ANTLR en Visual Studio Code

Maximiliano A. Eschoyez

2019

Resumen

Esta guía tiene como fin explicar la utilización de ANTLR en la IDE Visual Studio Code. Se explican los pasos mínimos desde la instalación de ANTLR Java hasta compilación de código fuente y la generación de diferentes gráficos de análisis.

1. *plug–in* ANTLR

En la página web de ANTLR se pueden encontrar los *plug–in* para diferentes IDEs.

Si bien para Visual Studio Code existen más herramientas para ANTLR, vamos a utilizar el *plug–in* de Mike Lischke ANTLR4 grammar syntax support (Figura 1).

El *plug–in* completo se encuentra publicado con acceso libre en GitHub. Este documento se basa en la documentación del *plug–in ANTLR*.

Figura 1: ANTLR4 grammar syntax support – Mike Lischke

2. Instalación del *plug–in*

La instalación se puede realizar de dos formas:

1. con el atajo de teclado `Ctl+Shift+x` o *clickeando* el ícono *Extensions* y buscándolo, o
2. con el atajo de teclado `Ctl+p` para ejecutar en el *VS Code Quick Open* el comando

```
ext install mike-lischke.vscode-antlr4
```

NOTA: La biblioteca ANTLR *complete* debe estar en el sistema o incluir el `.jar` en el proyecto para ejecutar el programa.

3. ¿Cómo vamos a trabajar?

Vamos trabajar dentro de un proyecto Java de tipo Maven, por lo tanto, es necesario instalar soporte Java, particularmente el *plug–in* ***Maven for Java*** (`vscjava.vscode-maven`). Para más información, ver la documentación *Java Project Management in VS Code* de la página de Visual Studio Code.

Para simplificar la generación del software, vamos a colocar todos los archivos `.java` en el mismo paquete que la aplicación. Igualmente, algunos archivos de salida de ANTLR se guardarán en la carpeta `.antlr`. Para modificar el archivo `settings.json`, se puede acceder de varias formas, pero la más simple es siguiendo estos pasos:

1. Abrir el *Command Palette* con `Ctl+Shift+P`,
2. Buscar la opción *Preferences: Open Settings (JSON)* y seleccionarla (Figura 2),
3. Agregar las siguientes líneas de código

```
"antlr4.generation.mode": "external",
"antlr4.generation.visitors": true
```

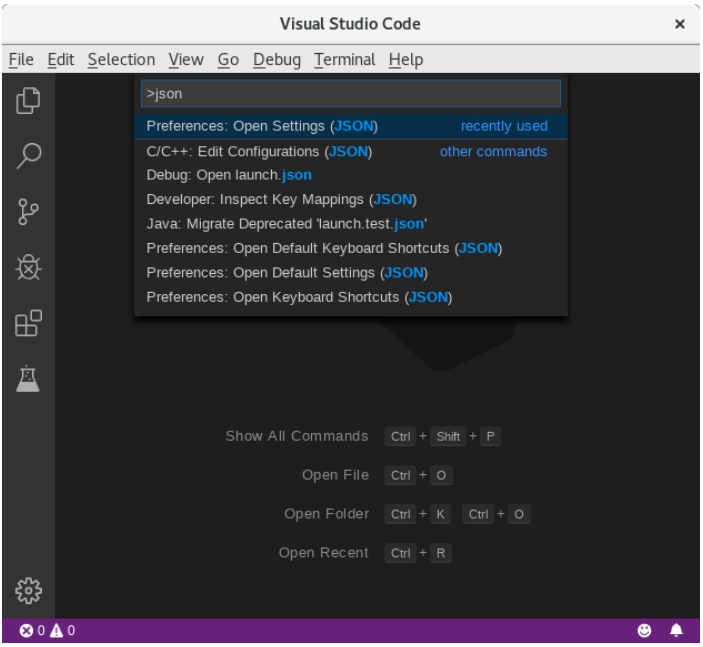

Figura 2: Acceso a la configuración (archivo `settings.json`).

Código 1: Ejemplo de archivo `settings.json`.

```
{
    "window.zoomLevel": 0,
    "editor.wordWrap": "bounded",
    // Generacion archivos ANTLR en el paquete Java
    "antlr4.generation.mode": "external",
    "antlr4.generation.visitors": true
}
```

Código 2: Ejemplo de archivo `.classpath`.

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="con"
      path="org.eclipse.jdt.launching.JRE_CONTAINER/org.
          eclipse.jdt.internal.debug.ui.launcher.
          StandardVMType/JavaSE-1.8"/>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="output" path="bin"/>
    <classpathentry kind="lib"
      path="/usr/share/java/antlr4-runtime.jar"/>
</classpath>
```

Hay que tener en cuenta que la coma es el separador en JSON y no debe faltar. Además, las líneas de código deben estar antes de la llave de cierre como en el ejemplo del Código 1.

Si no incluimos el `.jar` de ANTLR en el proyecto, debemos configurar el proyecto para que se pueda utilizar la biblioteca del sistema. En el archivo `.classpath` debemos agregar la línea

```
<classpathentry kind="lib"
    path="/usr/share/java/antlr-4.x.x-complete.jar" />
```

donde "`/usr/share/java/antlr-4.x.x-complete..jar`" es para Debian 9 y deben reemplazarla por lo que corresponda. El archivo de ejemplo completo puede verse en el Código 2.

4. Primer Proyecto

Ya instalados y configurados los *plug–ins* necesarios, podemos comenzar el primer proyecto.

4.1. Crear Proyecto Java

El primer paso es crear un proyecto Java. Para esto, se puede acceder al *Command Palette* con el atajo `Ctl+Shift+P`, escribir la palabra *project* y elegir la opción "*Java: Create Java Project*" (Fig. 3).

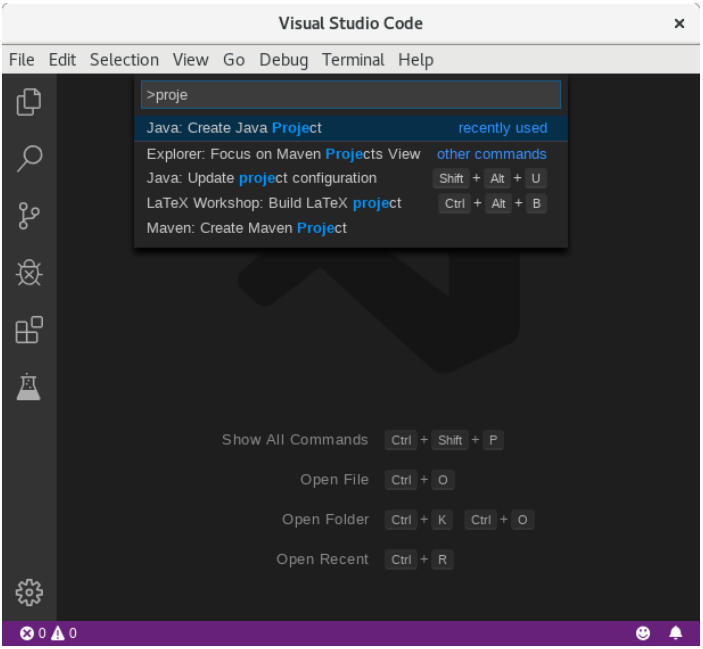

Figura 3: Nuevo proyecto Java.

Luego, elegir la carpeta destino y darle nombre al proyecto. Al finalizar estos pasos, tendremos un proyecto Java con un paquete por defecto denominado `App`. El archivo `app.java` contiene el método `main()` y está listo para compilar y ejecutar.

Recordemos que Visual Studio Code está pensado para desarrollar software, por lo tanto, cuando querramos ejecutar nuestro software vamos a hacerlo sobre el *debugger*. La ejecución se realiza presionando `F5`. La primera ejecución del proyecto necesita que se configuren parámetros de compilación qué, para nuestro caso, será suficiente con las configuraciones por defecto. Al presionar `F5` por primera vez debemos elegir "Java" como tipo de proyecto (Fig. 4). Esta acción crea y abre el archivo de configuración `launch.json` en la carpeta `.vscode` del proyecto. Si presionamos nuevamente `F5` se ejecutará el proyecto y veremos que se abre el panel del *debugger* y se mues-

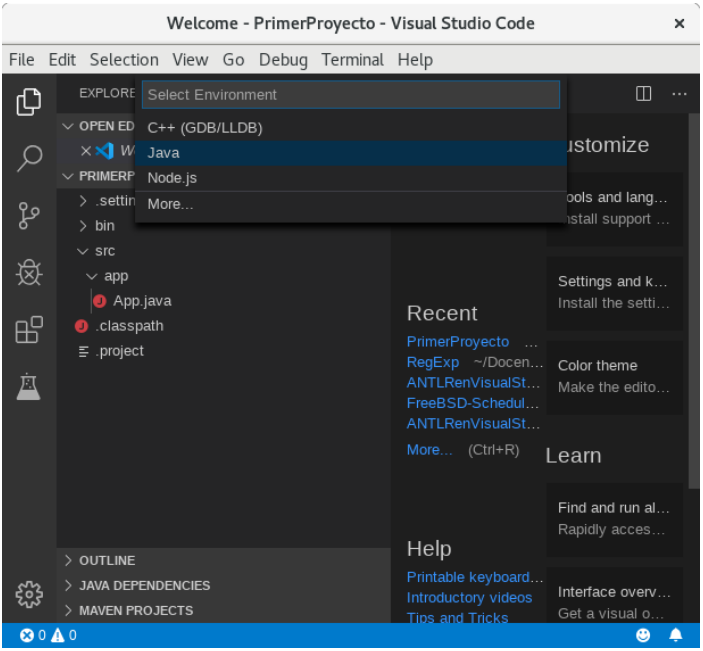

Figura 4: Elegir Java para compilar.

tra la salida por la *debug console* con el texto "*Hello Java*" (Fig. 5). Las próximas veces se ejecutará nuestro programa sin necesidad de realizar configuraciones adicionales.

Finalmente, es necesario modificar el archivo `launch.json` para habilitar la visualización del árbol sintáctico (para más información ver la guía del *plug–in* ANTLR *Debugging ANTLR4 grammars*). Se debe agregar a la lista de opciones la entrada del Código 3.

Para el correcto funcionamiento de ANTLR en nuestro proyecto es necesario ajustar del Código 3 las entradas:

`input` el archivo de entrada a *parsear*,

`startRule` el símbolo inicial,

`grammar` (opcional) el archivo ANTLR con las reglas gramaticales,

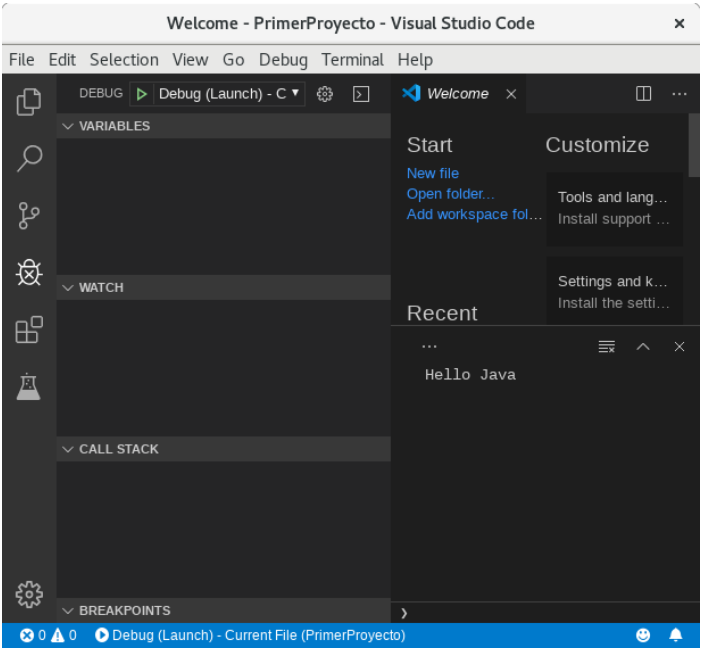

Figura 5: Ejecutar el proyecto Java.

`actionFile` (opcional) ver documentación.

4.2. Crear Proyecto Java con Maven

sitio web de Maven
guía rápida sobre Maven

4.3. Archivo ANTLR

Con el proyecto Java creado y listo para trabajar vamos a crear el archivo para ANTLR. Los archivos de ANTLR llevan extensión `.g` o `.g4`, pero utilizaremos la segunda opción. El atajo de teclado para crear un archivo vacío es `Ctl+n`, que para hacer efectivo el coloreo hay que guardarlo (`Ctl+s`) con la extensión apropiada.

Código 3: Entrada en archivo `settings.json` para ANTLR.

```
{
    "name": "Debug ANTLR4 grammar",
    "type": "antlr-debug",
    "request": "launch",
    // Archivo a parsear
    "input": "src/App/codigo.c",
    // Simbolo inicial
    "startRule": "programa",
    // Muestra Arbol Sintactico
    "printParseTree": true,
    "visualParseTree": true
}
```

ANTLR permite la generación del *lexer* y del *parser*. Por lo tanto, los archivos `.g4` pueden ser para el primero, el segundo o ambos combinados. Nosotros utilizaremos archivos combinados dentro del paquete que contiene el método `main()` para facilitar la ejecución y visualización de resultados. En particular, en el proyecto ejemplo guardaremos el archivo `.g4` en la carpeta `src/app/`.

4.4. Ejemplo Archivo ANTLR

A modo de ejemplo, el Código 4 es un archivo `.g4` que realiza la búsqueda de identificadores tipo Java (nombres de variable o de método). Tanto las reglas léxicas como las gramaticales comienzan con el identificador o etiqueta y terminan en punto y como (`;`). Los dos puntos (`:`) indican el comienzo de la regla y la barra vertical o *pipe* (`|`) separan las distintas reglas alternativas. Las expresiones regulares para la detección de *tokens* se etiquetan con nombres en mayúsculas, como ser `ID` en la línea 10. Las reglas gramaticales se etiquetan con nombres en minúsculas, como ser `s`.

Las palabras reservadas del Código 4 significan lo siguiente:

`grammar` Al comienzo del archivo (línea 1) se indica qué queremos generar, siendo `grammar` la palabra reservada tanto para un *parser* como para un archivo combinado. La otra alternativa sería `lexer`. La palabra `id` es el nombre del *parser*.

Código 4: Ejemplo de archivo `.g4`.

```
1  grammar id;
2
3  @header {
4  package app;
5  }
6
7  fragment LETRA : [A-Za-z] ;
8  fragment DIGITO : [0-9] ;
9
10 ID : (LETRA | '_')(LETRA | DIGITO | '_') ;
11 OTRO : . ;
12
13
14 s : ID { System.out.println("ID"); } s
15   | OTRO { System.out.println("Otro"); } s
16   |
17   ;
```

`@header` En la línea 3 se utiliza el bloque indicado como `@header` para colocar código fuente en Java que necesitamos que aparezca en todos los código fuente generados por ANTLR. En el ejemplo se consigna solamente el `package` al que pertenecen.

`fragment` En las líneas 7 y 8 la palabra `fragment` indica que la expresión regular se utilizará para construir expresiones regulares más complejas, por lo tanto, no se utiliza durante el análisis léxico.

Cada vez que se grabe el archivo en disco, el *plug–in* de ANTLR regenerará todos los archivos.