ANTLR4

Ultima atualização a 12 de maio de 2020

O seu nome significa **ANother Tool for Language Recognition** e é um gerador de processadores de linguagens.

> Processadores são ferramentas responsáveis por ler, processar ou traduzir linguagens.

Comandos da bash

Para executar o ANTLR4 existem vários comandos da bash, que são descritos abaixo com as suas funcionalidades.

Para facilitar a sua execução, criei ainda alguns alias (instruções aqui).

```
# Comando           # Alias              # Descrição
antlr4              a4                   # Compilação de gramáticas
antrl4-test         a4test               # Depuração de gramáticas
antlr4-clean        a4clean              # Eliminação dos ficheiros gerados
antlr4-main         a4main               # Gerar classe main para a gramática
antlr4-visitor      a4visit              # Gerar classe visitor para a
gramática
antlr4-listener     a4listen             # Gerar classe listener para a
gramática
antlr4-build        a4build              # Compilar gramáticas e código Java
gerado
antlr4-run          a4run                # Executar classe <gramática>Main
antlr4-jar-run      a4jar                # Executar um ficheiro .jar
antlr4-javac        a4javac              # Compilador Java
antlr4-java         a4java               # Máquina virtual Java
antlr4-java-clean   a4java-clean         # Eliminar ficheiros binários java
antlr4-view-javadoc a4view-javadoc       # Documentação Java (no browser,
local)
```

Gramáticas

Para definir uma gramática em ANTLR4 deve ser criado um ficheiro com extensão **g4**, ao qual deve anteceder o nome da gramática.

> `<grammarName>.g4`

As instruções são definidas uma em cada linha, e **terminam com ponto e vírgula**!

Para definir o nome da gramática utiliza-se a seguinte instr

> `grammar <grammarName>;`

As regras seguem a estrutura abaixo, sendo o α **minúsculo para sintáticas** e **maiúsculo para léxicas**. β é uma expressão simbólica que equivale a α, que pode ser um token, uma regra (sintáica ou léxica), entre outros.

> `α:β;`

Na definição de ***tokens*** é fundamental que as **strings** sejam delimitadas por **pelicas**!

Compilação

```
$ antlr4 <grammarName>.g4
$ antrl4-build <grammarName>
# = $ antlr4-javac *.java
```

Gera os seguintes ficheiros:

> `<grammarName>Lexer.java` , que gera *tokens* para a árvore sintática
>
> `<grammarName>Parser.java`, que gera a árvore sintática
>
> `<grammarName>Listener.java` e `<grammarName>BaseListener.java`, responsáveis pela execução do listener
>
> `<grammarName>.tokens` e `<grammarName>Lexer.tokens`, que indentificam os *tokens* (pouco importante)

Testes

```
$ antlr4-test <grammarName> <sintaticMainRuleName>
# Caso hajam erros na validação serão mostrados, caso contrário não acontece
nada
# No news is good news!

# Options
-tokens     # Mostra tokens gerados
-gui        # Mostra árvore sintática (ferramenta visual)
```

Depois de executar o comando, escrever o código a validar e pressionar CTRL+D.

Em alternativa, pode ser injetado código, antecedendo `echo "<codeToValidate>" |` ao comando acima.

Main

Para gerar a classe **main**.

```
$ antlr4-main <grammarName> <sintaticMainRuleName>
$ antlr4-build <grammarName>
```

E para a executar.

```
$ antlr4-run
# = $ java -ea <grammarName>.java <grammarName>Lexer.tokens
```

Para evitar a injeção de código ou a necessidade de pressionar CTRL+D, podemos criar uma classe main **interativa** com a opção `-i`.

```
$ antlr4-main -i <grammarName> <sintaticMainRuleName>
```

> Se *main* não for interativo, primeiro é executado o *visitor* para todas as linhas até ao EOF e só depois o *listener*. Caso contrário, são os dois executados para cada linha, uma linha de cada vez.

Se por acaso quisermos **mudar o nome da main rule**, não necessitamos de criar uma nova classe. Basta mudar na instrução `ParseTree tree = parser.<oldRule>()` mudar a `<oldRule>` para `<newRule>`.

Atributos, ações e argumentos

É possível adicionar **atributos** e **ações** (entre chavetas) às regras.

```
assign: ID '=' e=expr ';'
    { System.out.println($ID.text + '=' + $e.v); }
    ;
expr returns[int v]: INT
    { $v = Integer.parseInteger($INT.text); }
    ;
```

Também podem ser passados **atributos como argumento**.

```
assign: ID '=' e=expr[true] ';'
    { System.out.println($ID.text + '=' + $e.v); }
    ;
expr[boolean inAssing] returns[int v]: INT
    {
        if ($inAssign)
            System.out.println("In assign!");
        $v = Integer.parseInteger($INT.text);
        // $INT.text = $INT.getText()
    }
    ;
```

Todos estes são injetados na classe `<rule>Context` no ficheiro `<grammarName>Parser.java`.

- Os atributos como atributos públicos da classe, sendo dados do tipo **Token**
- Os argumentos como atributos públicos inicializados no construtor da classe
- Os valores de retorno também como atributos público

As ações podem também ser adicionadas no preâmbulo da gramática.

```
grammar <name>;
@header {
    // Código injetado no início dos ficheiros (parser e lexer)
    import java.util.*;
}
@members {
    // Acrescentado às classes do analisador sintático/léxico
    // Por isso, acessível em todos os seus métodos
    List<String> varTable = new ArrayList<>();
}
```

Pode haver ser restringido o analisador ao qual estas são adicionadas:
`@lexer/parser:header/members`

Contexto automático

Para facilitar a análise semântica, o ANTRL4 gera classes com o contexto de todas as regras da gramática, como é o caso dos *listeners* e dos *visitors*.

> A principal distinção entre os dois é que os métodos do ***listener*** são invocados pelo *walker* do ANTLR4, enquanto que os do ***visitor*** devem fazer chamadas explícitas aos dos seus filhos, sob pena de a sub-árvore do método que não invoca os filhos não ser percorrida.
>
> *Fonte: stack overflow*

Adicionar visits/callbacks

Na mesma regra sintática podemos adicionar *visits/callbaks* a diferentes alternativas. Para isso, basta adicionar uma **label** a cada uma.

```
r: a    #altA
 | b    #altB
```

Assim, quer nos *visitors*, quer nos *listeners*, serão criados *visits* e *callbacks* para as alternativas apresentadas, deixando de aparecer a regra em si.

No caso representado acima, no *visitor*, o método `visitR` vai dar lugar aos métodos `visitAltA` e `visitAltB`.

Notas adicionais

Correr gramáticas com analisadores separados

Quando os analisadores léxico e sintático são definidos em ficheiros separados a sua compilação e execução é ligeiramente diferente.

> A separação entre os analisadores potencia a reutilização da gramática. Ver *slides* teóricos para perceber como esta separação deve ser feita.

Para compilar a gramática.

```
antlr4 <grammarName>Lexer.g4 #Sempre primeiro que o Parser!
antlr4 <grammarName>Parser.g4
antlr4-build
antlr4-test <grammarName> <mainRule>
```

Para gerar o main

```
antlr4 <grammarName>Lexer.g4
antlr4 <grammarName>Parser.g4
antlr4-main -f <grammarName> compilationUnit [-l <listener> -v <visitor>]
```

Atenção! Tanto os *visitors* como os *listeners* devem ser gerados sobre o analisador sintático apenas!

Input do utilizador

Se quisermos pedir ao utilizador algum tipo de *input* num *listener* ou *visitor*, não podemos simplesmente utilizar um *Scanner(System.in)*, pois a entrada do *antlr4-run* vai entrar em colisão com a do *Scanner*, uma vez que se geram dois acesso independentes ao *System.in*.

Há duas formas de resolver este problema:

- Alterar a fonte do *input stream* do *Scanner* para o ficheiro **dev/tty**, que corresponde ao terminal.

```
Scanner sc = null;
try {
    sc = new Scanner(new File("dev/tty"));
} catch (FileNotFoundException e) {
    System.err.println("Could not read from terminal!");
    System.exit(0);
}
```

- Em alternativa ao *antlr4-run* e injetarmos na sua execução o conteúdo do ficheiro, podemos passá-lo como argumento da execução.

```
# Em vez de...
cat p1.txt | antlr4-run <grammarName>
# = cat p1.txt | java -ea <grammarName>Main.java
# Executar...
java -ea <grammarName>Main.java p1.txt
```

É no entanto necessário que o *main* processe os ficheiros que recebe como argumento.

```java
// Hipótese 1
public static void main(InputStream in) {
    Scanner sc = new Scanner(in);
    ...
}
// Hipótese 2
public static void main(String[] args) {
    for (String s: args){
        process(new FileInputStream(s));
    }
    ...
}
```