Бесплатная электронная книга

УЧУСЬ ANTLR

Free unaffiliated eBook created from
Stack Overflow contributors.

#antlr

Около

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

глава 1: Начало работы с ANTLR

замечания

ANTLR (еще один инструмент для распознавания языков) - мощный генератор синтаксического анализатора для чтения, обработки, выполнения или перевода структурированных текстовых или двоичных файлов. Он широко используется для создания языков, инструментов и фреймворков. Из грамматики ANTLR генерирует синтаксический анализатор, который может строить и ходить деревья синтаксического анализа.

- Официальный сайт antlr (всегда указывает на последнюю версию)

Версии Antlr
Antlr разделяется на две большие части, грамматику (файлы грамматики) и сгенерированные файлы кода, которые выводятся из грамматики на основе целевого языка. Версия antlr находится в формате V1.V2.V3:

- V1: изменение в V1 означает, что новый синтаксис функций был введен в грамматических файлах
- V2: изменение в V2 означает, что в сгенерированные файлы были добавлены новые функции или основные исправления (например, добавление новых функций)
- V3: означает исправления ошибок или незначительные улучшения

Цели библиотек времени выполнения и цели генерации кода
Инструмент Antlr написан на Java, однако он способен генерировать парсеры и лексеры на разных языках. Для запуска анализатора и лексера вам также понадобится библиотека времени выполнения antlr наряду с парсером и лексерским кодом. Поддерживаемый целевой язык (и библиотеки времени выполнения):

- Джава
- C #
- Python (2 и 3)
- JavaScript

Версии

Версия	Дата выхода
2,0	1997-05-01

Версия	Дата выхода
3.0	2011-01-19
4,0	2013-01-21
4,1	2013-07-01
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

Привет, мир

Простую мировую грамматику приветствия можно найти здесь :

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

Чтобы создать этот образец .g4, вы можете запустить следующую команду из терминала / командной строки операционной системы:

```
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
```

Построение этого примера должно привести к следующему выводу в каталоге файлов Hello.g4:

1. Hello.tokens
2. HelloBaseListener.java
3. HelloLexer.java
4. HelloLexer.tokens
5. HelloListener.java
6. HelloParser.java

При использовании этих файлов в вашем собственном проекте обязательно включите файл jar ANTLR. Чтобы скомпилировать все эти файлы с помощью Java, в том же рабочем каталоге или по пути выполните следующую команду:

```
javac *.java
```

Прочитайте Начало работы с ANTLR онлайн: https://riptutorial.com/ru/antlr/topic/4453/начало-работы-с-antlr

глава 2: TestRig / grun

Examples

Настройка TestRig

ANTLR содержит средство тестирования в своей библиотеке времени исполнения, этот инструмент можно использовать для отображения информации, детализирующей, как выполняется синтаксический анализ, для соответствия ввода с определенными правилами в вашем файле грамматики.

Чтобы использовать этот инструмент, содержащийся в файле jar ANTLR, вы должны настроить путь к классам систем, чтобы разрешить доступ как к инструменту ANTLR, так и к библиотеке времени выполнения:

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

> Примечание. Убедитесь, что точка предшествует любому пути, чтобы гарантировать, что виртуальная машина Java не увидит классы в вашем текущем рабочем каталоге.

Alises можно использовать в Linux / MAC / Unix для упрощения используемых команд:

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

Установка примечаний по окнам для псевдонимов и настройки класса пути может быть более сложной, см. Здесь более подробную информацию.

Доступ к TestRig

Как только вы настроите свой псевдоним, вы можете настроить TestRig следующим образом, снова используя псевдоним, рекомендуется уменьшить время, необходимое для выполнения действия:

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

Если вы не хотите настраивать псевдоним в окнах, вы можете получить доступ к TestRig, выполнив следующую команду в том же месте, что и каталог 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
```

Чтобы запустить TestRig на вашей грамматике, вы можете передать параметры для своей грамматики следующим образом:

```
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'.
```

Построение грамматики с визуальным деревом анализа

Указание `-gui` командной строки `-gui` при запуске грамматики ANTLR в тестовой установке приведет к появлению окна с визуальным представлением дерева синтаксического анализа. Например:

Учитывая следующую грамматику:

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
   ;
```

Учитывая следующий файл 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
}
```

Синтаксис синтаксиса синтаксиса:

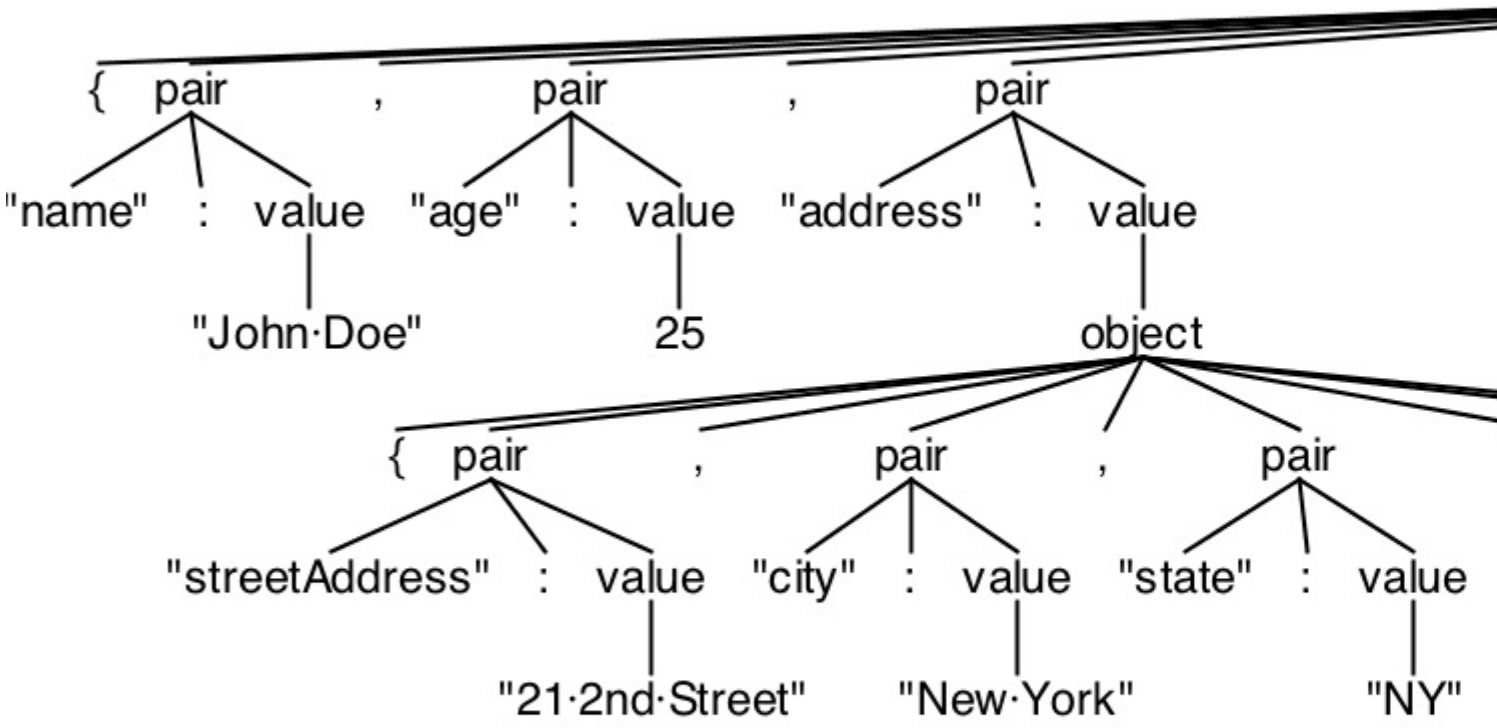
```
export CLASSPATH=".:/usr/local/lib/antlr-4.0-complete.jar:$CLASSPATH"

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
```

приведет к сгенерированным файлам **.java** и **.tokens** , а также скомпилированным файлам **.class** :

```
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
```

и следующее дерево разбора:



Прочитайте TestRig / grun онлайн: https://riptutorial.com/ru/antlr/topic/3270/testrig---grun

глава 3: Введение в ANTLR v3

Examples

Установка и настройка

Как установить ANTLR в Eclipse

(Последнее тестирование на Indigo и ANTLR IDE 2.1.2)

1. Установите Eclipse.
2. Загрузите ANTLR полную бинарную банку, которая включает ANTLR v2. Извлечение в каталог temp. Скопируйте папку antlr-nn в соответствующее постоянное место, например, в ту же папку, в которую установлен Eclipse.
3. Добавьте сайт обновления ANTLR IDE в Eclipse.
    - В Eclipse нажмите «Справка» и выберите «Установить новое программное обеспечение».
    - Нажмите кнопку «Добавить ...».
    - В окне «Добавить репозиторий» введите «Тип местоположения» http://antlrv3ide.sourceforge.net/updates и введите что-то вроде ANTLR IDE для имени и нажмите «OK», чтобы вернуться в окно «Доступное программное обеспечение».
    - Установите флажок ANTLR IDE vn.nn и щелкните по нему до тех пор, пока он не будет установлен. Eclipse, вероятно, перезапустится.
4. Настройте IDE ANTLR.
    - В главном окне Eclipse нажмите «Окно», затем «Настройки».
    - В левой панели разверните ANTLR и выберите «Builder».
    - На правой панели нажмите кнопку «Добавить ...».
    - В окне «Добавить ANTLR Package» выберите «Directory ...» и перейдите к папке antlr-nn и нажмите «OK».
    - Нажмите «OK», чтобы закрыть окно «Добавить пакет ANTLR».
    - Выберите «Генератор кода» в левой панели и нажмите «Отредактировать папку проекта» в правой панели. Введите имя папки. Примеры: antlr-java или antlr-generated.
    - Выберите любые другие параметры конфигурации, но НЕ проверяйте -nfa или -dfa в окне «Общие» в окне «Строительство». Если флажок установлен, это приведет к ошибкам ANTLR, предотвращающим создание Java-файлов в выходной папке.
    - Нажмите «OK», чтобы закрыть окно «Настройки».
5. Создайте новый проект Java и включите поддержку ANTLR.
    - В главном окне Eclipse перейдите в File, New, Java Project. Нажмите «Далее»,

введите название проекта и нажмите «Готово».
    - Чтобы включить поддержку ANTLR для проекта, в окне «Проводник пакетов» (левая панель) щелкните правой кнопкой мыши только что созданный проект и выберите «Настроить», «Преобразовать в проект ANTLR».
    - Добавьте в проект полный файл jar ANTLR: щелкните правой кнопкой мыши проект и выберите «Свойства», «Путь сборки Java», «Добавить внешние JAR ...», перейдите к файлу jar ANTLR, выберите его и нажмите «OK». Нажмите «OK», чтобы закрыть окно «Свойства проекта».
6. Создайте грамматику ANTLR.
    - Создайте новую грамматику ANTLR: щелкните правой кнопкой мыши папку src проекта, затем File, New, Other, разверните ANTLR и выберите Комбинированную грамматику. Нажмите «Далее», введите имя грамматики, выберите «Язык» и нажмите «Готово».
    - Файл «.g» создается с выбранными опциями и пустым правилом. Добавьте язык options = Java, @header, @lexer :: header и @members в верхней части (см. Пример). Автоматическое завершение - это самый простой способ добавить их (нажмите CTRL-пространство, чтобы открыть список автозавершения).
7. Сохраните грамматику.
    - При сохранении в Project Explorer должна появиться папка, содержащая сгенерированный Java-код для грамматики. Если это не так, убедитесь, что параметры -nfa или -dfa не отмечены в настройках ANTLR в разделе «Общие» в окне «Строительство» (шаг 4g). [Подтвердите, если это необходимо: проверьте, что переменная окружения CLASSPATH указывает на Java7, которая соответствует вашей установке Eclipse (32 или 64 бита), а переменная среды Windows Path имеет Java7 SDK.]
    - Чтобы избежать «не может быть разрешено для типа» ошибок Java, щелкните правой кнопкой мыши папку, содержащую сгенерированный код Java, а затем «Путь сборки», «Использовать в качестве исходной папки».

ОБРАЗЦОВЫЙ КОМБИНИРОВАННЫЙ ГРАММАТ

```
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';
```

Прочитайте Введение в ANTLR v3 онлайн: https://riptutorial.com/ru/antlr/topic/6629/введение-в-antlr-v3

глава 4: Введение в ANTLR v4

замечания

ANTLR v4 - мощный инструмент, используемый для создания новых языков программирования и обработки / перевода структурированных текстовых или двоичных файлов. ANTLR использует созданную вами грамматику для генерации парсера, который может создавать и перемещать дерево разбора (или абстрактное синтаксическое дерево, AST). Парсер состоит из выходных файлов на указанном вами целевом языке. ANTLR v4 поддерживает несколько целей, включая: Java, C #, JavaScript, Python2 и Python3. Работа над C ++ работает. Для работы с IDE в GUI существуют плагины для Visual Studio, Intellij, NetBeans и Eclipse.

Для получения общей информации посетите веб-сайт ANTLR . Чтобы серьезно относиться к ANTLR, ознакомьтесь с очень рекомендуемой книгой, написанной Терренсом Парром (парнем, создавшим ANTLR) . Окончательный ANTLR 4 Reference .

Значительная информация о версии

- 4.5: 01/22/15 - Добавлен целевой объект JavaScript и обновлена цель C #. 4.5 Примечания к выпуску
- 4.4: 07/16/14 - Добавлены Python2 и Python3 в качестве целей. 4.4 Примечания к выпуску
- 4.3: 06/18/14 - исправлены основные ошибки; подготовленный для добавления новых целей. 4.3 Примечания к выпуску
- 4.2: 02/04/14 - Улучшен синтаксис для выбора / соответствия деревьев разбора. 4.2 Примечания к выпуску
- 4.1: 06/30/13 - улучшенная производительность синтаксического анализа; экспортировать ACT в PNG. 4.1 Примечания к выпуску
- 4.0: 01/21/13 - Первоначальный выпуск.

Examples

Установка для использования в командной строке

ANTLR распространяется как файл Java Jar. Его можно скачать здесь . Поскольку ANTLR скомпилирован как файл jar, он впоследствии требует, чтобы среда выполнения Java работала, если у вас ее нет. Ее можно скачать здесь .

После загрузки файла ANTLR JAR вы можете запустить ANTLR из командной строки так же, как и любой другой JAR-файл:

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

(Предположим, что вы работаете в том же каталоге, что и файл antlr-4.5.3-complete.jar).

Это должно вывести что-то похожее на это:

```
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
```

другие рекомендуемые действия для настройки:

```
     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 %*
```

После настройки вы можете создать приложение, используя ваш грамматический файл . g4:

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

Вы также можете создать приложение на других языках с параметром -Dlanguage. Например, чтобы сгенерировать файлы C #, вы должны сделать что-то вроде этого:

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

См. Здесь полный список подготовленных грамматик для общих языков программирования.

Установка с использованием средств автоматизации сборки

Загрузите последнюю версию ANTLR и извлеките ее в папку.

Вы можете использовать также Maven, Gradle или другой инструмент построения, чтобы зависеть от его среды выполнения (классы, которые используют сгенерированные грамматики): `org.antlr:antlr4-runtime` .

Чтобы автоматически - часть процесса сборки - генерировать парсер в проекте maven, используйте плагин Maven : `org.antlr:antlr4` .

Установить в Eclipse и создать Hello World

(Протестировано с ANTLR 4.5.3, Eclipse Neon, ANTLR 4 IDE 0.3.5 и Java 1.8)

1. Загрузите последний ANTLR . Удостоверьтесь, что вы получили полную версию бинарных файлов ANTLR Java. Сохраните в любом подходящем месте, например папку, в которой хранятся другие библиотеки Java. Неважно, где, просто помните место.

2. Установите ANTLR IDE в Eclipse.

    - В меню Eclipse нажмите «Справка» и выберите «Затмение».
    - В поле «Найти» введите antlr и нажмите «Перейти».
    - Нажмите «Установить» для IDE ANTLR 4.
    - Нажмите «Готово» в окне «Подтвердить выбранные функции».
    - Если появится окно предупреждения безопасности, нажмите «OK».
    - Перезапустить Eclipse.

3. Работайте над ошибкой «Не удалось создать инжектор ...».

    - При доступе к ANTLR 4 Preferences в Eclipse или когда не задана переменная HOME HOME, возникает следующая ошибка: Не удалось создать инжектор для com.github.jknack.antlr-4ide.Antlr4 для com.github.jknack.antlr-4ide.Antlr4 ,
    - Убедитесь, что установлен параметр HOME. Если нет, установите его в соответствии с вашей системой.
    - Загрузите Xtext 2.7.3 в том же месте, что и antlr-nnn-complete.jar.
    - В Eclipse нажмите «Справка» и выберите «Установить новое программное обеспечение».
    - Нажмите «Добавить ...», чтобы перейти в окно «Добавить репозиторий».
    - Введите имя, xtext 2.7.3, а затем нажмите «Архив ...», перейдите к файлу Xtext 2.7.3 и выберите его, затем нажмите «OK».
    - В окне «Установка» нажмите кнопку «Выбрать все», затем нажмите «Далее»> дважды, примите лицензионное соглашение. и нажмите «Готово».
    - Перезапустить Eclipse.

4. Скажите Eclipse / Java, где ANTLR.

    - В Eclipse нажмите «Окно» и выберите «Настройки».

- На левой панели разверните узел Java и Build Path, затем выберите переменные класса.
- В правой панели нажмите «Создать ...», введите имя и нажмите «Файл ...» и перейдите к своему местоположению antlr-nnn-complete.jar. Нажмите «OK», чтобы вернуться в окно «Переменные переменных класса».
- Нажмите «OK», чтобы выйти из «Настройки».

5. (Необязательно) Настройте каталог сгенерированных источников ANTLR IDE.

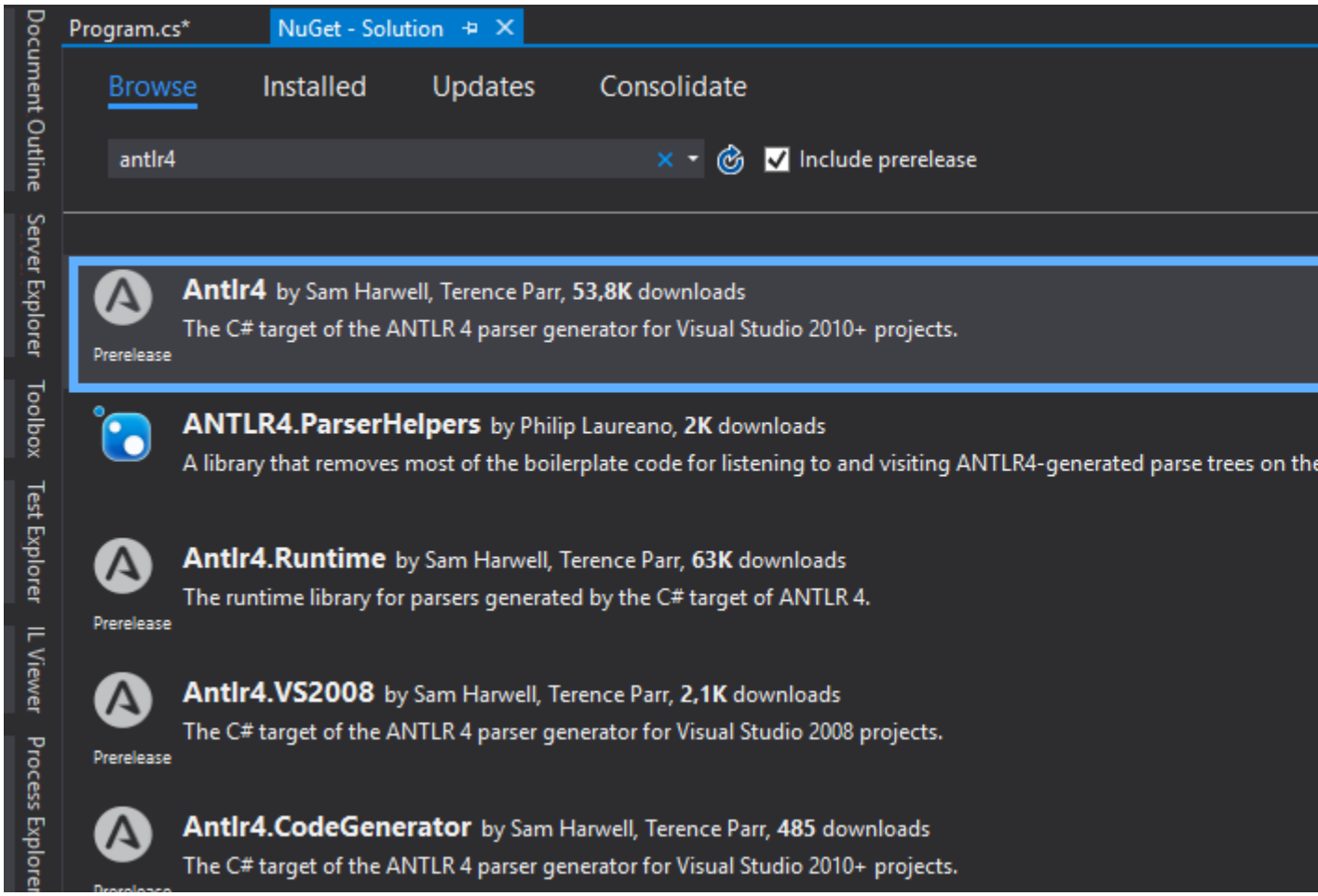
   - В главном окне Eclipse нажмите «Окно», затем «Настройки».
   - На левой панели разверните ANTLR 4 и выберите «Инструмент».
   - В разделе «Параметры» при необходимости измените каталог. Например, java - это мой целевой язык, поэтому я использую ./antlr-java.
   - Нажмите «OK», чтобы закрыть окно «Настройки».

6. Создайте проект ANTLR 4.

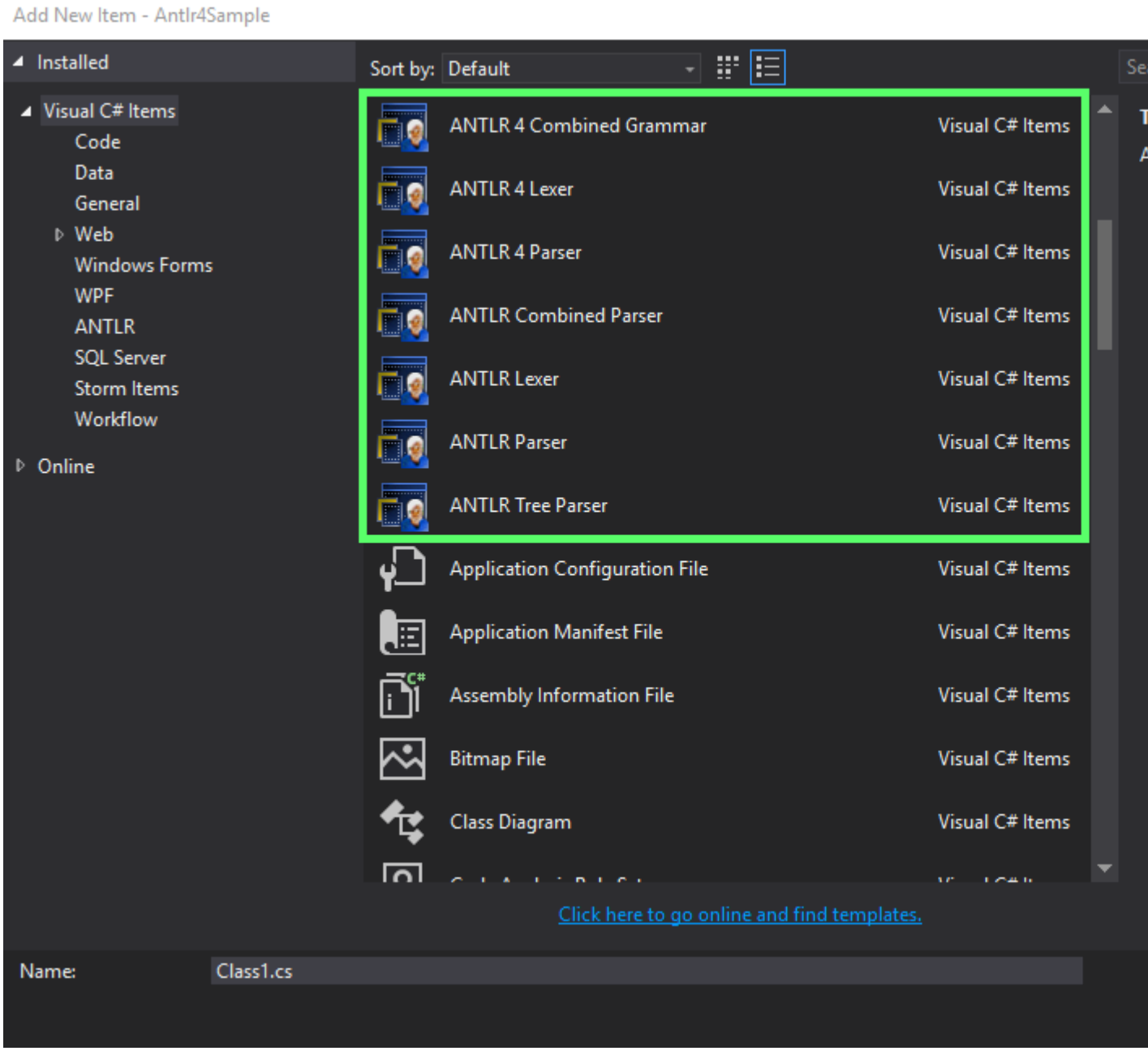
   - В главном окне Eclipse откройте «Файл», «Создать», «Проект».
   - В окне «Новый проект» разверните «Общие» и выберите «Проект ANTLR 4».
   - Нажмите «Далее», введите название проекта и нажмите «Готово».
   - Новый проект по умолчанию содержит файл Hello.g4 и автоматически создаст стандартную программу Hello World.
   - В Проводнике пакетов разверните новую папку проекта, чтобы увидеть файл g4 и папку с именем target (или имя, которое вы дали на шаге 5), содержащее целевые исходные файлы.

Установка ANTLR в Visual Studio 2015 (с использованием Nuget)

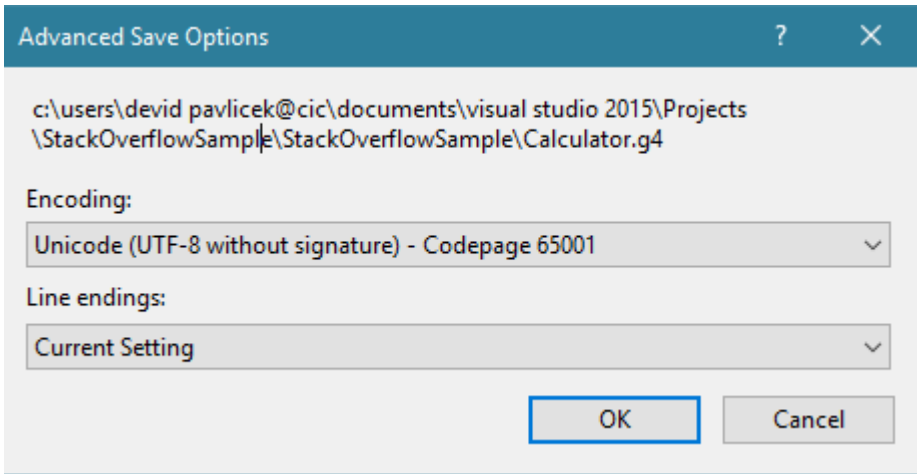
1. Откройте Visual Studio 2015, перейдите в Инструменты → Расширения → Онлайн и найдите Antlr. Загрузите расширение ANTLR Language Support (Создано Sam Harwell) и перезапустите Visual Studio.
2. Создайте новый проект консольного приложения. Щелкните правой кнопкой мыши на Solution → Manage Nuget Packages for Solution → Browse (Tab) и найдите Antlr4 и установите его.



3. Добавьте новый элемент в свой проект, щелкнув его правой кнопкой мыши. Ищите шаблоны ANTLR4.

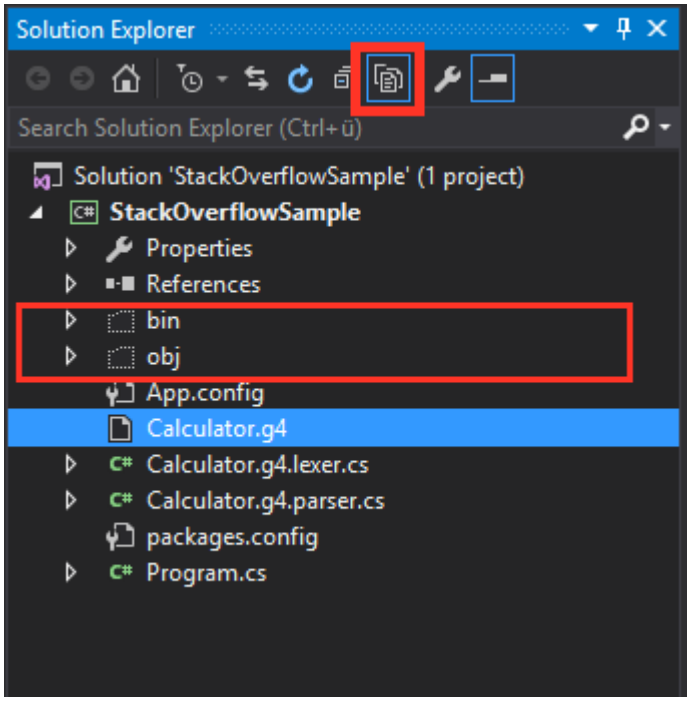


4. Из файла ANTLR (окончание .g4) перейдите в меню «Файл → Предварительное сохранение параметров» и выполните поиск Unicode (UTF-8 **без подписи**) - Codepage 65001 и нажмите «OK». Это оно.

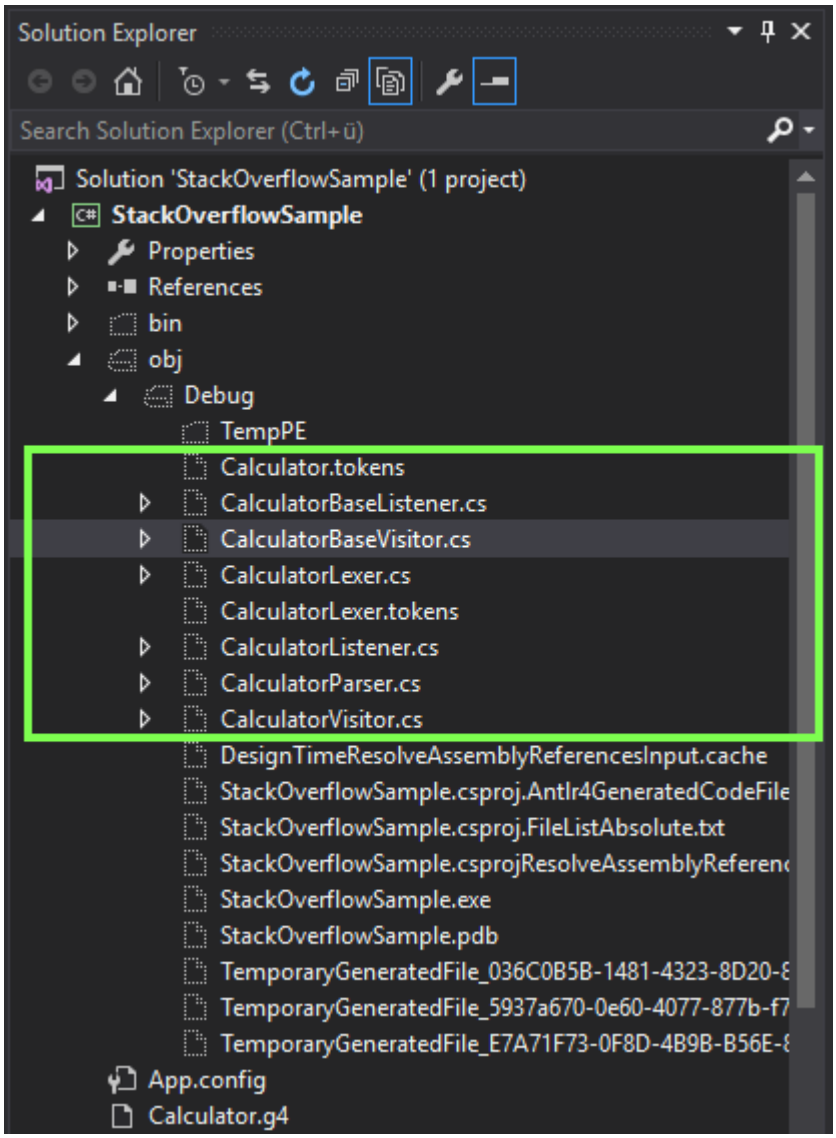


Если все работает

- Создайте элемент комбинированной грамматики ANTLR 4 и назовите его Calculator.g4
- Скопируйте и вставьте исходный код калькулятора из этого проекта Github здесь: Калькулятор от Tom Everett
- Сменить грамматический калькулятор на грамматический калькулятор
- В обозревателе решений → Нажмите Показать все файлы.



- Сохранить и запустить (запустить) проект
- В обозревателе решений в папке obj вы должны увидеть классы cs, созданные как посетитель и слушатель. Если это так, вам это удалось. Теперь вы можете начать работу с ANTLR в Visual Studio 2015.



Прочитайте Введение в ANTLR v4 онлайн: https://riptutorial.com/ru/antlr/topic/2856/введение-в-antlr-v4

глава 5: Посетители

Вступление

В чем разница между слушателем и посетителем? Разница между механизмами слушателя и посетителя - это методы слушателя, вызываемые объектом walker, предоставленным ANTLR, тогда как методы посетителей должны перемещать своих детей с явным вызовом посещения. Забытие вызова функции посещения () для дочерних узлов узла означает, что эти поддеревья не посещаются. У посетителя есть возможность ходить по деревьям, а в слушателе вы реагируете только на пешеход.

Examples

пример

Пример грамматики (Expr.g4)

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

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

Создание посетителя

Чтобы создать посетителя или отключить посетителя для вашей грамматики, вы используете следующие флаги:

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

Команда командной строки / терминала для создания вашей грамматики с посетителем будет отформатирована, как показано ниже, в отношении выбранного флага и возможных псевдонимов:

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

Результатом будет синтаксический анализатор / лексер с посетителем или посетителем, соответственно.

Результат. В этом примере вывод будет **ExprBaseVisitor.java** и **ExprVisitor.java** . Это соответствующие java-файлы для реализации функций посетителя. Часто идеально подходит для создания нового класса и расширения ExprBaseVisitor для реализации новых функций посетителя для каждого метода.

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

Прочитайте Посетители онлайн: https://riptutorial.com/ru/antlr/topic/8211/посетители

глава 6: Правила Lexer в v4

Examples

Простые правила

Правила Lexer определяют типы токенов. Их имя должно начинаться с буквы верхнего регистра, чтобы отличать их от правил парсера.

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

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

Основной синтаксис:

Синтаксис	Имея в виду
`A`	Правило совпадения lexer или фрагмент с именем `A`
`AB`	Матч `A` за которым следует `B`
`(A|B)`	Сопоставьте либо `A` либо `B`
`'text'`	Соответствовать литералу *"text"*
`A?`	Матч `A` ноль или один раз
`A*`	Матч `A` ноль или более раз
`A+`	Матч `A` или несколько раз
`[A-Z0-9]`	Сопоставьте один символ в определенных диапазонах (в этом примере между AZ или 0-9)
`'a'..'z'`	Альтернативный синтаксис для диапазона символов
`~[AZ]`	Отрицание диапазона - соответствует любому одиночному символу *не* в диапазоне
`.`	Совпадение любого персонажа

Фрагменты

Фрагменты являются многократными частями правил лексера, которые не могут совпадать сами по себе - на них нужно ссылаться из правила лексера.

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

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

Неявные правила лексера

Когда жетоны, подобные `'{'` , используются в правиле *парсера* , для них создается неявное правило лексера, если не существует явного правила.

Другими словами, если у вас есть правило lexer:

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

Тогда оба этих правила синтаксического анализа эквивалентны:

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

Но если `OPEN_BRACE` lexer `OPEN_BRACE` *не* определено, будет создано неявное анонимное правило. В этом случае неявное правило будет определено так, *как если бы* оно было определено *перед* другими правилами: оно будет иметь более высокий приоритет, чем другие правила.

Приоритетные правила

Несколько правил lexer могут соответствовать одному и тому же входному тексту. В этом случае тип токена будет выбран следующим образом:

- Сначала выберите правило lexer, которое соответствует *самому длинному* входу
- Если текст соответствует неявно определенному токену (например, `'{'`), используйте неявное правило
- Если несколько правил lexer совпадают с одной и той же длиной ввода, выберите *первую* , основанную на порядке определения

Следующая комбинированная грамматика:

```
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;
```

Учитывая следующий ввод:

```
aaa foo bar baz barz
```

Выполним следующую лексерную последовательность из лексера:

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

- `aaa` имеет тип `IDENTIFIER`

  Только правило `IDENTIFIER` может соответствовать этому знаку, нет никакой двусмысленности.

- `foo` имеет тип `'foo'`

  Правило анализатора `randomParserRule` вводит неявный тип токена `'foo'` , который является приоритетным по правилу `IDENTIFIER` .

- `bar` имеет тип `BAR`

  Этот текст соответствует правилу `BAR` , которое определено *до* правила `IDENTIFIER` , и поэтому имеет приоритет.

- `baz` имеет тип `IDENTIFIER`

  Этот текст соответствует правилу `BAZ` , но также соответствует правилу `IDENTIFIER` . Последний выбирается так, как он определен *до* `BAR` .

  Учитывая грамматику, `BAZ` *никогда* не сможет сравниться, так как правило `IDENTIFIER` уже охватывает все, что может соответствовать `BAZ` .

- `barz` имеет тип `IDENTIFIER`

  Правило `BAR` может соответствовать первым 3 символам этой строки (`bar`), но правило `IDENTIFIER` будет соответствовать 4 символам. Поскольку `IDENTIFIER` соответствует более длинной подстроке, он выбирается над `BAR` .

Как правило, определенные правила должны быть определены *перед* более общими правилами. Если правило может соответствовать только входному сигналу, который уже покрыт ранее определенным правилом, он *никогда не* будет использоваться.

Неявно определенные правила, такие как `'foo'` действуют так, как если бы они были определены *перед* всеми другими правилами lexer.

Команды Lexer

Правило lexer может иметь связанные *команды* :

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

Команды определяются после a `->` в конце правила.

- `skip` : `skip` согласованный текст, не будет выдан токен
- `channel(n)` : выдает токен на другом канале
- `type(n)` : Изменяет тип испущенного маркера
- `mode(n)` , `pushMode(n)` , `popMode` , `more` : управляет лексерскими режимами

Действия и семантические предикаты

Действие lexer - это блок произвольного кода на целевом языке, окруженный `{ ... }` , который выполняется во время сопоставления:

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

Семантический предикат - это блок произвольного кода на целевом языке, окруженный `{ ... }?` , который вычисляет логическое значение. Если возвращаемое значение равно false, правило lexer пропускается.

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

Семантические предикаты должны быть определены в конце правила, когда это возможно по соображениям производительности.

Прочитайте Правила Lexer в v4 онлайн: https://riptutorial.com/ru/antlr/topic/3271/правила-lexer-в-v4

глава 7: Слушатели

Examples

События прослушивателя с помощью ярлыков

Маркировка альтернатив внутри правила, начинающегося с оператора `#` указывает ANTLR генерировать методы прослушивания для каждой метки, соответствующей альтернативе.

Указав метку для каждой альтернативы в следующем правиле:

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

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

Будет генерировать следующие методы в сгенерированном интерфейсе, который расширяет `ParseTreeListener` :

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

Прочитайте Слушатели онлайн: https://riptutorial.com/ru/antlr/topic/6717/слушатели

глава 8: Цели ANTLR / Язык Runtimes

Examples

Поддержка языков

ANTLR способен генерировать парсеры для ряда языков программирования:

1. Цель C #
2. Python Target
3. Цель JavaScript
4. Java-цель

По умолчанию ANTLR генерирует парсер из командной строки на языке программирования Java:

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

Чтобы изменить целевой язык, вы можете запустить следующую команду из терминала / командной строки OS:

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

Вместо того, чтобы каждый раз использовать параметр «-Dlanguage» в командной строке / терминале для создания нужного парсера для определенного языка, вы можете выбрать цель из вашего файла грамматики .g4, включив цель в глобальный раздел:

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

Чтобы использовать сгенерированный вывод синтаксического анализатора, убедитесь, что время выполнения ANTLR для указанного языка:

1. Время выполнения CSharp
2. Время выполнения Python 2
3. Время выполнения python 3

Полные инструкции и информация о библиотеках времени выполнения ANTLR

Настройка парсера Python

После запуска вашего файла грамматики .g4 с помощью ANTLR.jar вам необходимо создать несколько файлов, например:

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

Чтобы использовать их в проекте python, используйте среду исполнения Python в рабочей области, чтобы любое приложение, которое вы разрабатываете, может получить доступ к библиотеке ANTLR. Это можно сделать, извлекая среду выполнения в текущую папку проекта или импортируя ее в свою среду IDE в зависимости от проекта.

```
#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)
```

Эта настройка включает в себя ваш синтаксический анализатор и принимает входные данные из командной строки, чтобы разрешить обработку файла, переданного в качестве параметра.

```
#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')
```

Эти файлы в сочетании с временем выполнения ANTLR и вашими файлами, созданными из вашего файла грамматики, будут принимать одно имя файла в качестве аргумента и читать и анализировать ваши правила грамматики.

Чтобы расширить базовую функциональность, вы также должны расширять приемник по умолчанию для обработки соответствующих событий для токенов, встречающихся во время выполнения.

Прочитайте Цели ANTLR / Язык Runtimes онлайн: https://riptutorial.com/ru/antlr/topic/3414/цели-antlr---язык-runtimes

кредиты

S. No	Главы	Contributors
1	Начало работы с ANTLR	Athafoud, cb4, Community, D3181, Gábor Bakos, KvanTTT
2	TestRig / grun	bn., D3181, Lucas Trzesniewski, Pascal Le Merrer
3	Введение в ANTLR v3	Athafoud, cb4
4	Введение в ANTLR v4	Athafoud, cb4, Community, D3181, Devid, Gábor Bakos, GRosenberg, Lucas Trzesniewski
5	Посетители	D3181
6	Правила Lexer в v4	Athafoud, bn., Loxley, Lucas Trzesniewski
7	Слушатели	bn., Lucas Trzesniewski
8	Цели ANTLR / Язык Runtimes	D3181