EBook Gratis

APRENDIZAJE ANTLR

Free unaffiliated eBook created from
Stack Overflow contributors.

#antlr

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con ANTLR

Observaciones

ANTLR (otra herramienta para el reconocimiento de idiomas) es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto binario o texto estructurado. Es ampliamente utilizado para construir lenguajes, herramientas y marcos. Desde una gramática, ANTLR genera un analizador que puede construir y caminar árboles de análisis.

- Sitio web oficial de antlr (siempre apunta a la última versión)

Versiones de Antlr
Antlr se divide en dos grandes partes, la gramática (archivos de gramática) y los archivos de código generados, que se derivan de la gramática basada en el idioma de destino. Las versiones de antlr están en el formato de V1.V2.V3:

- V1: el cambio en V1 significa que se introdujo una nueva sintaxis de características en los archivos de gramática
- V2: el cambio en V2 significa que se introdujeron nuevas características o correcciones importantes en los archivos generados (por ejemplo, adición de nuevas funciones)
- V3: significa correcciones de errores o mejoras menores

Bibliotecas en tiempo de ejecución y objetivos de generación de código
La herramienta Antlr está escrita en Java, sin embargo, es capaz de generar analizadores y lexers en varios idiomas. Para ejecutar el analizador y el lexer también necesitará tener la biblioteca de ejecución de antlr junto con el analizador y el código del lexer. El idioma de destino admitido (y las bibliotecas de tiempo de ejecución) son los siguientes:

- Java
- DO#
- Python (2 y 3)
- JavaScript

Versiones

Versión	Fecha de lanzamiento
2.0	1997-05-01
3.0	2011-01-19
4.0	2013-01-21
4.1	2013-07-01

Versión	Fecha de lanzamiento
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

Hola Mundo

Una simple gramática de hola mundo se puede encontrar aquí :

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

Para compilar este ejemplo .g4, puede ejecutar el siguiente comando desde su terminal / línea de comandos de sistemas operativos:

```
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 creación de este ejemplo debería generar el siguiente resultado en el directorio de archivos Hello.g4:

1. Hello.tokens
2. HelloBaseListener.java

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

Cuando utilice estos archivos en su propio proyecto, asegúrese de incluir el archivo jar ANTLR. Para compilar todos estos archivos utilizando Java, en el mismo directorio operativo o por ruta, ejecute el siguiente comando:

```
javac *.java
```

Lea Empezando con ANTLR en línea: https://riptutorial.com/es/antlr/topic/4453/empezando-con-antlr

Capítulo 2: Introducción a ANTLR v3

Examples

Instalación y configuración

Cómo instalar ANTLR en Eclipse

(Última prueba en Indigo y ANTLR IDE 2.1.2)

1. Instale Eclipse.
2. Descarga el tarro de binarios completo de ANTLR que incluye ANTLR v2. Extraer a un directorio temporal. Copie la carpeta antlr-nn en una ubicación permanente apropiada, por ejemplo, la misma carpeta en la que está instalado Eclipse.
3. Agregue el sitio de actualización IDE de ANTLR a Eclipse.
    - En Eclipse, haga clic en Ayuda y seleccione Instalar nuevo software.
    - Haga clic en el botón Agregar ...
    - En la ventana Agregar repositorio, para Ubicación escriba http://antlrv3ide.sourceforge.net/updates y escriba algo como ANTLR IDE para el Nombre y haga clic en Aceptar para volver a la ventana de Software disponible.
    - Marque la casilla de ANTLR IDE vn.nn y haga clic en hasta que esté instalado. Eclipse probablemente se reiniciará.
4. Configurar el IDE ANTLR.
    - En la ventana principal de Eclipse, haga clic en Ventana y luego en Preferencias.
    - En el panel izquierdo, expanda ANTLR y seleccione Generador.
    - En el panel derecho, haga clic en el botón Agregar ...
    - En la ventana Agregar paquete ANTLR, haga clic en Directorio ... y navegue hasta la ubicación de la carpeta antlr-nn y haga clic en Aceptar.
    - Haga clic en Aceptar para cerrar la ventana Agregar paquete ANTLR.
    - Seleccione Generador de código en el panel izquierdo y haga clic en la carpeta relativa al Proyecto en el panel derecho. Escriba un nombre de carpeta. Ejemplos: antlr-java o antlr-generados.
    - Seleccione cualquier otro parámetro de configuración pero NO marque –nfa o –dfa en la sección General en la ventana de Construcción. Si está marcado, esto causará errores ANTLR que impiden que se generen archivos java en la carpeta de salida.
    - Haga clic en Aceptar para cerrar la ventana de Preferencias.
5. Crea un nuevo proyecto Java y habilita el soporte ANTLR.
    - Desde la ventana principal de Eclipse, vaya a Archivo, Nuevo, Proyecto Java. Haga clic en Siguiente, escriba un nombre de proyecto y haga clic en Finalizar.
    - Para habilitar el soporte de ANTLR para el proyecto, en la ventana del Explorador de paquetes (panel izquierdo), haga clic con el botón derecho en el proyecto que acaba de crear y seleccione Configurar, Convertir a proyecto ANTLR.
    - Agregue el archivo jar completo de ANTLR al proyecto: haga clic con el botón derecho en el proyecto y seleccione Propiedades, Java Build Path, haga clic en Agregar JAR

externos ..., busque el archivo jar de ANTLR, selecciónelo y haga clic en Aceptar. Haga clic en Aceptar para cerrar la ventana Propiedades del proyecto.

6. Crear una gramática ANTLR.
    - Cree una nueva gramática ANTLR: haga clic con el botón derecho en la carpeta src del proyecto, luego en Archivo, Nuevo, Otro, expanda ANTLR y seleccione Gramática combinada. Haga clic en Siguiente, escriba el nombre de la gramática, seleccione una opción de Idioma y haga clic en Finalizar.
    - Se crea un archivo ".g" con las opciones seleccionadas y una regla en blanco. Agregue las opciones language = Java, @header, @lexer :: header y @members en la parte superior (vea el ejemplo). La finalización automática es la forma más fácil de agregarlos (presione CTRL-espacio para que aparezca la lista de finalización automática).
7. Guarda la gramática.
    - Cuando se guarda, una carpeta que contenga el código Java generado para la gramática debe aparecer en el Explorador de proyectos. Si no es así, asegúrese de que las opciones –nfa o –dfa no estén marcadas en Preferencias ANTLR en General en la ventana de Construcción (Paso 4g). [Confirme si es necesario: verifique que la variable de entorno CLASSPATH apunta al Java7 que coincida con su instalación de Eclipse (32 o 64 bits) y la variable de entorno de Windows Path tenía Java7 SDK.]
    - Para evitar errores de Java "no se pueden resolver en un tipo", haga clic con el botón derecho en la carpeta que contiene el código Java generado, luego en Crear ruta, Usar como carpeta de origen.

MUESTRA COMBINADA DE GRAMÁ

```
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';
```

Lea Introducción a ANTLR v3 en línea: https://riptutorial.com/es/antlr/topic/6629/introduccion-a-antlr-v3

Capítulo 3: Introducción a ANTLR v4

Observaciones

ANTLR v4 es una poderosa herramienta utilizada para crear nuevos lenguajes de programación y procesar / traducir texto binario o texto estructurado. ANTLR utiliza una gramática que usted crea para generar un analizador que puede construir y atravesar un árbol de análisis (o árbol de sintaxis abstracta, AST). El analizador consta de archivos de salida en el idioma de destino que especifique. ANTLR v4 admite varios destinos, incluidos: Java, C #, JavaScript, Python2 y Python3. Se está trabajando en el soporte para C ++. Para trabajar en IDE de GUI, existen complementos para Visual Studio, Intellij, NetBeans y Eclipse.

Para información general, visite el sitio web de ANTLR . Para hablar en serio sobre ANTLR, echa un vistazo al libro altamente recomendado escrito por Terrence Parr (el tipo que creó ANTLR) La referencia definitiva de ANTLR 4 .

Información importante de la versión

- 4.5: 01/22/15 - Se agregó el objetivo de JavaScript y se actualizó el objetivo de C #. 4.5 Notas de la versión
- 4.4: 07/16/14 - Se agregaron Python2 y Python3 como objetivos. 4.4 Notas de la versión
- 4.3: 18/06/14 - Corrección de errores mayores; Preparado para añadir nuevos objetivos. 4.3 Notas de la versión
- 4.2: 02/04/14 - Sintaxis mejorada para seleccionar / hacer coincidir árboles de análisis. 4.2 Notas de la versión
- 4.1: 30/06/13 - Mejora el rendimiento de análisis; exportar ASTs a PNG. 4.1 Notas de la versión
- 4.0: 01/21/13 - Lanzamiento inicial.

Examples

Instalación para uso de línea de comandos

ANTLR se distribuye como un archivo Java Jar. Se puede descargar aquí . Como ANTLR se compila como un archivo jar, posteriormente requiere que el entorno de ejecución de Java funcione, si no lo tiene, puede descargarlo aquí .

Una vez que se haya descargado el archivo JAR ANTLR, puede ejecutar ANTLR desde la línea de comandos de la misma manera que cualquier otro archivo JAR:

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

(Suponiendo que está operando en el mismo directorio que el archivo antlr-4.5.3-complete.jar).

Esto debería producir algo similar a esto:

```
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
```

Otras acciones recomendadas para la instalación incluyen:

```
     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 %*
```

Después de la configuración, puede crear una aplicación utilizando su archivo de gramática .g4:

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

También puede crear una aplicación en otros idiomas con el parámetro -Dlanguage. Por ejemplo, para generar archivos C # harías algo como esto:

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

Consulte aquí la lista completa de gramáticas prefabricadas para lenguajes de programación comunes.

Instalación usando herramientas de Build Automation

Descargue la última versión de ANTLR y extráigala a una carpeta.

También puede usar Maven, Gradle u otra herramienta de compilación para depender de su tiempo de ejecución (las clases que usan las gramáticas generadas): `org.antlr:antlr4-runtime` .

Para poder generar automáticamente el analizador en un proyecto de maven, como parte del proceso de compilación, use el complemento de Maven : `org.antlr:antlr4` .

Instalar en Eclipse y construir Hello World

(Probado con ANTLR 4.5.3, Eclipse Neon, ANTLR 4 IDE 0.3.5 y Java 1.8)

1. Descarga la última versión de ANTLR . Asegúrate de obtener el tarro de archivos binarios de Java de ANTLR completo. Guarde en cualquier ubicación apropiada, por ejemplo, la carpeta donde se almacenan otras bibliotecas de Java. No importa dónde, solo recuerda la ubicación.

2. Instale el IDE ANTLR en Eclipse.

    - Desde el menú de Eclipse, haga clic en Ayuda y seleccione Eclipse Marketplace.
    - En el cuadro Buscar: escriba antlr y haga clic en Ir.
    - Haga clic en Instalar para ANTLR 4 IDE.
    - Haga clic en Finalizar en la ventana Confirmar características seleccionadas.
    - Si aparece una ventana de advertencia de seguridad, haga clic en Aceptar.
    - Reinicie Eclipse.

3. Evite el error "Error al crear el inyector ...".

    - Al acceder a las Preferencias de ANTLR 4 en Eclipse o cuando la variable de entorno HOME no está configurada, ocurre el siguiente error: Error al crear el inyector para com.github.jknack.antlr-4ide.Antlr4 para com.github.jknack.antlr-4ide.Antlr4 .
    - Asegúrese de que la variable de entorno HOME esté configurada. Si no, configúrelo como corresponda para su sistema.
    - Descargue Xtext 2.7.3 en la misma ubicación que antlr-nnn-complete.jar.
    - En Eclipse, haga clic en Ayuda y seleccione Instalar nuevo software.
    - Haga clic en Agregar ... para acceder a la ventana Agregar repositorio.
    - Escriba un nombre, xtext 2.7.3 por ejemplo, luego haga clic en Archivar ..., navegue hasta el archivo Xtext 2.7.3 y selecciónelo, luego haga clic en Aceptar.
    - En la ventana de instalación, haga clic en el botón Seleccionar todo y luego haga clic en Siguiente> dos veces, acepte el acuerdo de licencia. y haga clic en Finalizar.
    - Reinicie Eclipse.

4. Dígale a Eclipse / Java dónde está ANTLR.

    - En Eclipse, haga clic en Ventana y seleccione Preferencias.
    - En el panel izquierdo, expanda Java y Build Path, luego seleccione Classpath Variables.
    - En el panel derecho, haga clic en Nuevo ..., ingrese un Nombre y haga clic en Archivo ... y busque su ubicación de antlr-nnn-complete.jar. Haga clic en Aceptar para volver a la ventana Variables de la ruta de clase.
    - Haga clic en Aceptar para salir de las Preferencias.

5. (Opcional) Configure el directorio de fuentes generadas por el IDE ANTLR.

    - En la ventana principal de Eclipse, haga clic en Ventana y luego en Preferencias.
    - En el panel izquierdo, expanda ANTLR 4 y seleccione Herramienta.

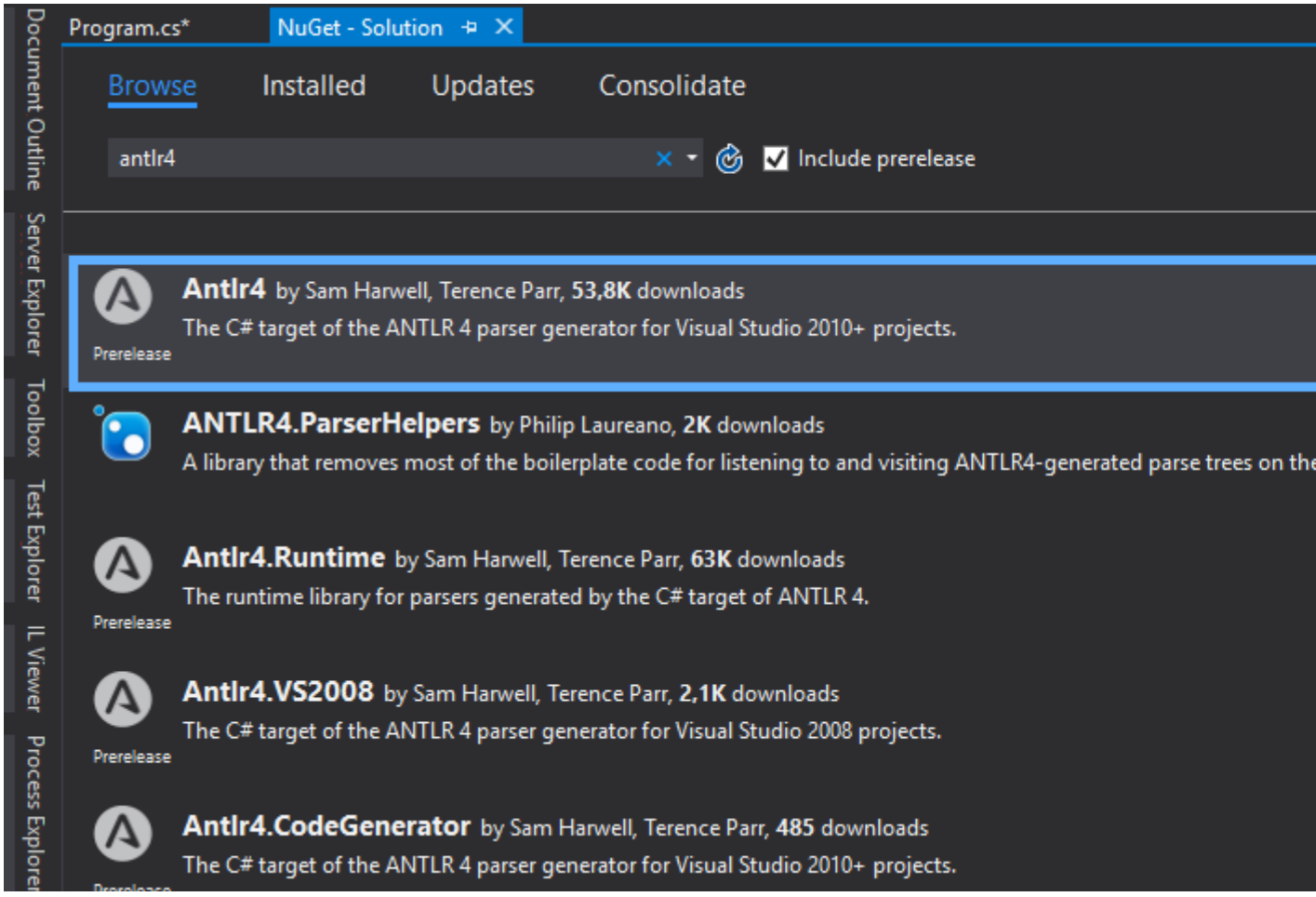
- En Opciones, cambie el directorio si lo desea. Por ejemplo, java es mi idioma de destino, así que uso ./antlr-java.
- Haga clic en Aceptar para cerrar la ventana de Preferencias.

6. Crea un proyecto ANTLR 4.

   - Desde la ventana principal de Eclipse, vaya a Archivo, Nuevo, Proyecto.
   - En la ventana Nuevo proyecto, expanda General y seleccione Proyecto ANTLR 4.
   - Haga clic en Siguiente, escriba un nombre de proyecto y haga clic en Finalizar.
   - El nuevo proyecto predeterminado contiene un archivo Hello.g4 y generará automáticamente el programa estándar "Hello World".
   - En el Explorador de paquetes, expanda la nueva carpeta del proyecto para ver el archivo g4 y una carpeta llamada target (o el nombre que le dio en el Paso 5) que contiene los archivos de origen de destino.

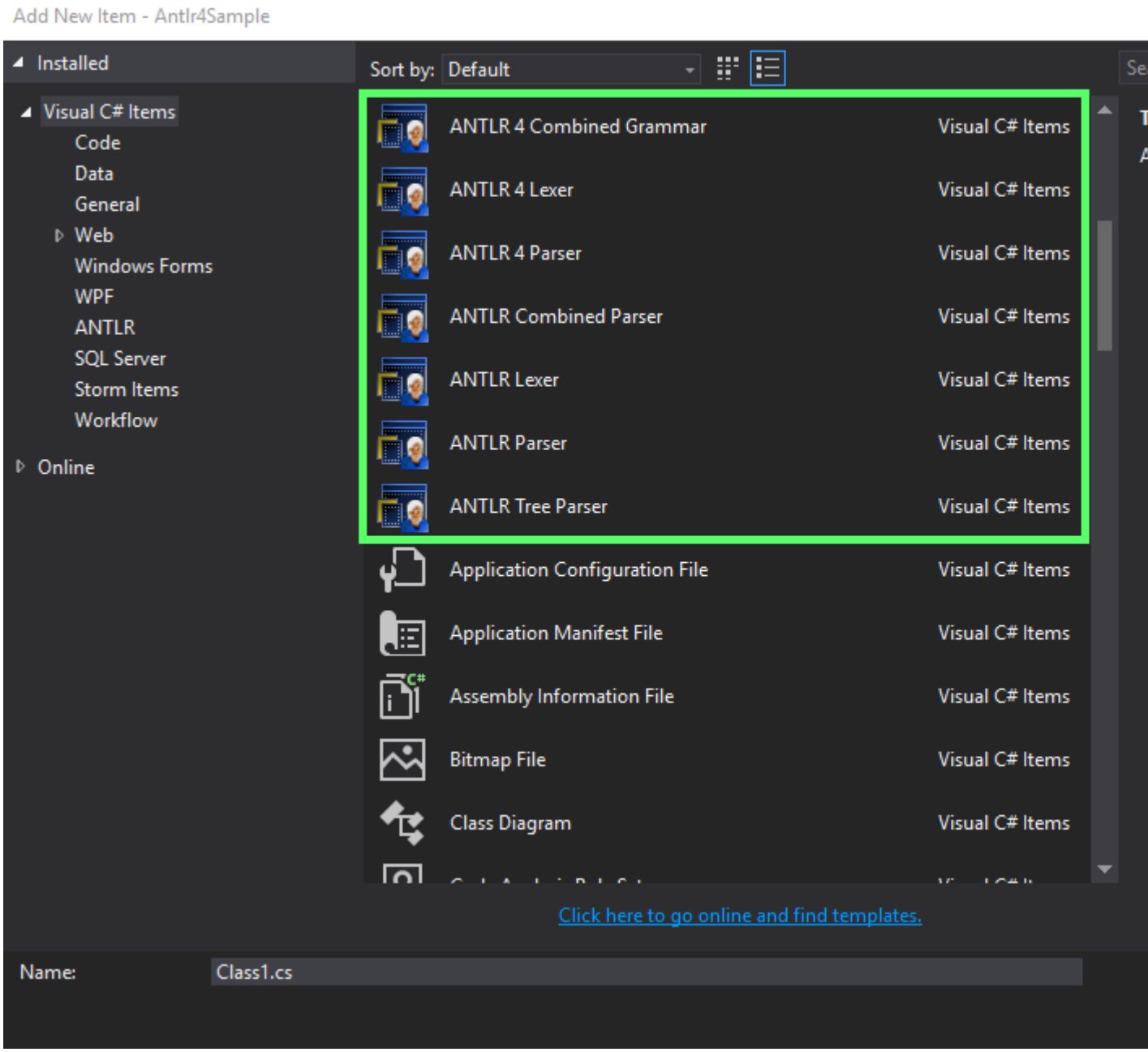
Instalando ANTLR en Visual Studio 2015 (usando Nuget)

1. Abra Visual Studio 2015, vaya a Herramientas → Extensiones → En línea y busque Antlr. Descargue la extensión ANTLR Language Support (creada por Sam Harwell) y reinicie Visual Studio.
2. Crear nuevo proyecto de aplicación de consola. Haga clic con el botón derecho en la Solución → Administrar paquetes de Nuget para la Solución → Examinar (pestaña) y busque Antlr4 e instálelo.

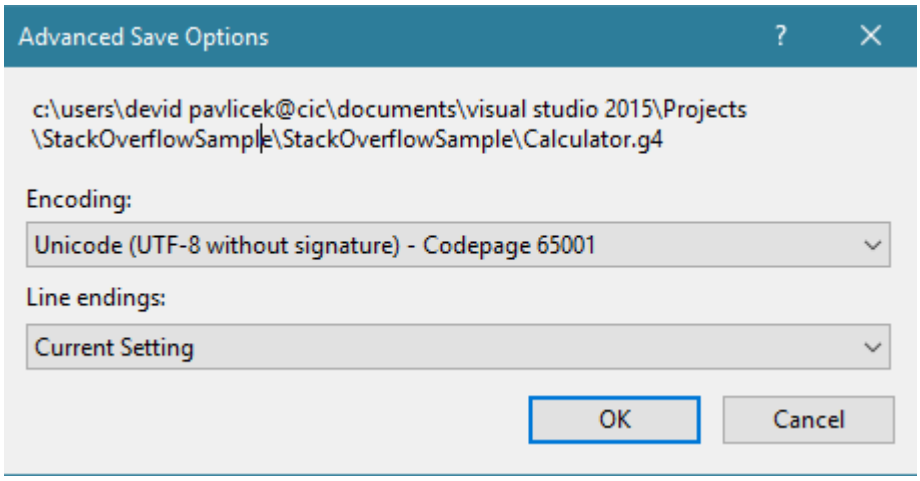


3. Agregue un nuevo elemento a su proyecto haciendo clic derecho en él. Y busque plantillas

ANTLR4.

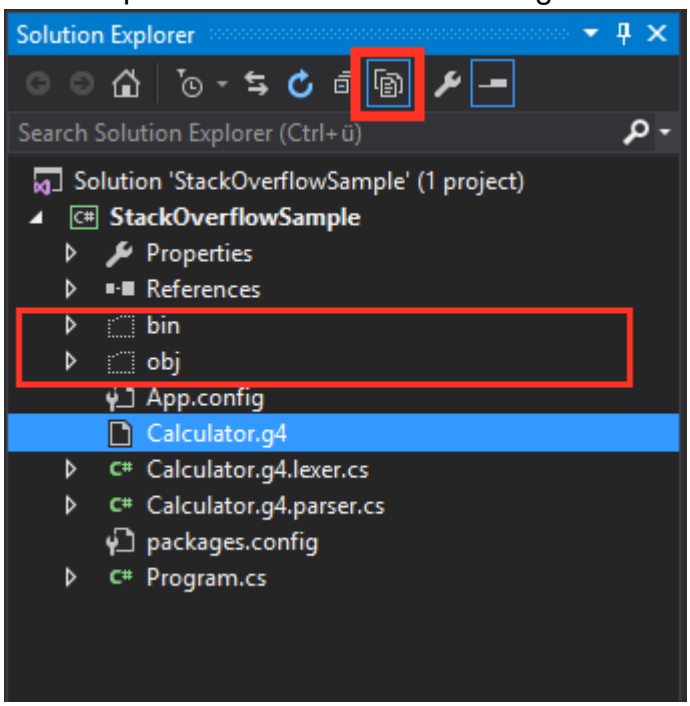


4. Desde su archivo ANTLR (finalizando .g4) vaya a Archivo → Opciones avanzadas de guardado y busque Unicode (UTF-8 **sin firma**) - Página de códigos 65001 y haga clic en Aceptar. Eso es.

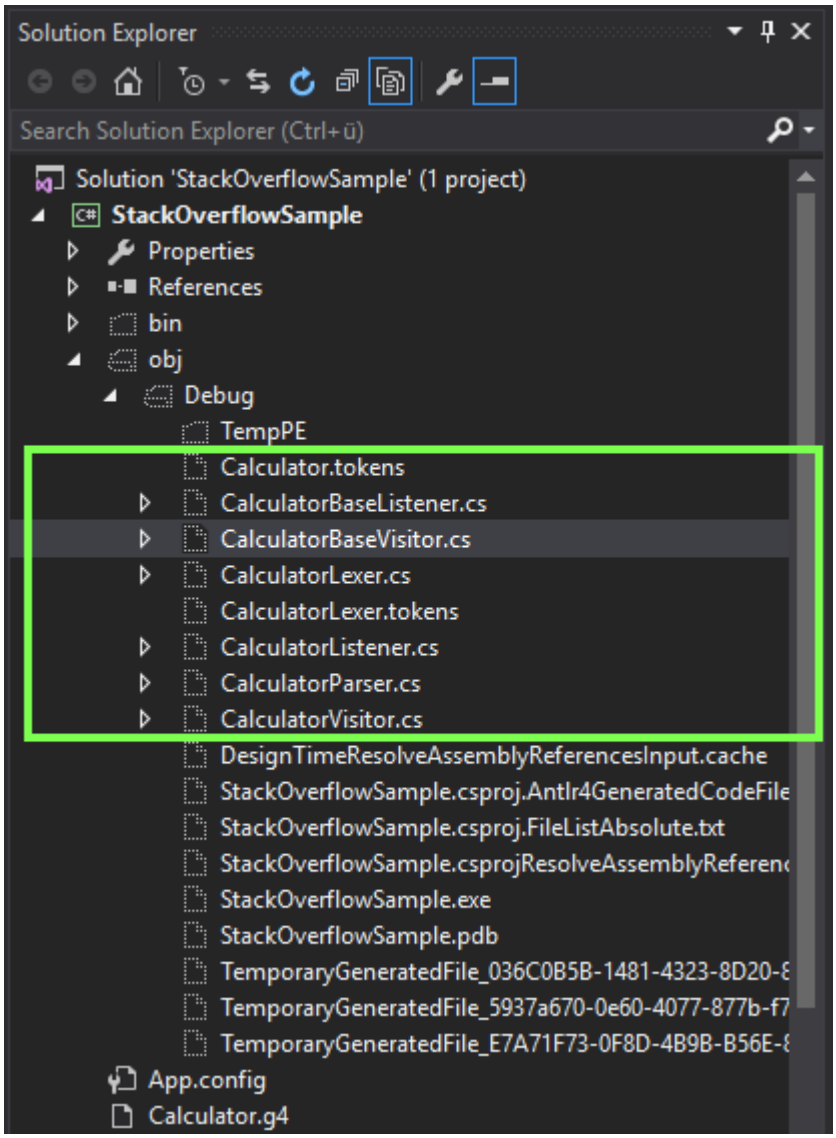


Prueba si todo funciona

- Cree un elemento de gramática combinada ANTLR 4 y asígnele el nombre Calculator.g4
- Copie y pegue el código fuente de la Calculadora de este proyecto de Github aquí: Calculadora por Tom Everett
- Cambiar la calculadora gramatical a la calculadora gramatical
- En el Explorador de soluciones → Haga clic en Mostrar todos los archivos.



- Guardar y Ejecutar (Iniciar) el proyecto
- En el Explorador de soluciones, en la carpeta obj, debería ver las clases de cs generadas como el visitante y el oyente. Si este es el caso que tuvo éxito. Ahora puede comenzar a trabajar con ANTLR en Visual Studio 2015.



Lea Introducción a ANTLR v4 en línea: https://riptutorial.com/es/antlr/topic/2856/introduccion-a-antlr-v4

Capítulo 4: Objetivos ANTLR / tiempos de ejecución de idiomas

Examples

Ayuda de idioma

ANTLR es capaz de generar analizadores para varios lenguajes de programación:

1. C # Target
2. Objetivo de Python
3. Objetivo de JavaScript
4. Java Target

De forma predeterminada, ANTLR generará un analizador desde la línea de comandos en el lenguaje de programación Java:

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

Para cambiar el idioma de destino, puede ejecutar el siguiente comando desde el terminal del sistema operativo / línea de comandos:

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

En lugar de usar el parámetro '-Dlanguage' en la línea de comandos / terminal cada vez que cree el analizador deseado para un idioma específico, puede seleccionar el destino desde su archivo de gramática .g4 al incluir el destino en la sección global:

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

Para usar la salida del analizador generada, asegúrese de tener el tiempo de ejecución ANTLR para el idioma especificado:

1. Tiempo de ejecución CSharp
2. Python 2 runtime
3. Python 3 runtime

Instrucciones completas e información sobre las bibliotecas de tiempos de ejecución de ANTLR

Configuración del analizador Python

Después de ejecutar su archivo de gramática .g4 con ANTLR.jar, debe tener una cantidad de archivos generados, tales como:

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

Para usarlos en un proyecto de Python, incluya el tiempo de ejecución de Python en su área de trabajo para que cualquier aplicación que esté desarrollando pueda acceder a la biblioteca ANTLR. Esto se puede hacer extrayendo el tiempo de ejecución en su carpeta de proyecto actual o importándolo dentro de su IDE en las dependencias de su proyecto.

```
#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)
```

Esta configuración incluye su analizador y acepta la entrada desde la línea de comandos para permitir el procesamiento de un archivo pasado como parámetro.

```
#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')
```

Estos archivos, junto con el tiempo de ejecución ANTLR y sus archivos generados a partir de su archivo de gramática, aceptarán un solo nombre de archivo como argumento y leerán y analizarán sus reglas gramaticales.

Para ampliar la funcionalidad básica, también debe ampliar la escucha predeterminada para manejar eventos relevantes para tokens que se encuentran durante el tiempo de ejecución.

Lea Objetivos ANTLR / tiempos de ejecución de idiomas en línea:
https://riptutorial.com/es/antlr/topic/3414/objetivos-antlr---tiempos-de-ejecucion-de-idiomas

Capítulo 5: Oyentes

Examples

Eventos de escucha usando etiquetas

El etiquetado de las alternativas dentro de una regla que comienza con el operador `#` le indica a ANTLR que genere métodos de escucha para cada etiqueta correspondiente a la alternativa.

Al especificar una etiqueta para cada alternativa en la siguiente regla:

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

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

`ParseTreeListener` los siguientes métodos en la interfaz generada que extiende `ParseTreeListener` :

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

Lea Oyentes en línea: https://riptutorial.com/es/antlr/topic/6717/oyentes

Capítulo 6: Reglas Lexer en v4

Examples

Reglas simples

Las reglas de Lexer definen tipos de token. Su nombre debe comenzar con una letra mayúscula para distinguirlos de las reglas del analizador.

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

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

Sintaxis básica:

Sintaxis	Sentido
`A`	Coincidir con la regla de lexer o fragmento llamado `A`
`AB`	Match `A` seguido de `B`
`(A|B)`	Coincidir con `A` o `B`
`'text'`	Emparejar literal *"texto"*
`A?`	Match `A` cero o una vez
`A*`	Match `A` cero o más veces
`A+`	Match `A` una o más veces
`[A-Z0-9]`	Haga coincidir un carácter en los rangos definidos (en este ejemplo entre AZ o 0-9)
`'a'..'z'`	Sintaxis alternativa para un rango de caracteres
`~[AZ]`	Negación de un rango - coincide con cualquier carácter individual que *no esté* en el rango
`.`	Coincidir con cualquier carácter individual

Fragmentos

Los fragmentos son partes reutilizables de las reglas de lexer que no pueden coincidir por sí solas: deben ser referenciados desde una regla de lexer.

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

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

Reglas lexer implícitas

Cuando se usan tokens como `'{'` en una regla de *analizador* , se creará una regla implícita de lexer para ellos a menos que exista una regla explícita.

En otras palabras, si tiene una regla lexer:

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

Entonces estas dos reglas del analizador son equivalentes:

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

Pero si la regla `OPEN_BRACE` *no* está definida, se creará una regla anónima implícita. En ese caso, la regla implícita se definirá *como si* se definiera *antes que* las otras reglas: tendrá una prioridad más alta que otras reglas.

Reglas de prioridad

Varias reglas lexer pueden coincidir con el mismo texto de entrada. En ese caso, el tipo de token se elegirá de la siguiente manera:

- Primero, seleccione la regla lexer que coincida con la entrada *más larga*
- Si el texto coincide con un token definido implícitamente (como `'{'`), use la regla implícita
- Si varias reglas lexer coinciden con la misma longitud de entrada, elija la *primera* , según el orden de definición

---

La siguiente gramática combinada:

```
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;
```

Dada la siguiente entrada:

```
aaa foo bar baz barz
```

Producirá la siguiente secuencia de token desde el lexer:

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

- `aaa` es de tipo `IDENTIFIER`

  Solo la regla `IDENTIFIER` puede coincidir con este token, no hay ambigüedad.

- `foo` es de tipo `'foo'`

  La regla del analizador `randomParserRule` introduce el tipo de token `'foo'` implícito, que es prioritario sobre la regla `IDENTIFIER` .

- `bar` es de tipo `BAR`

  Este texto coincide con la regla `BAR` , que se define *antes que* la regla `IDENTIFIER` , y por lo tanto tiene prioridad.

- `baz` es de tipo `IDENTIFIER`

  Este texto coincide con la regla `BAZ` , pero también coincide con la regla `IDENTIFIER` . Este último es elegido como se define *antes de* `BAR` .

  Dada la gramática, `BAZ` *nunca* podrá coincidir, ya que la regla `IDENTIFIER` ya cubre todo lo que `BAZ` puede igualar.

- `barz` es de tipo `IDENTIFIER`

  La regla `BAR` puede coincidir con los primeros 3 caracteres de esta cadena (`bar`), pero la regla `IDENTIFIER` coincidirá con 4 caracteres. Como `IDENTIFIER` coincide con una subcadena más larga, se elige sobre `BAR` .

Como regla general, las reglas específicas deben definirse *antes que* las reglas más genéricas. Si una regla solo puede coincidir con una entrada que ya está cubierta por una regla definida previamente, *nunca* se utilizará.

Las reglas definidas implícitamente, como `'foo'` actúan como si estuvieran definidas *antes que* todas las demás reglas del lexer.

Comandos de Lexer

Una regla lexer puede tener *comandos* asociados:

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

Los comandos se definen después de un `->` al final de la regla.

- `skip` : `skip` el texto coincidente, no se emitirá ningún token
- `channel(n)` : emite el token en un canal diferente
- `type(n)` : cambia el tipo de token emitido
- `mode(n)` , `pushMode(n)` , `popMode` , `more` : controla los modos del lexer

Acciones y predicados semánticos.

Una acción lexer es un bloque de código arbitrario en el idioma de destino rodeado por `{ ... }` , que se ejecuta durante la coincidencia:

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

Un predicado semántico es un bloque de código arbitrario en el idioma de destino rodeado por `{ ... }?` , que se evalúa a un valor booleano. Si el valor devuelto es falso, se omite la regla de lexer.

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

Los predicados semánticos deben definirse al final de la regla siempre que sea posible por razones de rendimiento.

Lea Reglas Lexer en v4 en línea: https://riptutorial.com/es/antlr/topic/3271/reglas-lexer-en-v4

Capítulo 7: TestRig / grun

Examples

Configurar TestRig

ANTLR contiene una herramienta de prueba en su biblioteca de tiempo de ejecución, esta herramienta puede usarse para mostrar información que detalla cómo se realiza el análisis para hacer coincidir la entrada con las reglas definidas en su archivo de gramática.

Para utilizar esta herramienta contenida en el archivo jar ANTLR, debe configurar la ruta de clase de su sistema para permitir el acceso tanto a la herramienta ANTLR como a la biblioteca de tiempo de ejecución:

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

> Nota: asegúrese de que el Punto preceda a cualquier ruta para garantizar que la máquina virtual java no vea clases en su directorio de trabajo actual.

Alises se puede usar en Linux / MAC / Unix para simplificar los comandos utilizados:

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

La configuración de notas en Windows para alias y la configuración de classpath puede ser más complicada, consulte aquí para obtener detalles más completos.

Accediendo a TestRig

Una vez que haya configurado su alias, puede configurar TestRig de la siguiente manera, nuevamente se recomienda usar un alias, ya que reduce la cantidad de tiempo requerido para realizar la acción:

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

Si no desea configurar un alias en Windows, puede acceder a TestRig ejecutando el siguiente comando en la misma ubicación que su directorio jar de 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
```

Para ejecutar TestRig en su gramática, puede pasar los parámetros para su gramática de la siguiente manera:

```
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'.
```

Construir gramática con Visual Parse Tree

Especificar la opción de línea de comando `-gui` al ejecutar una gramática ANTLR en la plataforma de prueba dará como resultado una ventana emergente con una representación visual del árbol de análisis. Por ejemplo:

Dada la siguiente gramática:

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
   ;
```

Dado el siguiente archivo JSON:

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
}
```

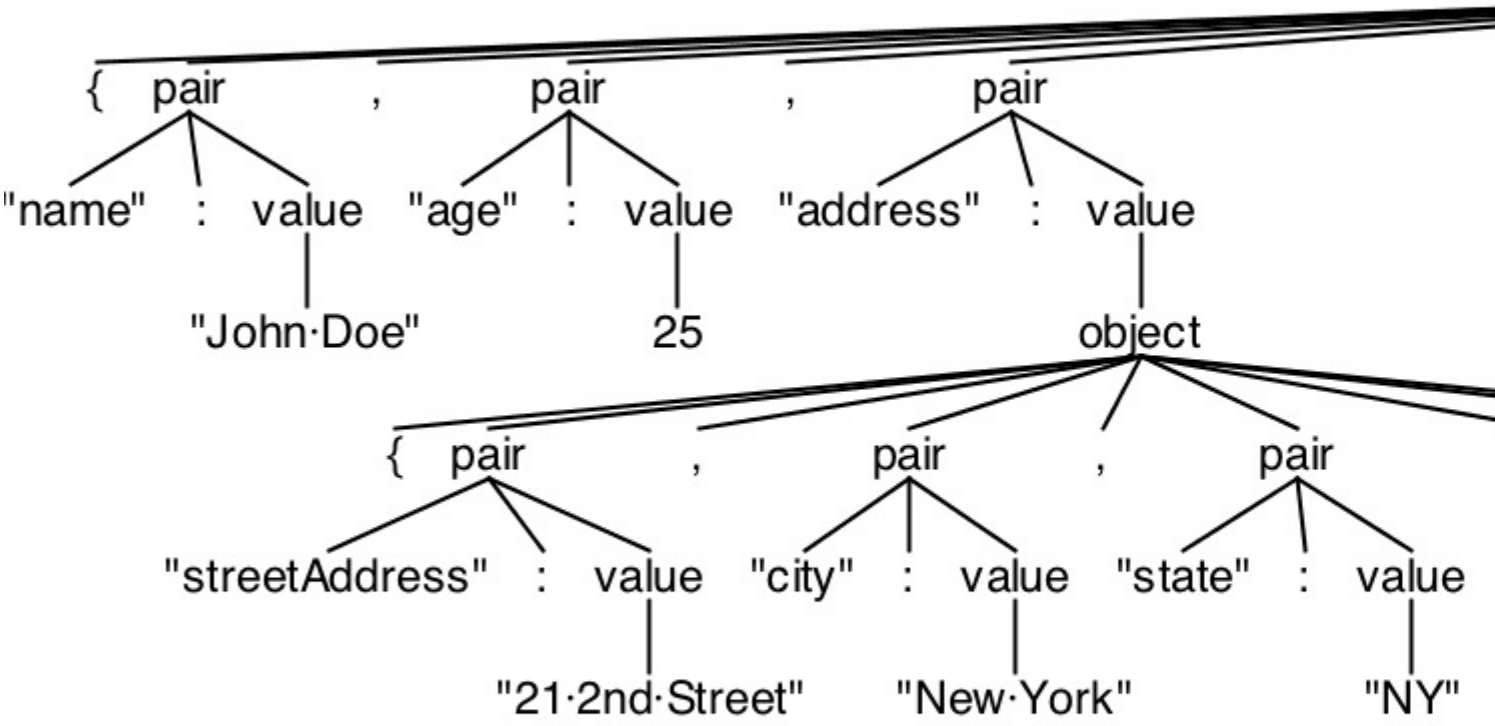
La siguiente sintaxis sintaxis de línea de comandos:

```
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
```

dará como resultado los archivos **.java** y **.tokens** generados, así como los archivos **.class** compilados:

```
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
```

y el siguiente árbol de análisis:



Lea TestRig / grun en línea: https://riptutorial.com/es/antlr/topic/3270/testrig---grun

Capítulo 8: Visitantes

Introducción

¿Cuál es la diferencia entre un oyente y un visitante? La diferencia entre los mecanismos de escucha y visitante es que los métodos de escucha son llamados por el objeto caminante proporcionado por ANTLR, mientras que los métodos de visita deben guiar a sus hijos con llamadas de visita explícitas. Olvidarse de invocar visit () en los hijos de un nodo significa que esos subárboles no son visitados. En el visitante, tenemos la posibilidad de caminar por el árbol, mientras que en el oyente solo estás reaccionando al caminante del árbol.

Examples

Ejemplo

Ejemplo de gramática (ejemplo 4)

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

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

Generando el visitante

Para generar un visitante, o para deshabilitar un visitante para su gramática, use los siguientes indicadores:

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

El comando de línea de comando / terminal para construir su gramática con un visitante se formateará como se muestra a continuación, con respecto a la bandera elegida y los posibles alias:

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

La salida será un analizador / lexer con un visitante o sin visitante, respectivamente.

Salida La salida será **ExprBaseVisitor.java** y **ExprVisitor.java** para este ejemplo. Estos son los archivos java relevantes para que pueda implementar la funcionalidad de visitante. A menudo es

ideal crear una nueva clase y extender ExprBaseVisitor para implementar una nueva funcionalidad de visitante para cada método.

```
// 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); }
}
```

Lea Visitantes en línea: https://riptutorial.com/es/antlr/topic/8211/visitantes

Creditos

S. No	Capítulos	Contributors
1	Empezando con ANTLR	Athafoud, cb4, Community, D3181, Gábor Bakos, KvanTTT
2	Introducción a ANTLR v3	Athafoud, cb4
3	Introducción a ANTLR v4	Athafoud, cb4, Community, D3181, Devid, Gábor Bakos, GRosenberg, Lucas Trzesniewski
4	Objetivos ANTLR / tiempos de ejecución de idiomas	D3181
5	Oyentes	bn., Lucas Trzesniewski
6	Reglas Lexer en v4	Athafoud, bn., Loxley, Lucas Trzesniewski
7	TestRig / grun	bn., D3181, Lucas Trzesniewski, Pascal Le Merrer
8	Visitantes	D3181