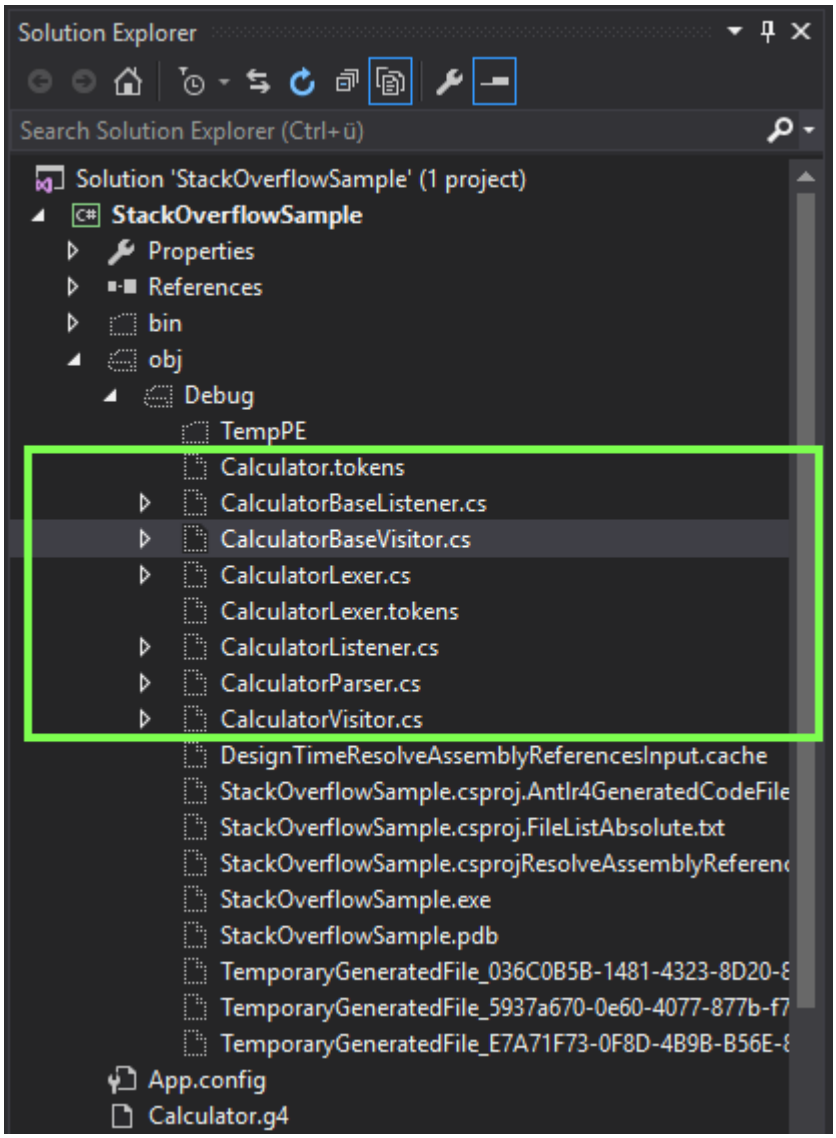


Tester si tout fonctionne

- Créez un élément de grammaire combinée ANTLR 4 et nommez-le Calculator.g4
- Copiez et collez le code source de la calculatrice de ce projet Github ici: Calculatrice de Tom Everett
- Changer la calculatrice grammaticale en calculatrice grammaticale
- Sur l'Explorateur de solutions → Cliquez sur Afficher tous les fichiers.



- Enregistrer et exécuter (Démarrer) le projet
- Dans l'Explorateur de solutions sous le dossier obj, vous devriez voir les classes cs générées comme le visiteur et le récepteur. Si c'est le cas, vous avez réussi. Vous pouvez maintenant commencer à travailler avec ANTLR dans Visual Studio 2015.



Lire Introduction à ANTLR v4 en ligne: https://riptutorial.com/fr/antlr/topic/2856/introduction-a-antlr-v4

Chapitre 5: Les auditeurs

Examples

Événements d'écoute à l'aide d'étiquettes

L'étiquetage des alternatives dans une règle commençant par l'opérateur `#` indique à ANTLR de générer des méthodes d'écoute pour chaque étiquette correspondant à l'alternative.

En spécifiant une étiquette pour chaque alternative dans la règle suivante:

```
// Rule
type : int     #typeInt
     | short   #typeShort
     | long    #typeLong
     | string  #typeString
     ;

// Tokens
int : 'int' ;
short : 'short' ;
long : 'long' ;
string : 'string' ;
```

`ParseTreeListener` les méthodes suivantes dans l'interface générée qui étend `ParseTreeListener` :

```
public void enterTypeInt(TypeShortContext ctx);
public void enterTypeShort(TypeIntContext ctx);
public void enterTypeLong(TypeLongContext ctx);
public void enterTypeString(TypeStringContext ctx);
```

Lire Les auditeurs en ligne: https://riptutorial.com/fr/antlr/topic/6717/les-auditeurs

Chapitre 6: Règles Lexer en v4

Examples

Règles simples

Les règles Lexer définissent les types de jeton. Leur nom doit commencer par une lettre majuscule pour les distinguer des règles d'analyseur.

```
INTEGER: [0-9]+;
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;

OPEN_PAREN: '(';
CLOSE_PAREN: ')';
```

Syntaxe de base:

Syntaxe	Sens
`A`	Faire correspondre la règle ou le fragment lexer nommé `A`
`AB`	Match `A` suivi de `B`
`(A|B)`	Match soit `A` ou `B`
`'text'`	Match littéral *"texte"*
`A?`	Match `A` zéro ou une fois
`A*`	Match `A` zéro ou plusieurs fois
`A+`	Match `A` une ou plusieurs fois
`[A-Z0-9]`	Faire correspondre un caractère dans les plages définies (dans cet exemple, entre AZ et 0-9)
`'a'..'z'`	Syntaxe alternative pour une plage de caractères
`~[AZ]`	Négation d'une plage - correspond à n'importe quel caractère unique qui *n'est pas* dans la plage
`.`	Correspond à n'importe quel caractère

Fragments

Les fragments sont des parties réutilisables des règles de lexer qui ne peuvent pas correspondre seules. Elles doivent être référencées à partir d'une règle de lexer.

```
INTEGER: DIGIT+
       | '0' [Xx] HEX_DIGIT+
       ;

fragment DIGIT: [0-9];
fragment HEX_DIGIT: [0-9A-Fa-f];
```

Règles implicites de lexer

Lorsque des jetons comme `'{'` sont utilisés dans une règle d' *analyse* , une règle lexer implicite sera créée à moins qu'une règle explicite n'existe.

En d'autres termes, si vous avez une règle de lexer:

```
OPEN_BRACE: '{';
```

Ensuite, ces deux règles d'analyse sont équivalentes:

```
parserRule: '{';
parserRule: OPEN_BRACE;
```

Mais si la règle `OPEN_BRACE` OPEN_BRACE n'est *pas* définie, une règle anonyme implicite sera créée. Dans ce cas, la règle implicite sera définie *comme si* elle était définie *avant* les autres règles: elle aura une priorité plus élevée que les autres règles.

Règles de priorité

Plusieurs règles de lexer peuvent correspondre au même texte d'entrée. Dans ce cas, le type de jeton sera choisi comme suit:

- Tout d'abord, sélectionnez la règle lexer qui correspond à l'entrée la *plus longue*
- Si le texte correspond à un jeton défini implicitement (comme `'{'`), utilisez la règle implicite
- Si plusieurs règles de lexer correspondent à la même longueur d'entrée, choisissez la *première en* fonction de l'ordre de définition.

La grammaire combinée suivante:

```
grammar LexerPriorityRulesExample;

// Parser rules

randomParserRule: 'foo'; // Implicitly declared token type

// Lexer rules

BAR: 'bar';
IDENTIFIER: [A-Za-z]+;
BAZ: 'baz';

WS: [ \t\r\n]+ -> skip;
```

Compte tenu de l'entrée suivante:

```
aaa foo bar baz barz
```

Produira la séquence de jetons suivante à partir du lexer:

```
IDENTIFIER 'foo' BAR IDENTIFIER IDENTIFIER
```

- `aaa` est de type `IDENTIFIER`

  Seule la règle `IDENTIFIER` peut correspondre à ce jeton, il n'y a pas d'ambiguïté.

- `foo` est de type `'foo'`

  La règle d'analyse syntaxique `randomParserRule` introduit le type de jeton implicite `'foo'` , qui est prioritaire sur la règle `IDENTIFIER` .

- `bar` est du type `BAR`

  Ce texte correspond à la règle `BAR` définie *avant* la règle `IDENTIFIER` et a donc la priorité.

- `baz` est de type `IDENTIFIER`

  Ce texte correspond à la règle `BAZ` , mais il correspond également à la règle `IDENTIFIER` . Ce dernier est choisi tel qu'il est défini *avant* `BAR` .

  Compte tenu de la grammaire, `BAZ` *ne* sera *jamais* en mesure de correspondre, car la règle `IDENTIFIER` couvre déjà tout ce que `BAZ` peut égaler.

- `barz` est du type `IDENTIFIER`

  La règle `BAR` peut correspondre aux 3 premiers caractères de cette chaîne (`bar`), mais la règle `IDENTIFIER` correspond à 4 caractères. Comme `IDENTIFIER` correspond à une sous-chaîne plus longue, il est choisi sur `BAR` .

En règle générale, les règles spécifiques doivent être définies *avant* des règles plus génériques. Si une règle ne peut correspondre qu'à une entrée déjà couverte par une règle précédemment définie, elle *ne* sera *jamais* utilisée.

Les règles implicitement définies telles que `'foo'` agissent comme si elles étaient définies *avant* toutes les autres règles lexer.

Commandes Lexer

Une règle lexer peut avoir des *commandes* associées:

```
WHITESPACE: [ \r\n] -> skip;
```

Les commandes sont définies après un `->` à la fin de la règle.

- `skip` : ignore le texte correspondant, aucun jeton ne sera émis
- `channel(n)` : émet le jeton sur un canal différent
- `type(n)` : change le type de jeton émis
- `mode(n)` , `pushMode(n)` , `popMode` , `more` : contrôle les modes lexer

Actions et prédicats sémantiques

Une action lexer est un bloc de code arbitraire dans le langage cible entouré de `{ ... }` , qui est exécuté lors de la correspondance:

```
IDENTIFIER: [A-Z]+ { log("matched rule"); };
```

Un prédicat sémantique est un bloc de code arbitraire dans le langage cible entouré de `{ ... }?` , qui évalue une valeur booléenne. Si la valeur renvoyée est false, la règle lexer est ignorée.

```
IDENTIFIER: [A-Z]+ { identifierIsValid() }?;
```

Les prédicats sémantiques doivent être définis à la fin de la règle chaque fois que cela est possible pour des raisons de performances.

Lire Règles Lexer en v4 en ligne: https://riptutorial.com/fr/antlr/topic/3271/regles-lexer-en-v4

Chapitre 7: TestRig / grun

Examples

Configuration de TestRig

ANTLR contient un outil de test dans sa bibliothèque d'exécution. Cet outil peut être utilisé pour afficher des informations détaillant la manière dont l'analyse est effectuée pour faire correspondre les entrées aux règles définies dans votre fichier de grammaire.

Pour utiliser cet outil contenu dans le fichier JAR ANTLR, vous devez configurer votre chemin de classe système pour autoriser l'accès à l'outil ANTLR et à la bibliothèque d'exécution:

```
export CLASSPATH=".:/usr/local/lib/antlr-4.5.3-complete.jar:$CLASSPATH"
```

> Remarque: Assurez-vous que le point précède tout chemin pour garantir que la machine virtuelle Java ne verra pas les classes dans votre répertoire de travail actuel.

Alises peut être utilisé sous Linux / MAC / Unix pour simplifier les commandes utilisées:

```
alias antlr4='java -jar /usr/local/lib/antlr-4.5.3-complete.jar'
//or any directory where your jar is located
```

Notez que la configuration des fenêtres pour les alias et la configuration des classpath peuvent être plus compliquées, voir ici pour plus de détails.

Accéder à TestRig

Une fois que vous avez configuré votre alias, vous pouvez configurer TestRig de la manière suivante, en utilisant à nouveau un alias, car cela réduit le temps requis pour effectuer l'action:

```
alias grun='java org.antlr.v4.runtime.misc.TestRig'
```

Si vous ne souhaitez pas configurer un alias sur Windows, vous pouvez accéder à TestRig en exécutant la commande suivante au même emplacement que votre répertoire jar ANTLR:

```
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.runtime.misc.TestRig
//or
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig
```

Pour exécuter TestRig sur votre grammaire, vous pouvez transmettre les paramètres de votre grammaire comme suit:

```
grun yourGrammar yourRule -tree //using the setup alias
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig yourGrammar YourRule -tree //on
windows with no alias
```

```
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig yourGrammar Hello r -tree
//Windows with the grammar Hello.g4 starting from the rule 'r'.
```

Construire une grammaire avec l'arborescence d'analyse visuelle

`-gui` spécifiez l'option de ligne de commande `-gui` lors de l'exécution d'une grammaire ANTLR dans le banc d'essai, une fenêtre apparaîtra avec une représentation visuelle de l'arbre d'analyse. Par exemple:

Compte tenu de la grammaire suivante:

JSON.g4

```
/** Taken from "The Definitive ANTLR 4 Reference" by Terence Parr */

// Derived from http://json.org
grammar JSON;

json
   : value
   ;

object
   : '{' pair (',' pair)* '}'
   | '{' '}'
   ;

pair
   : STRING ':' value
   ;

array
   : '[' value (',' value)* ']'
   | '[' ']'
   ;

value
   : STRING
   | NUMBER
   | object
   | array
   | 'true'
   | 'false'
   | 'null'
   ;


STRING
   : '"' (ESC | ~ ["\\])* '"'
   ;
fragment ESC
   : '\\' (["\\/bfnrt] | UNICODE)
   ;
fragment UNICODE
   : 'u' HEX HEX HEX HEX
   ;
fragment HEX
   : [0-9a-fA-F]
```

```
   ;
NUMBER
   : '-'? INT '.' [0-9] + EXP? | '-'? INT EXP | '-'? INT
   ;
fragment INT
   : '0' | [1-9] [0-9]*
   ;
// no leading zeros
fragment EXP
   : [Ee] [+\-]? INT
   ;
// \- since - means "range" inside [...]
WS
   : [ \t\n\r] + -> skip
   ;
```

Étant donné le fichier JSON suivant:

example.json

```
{
  "name": "John Doe",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

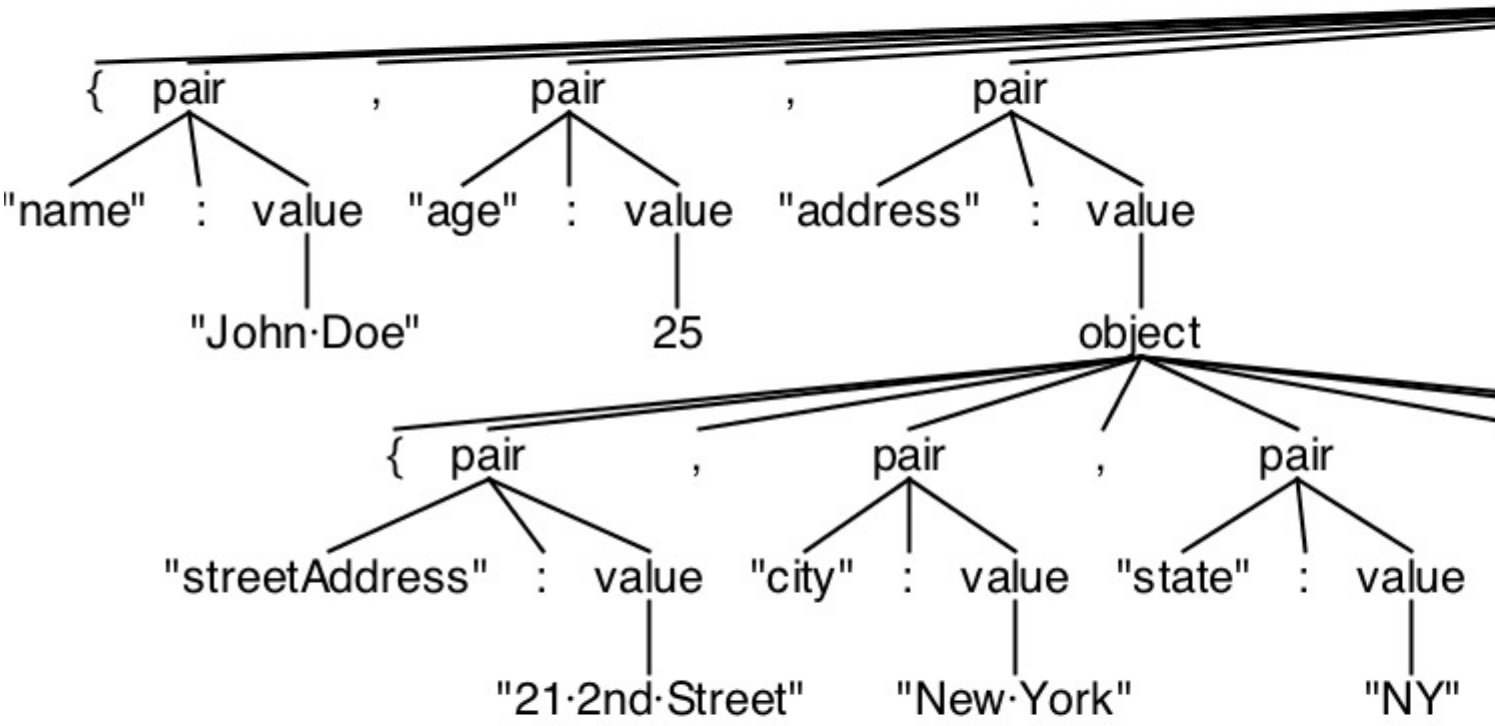
La syntaxe de ligne de commande suivante:

```
export CLASSPATH=".:/usr/local/lib/antlr-4.0-complete.jar:$CLASSPATH"

alias antlr4='java -jar /usr/local/lib/antlr-4.0-complete.jar'

alias grun='java org.antlr.v4.runtime.misc.TestRig'

antlr4 -o . -lib . -no-listener -no-visitor  JSON.g4; javac *.java; grun JSON json -gui
example.json
```

entraînera les fichiers **.java** & **.tokens générés** , ainsi que les fichiers **.class** compilés:

```
JSON.g4                          JSONLexer.class                JSONListener.java
JSONParser$PairContext.class     JSON.tokens                    JSONLexer.java
JSONParser$ArrayContext.class    JSONParser$ValueContext.class  JSONBaseListener.class
JSONLexer.tokens                 JSONParser$JsonContext.class   JSONParser.class
JSONBaseListener.java            JSONListener.class
JSONParser$ObjectContext.class   JSONParser.java
```

et l'arbre d'analyse suivant:



Lire TestRig / grun en ligne: https://riptutorial.com/fr/antlr/topic/3270/testrig---grun

Chapitre 8: Visiteurs

Introduction

Quelle est la différence entre un auditeur et un visiteur? La différence entre les mécanismes d'écoute et de visiteur est que les méthodes d'écoute sont appelées par l'objet walker fourni par ANTLR, tandis que les méthodes visiteur doivent guider leurs enfants avec des appels de visite explicites. Oublier d'appeler visit () sur les enfants d'un nœud signifie que ces sous-arbres ne sont pas visités. En visiteur, nous avons la possibilité de marcher dans les arbres, alors qu'en écoute, vous ne faites que réagir au déambulateur.

Examples

Exemple

Exemple de grammaire (Expr.g4)

```
grammar Expr;
prog:    (expr NEWLINE)* ;
expr:    expr ('*'|'/') expr

    |    expr ('+'|'-') expr
    |    INT
    |    '(' expr ')'
    ;
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

Générer le visiteur

Pour générer un visiteur ou désactiver un visiteur pour votre grammaire, utilisez les indicateurs suivants:

```
 -visitor           generate parse tree visitor
 -no-visitor        don't generate parse tree visitor (default)
```

La commande commandline / terminal pour construire votre grammaire avec un visiteur sera formatée comme indiqué ci-dessous, en ce qui concerne les alias choisis et les alias possibles:

```
java - jar antlr-4.5.3-complete.jar Expr.g4 -visitor
java - jar antlr-4.5.3-complete.jar Expr.g4 -no-visitor
```

La sortie sera un analyseur / lexer avec un visiteur ou aucun visiteur, respectivement.

Sortie La sortie sera **ExprBaseVisitor.java** et **ExprVisitor.java** pour cet exemple. Ce sont les fichiers java pertinents pour vous permettre de mettre en œuvre les fonctionnalités des visiteurs. Il est souvent idéal de créer une nouvelle classe et d'étendre ExprBaseVisitor pour implémenter de

nouvelles fonctionnalités pour chaque méthode.

```
// Generated from Expr.g4 by ANTLR 4.5.3
import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;

/**
 * This class provides an empty implementation of {@link ExprVisitor},
 * which can be extended to create a visitor which only needs to handle a subset
 * of the available methods.
 *
 * @param <T> The return type of the visit operation. Use {@link Void} for
 * operations with no return type.
 */
public class ExprBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements ExprVisitor<T>
{
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitProg(ExprParser.ProgContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitExpr(ExprParser.ExprContext ctx) { return visitChildren(ctx); }
}
```

Lire Visiteurs en ligne: https://riptutorial.com/fr/antlr/topic/8211/visiteurs

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec ANTLR	Athafoud, cb4, Community, D3181, Gábor Bakos, KvanTTT
2	ANTLR Cibles / Runtimes linguistiques	D3181
3	Introduction à ANTLR v3	Athafoud, cb4
4	Introduction à ANTLR v4	Athafoud, cb4, Community, D3181, Devid, Gábor Bakos, GRosenberg, Lucas Trzesniewski
5	Les auditeurs	bn., Lucas Trzesniewski
6	Règles Lexer en v4	Athafoud, bn., Loxley, Lucas Trzesniewski
7	TestRig / grun	bn., D3181, Lucas Trzesniewski, Pascal Le Merrer
8	Visiteurs	D3181