ANTLR4 Visitors

Ultima atualização a 23 de abril de 2020

Conteúdo

Criar_visitor

Para criar um *visitor* devem ser executados os seguintes comandos:

```
$ antlr4 <grammarName>.g4 -visitor
# Cria <grammarName>Visitor.java e <grammarName>BaseVisitor.java
$ antlr4-visitor <grammarName> <visitorName> <dataType>
# Copia <grammarName>BaseVisitor.java, alterando o tipo de dados genérico para
passado como argumento, para o ficheiro <visitorName>.java
# Em alternativa, podemos ser nós a copiá-lo e a fazer as alterações
```

Criar_main_com_visitor

Para criar **main** com o visitor associado, devem ser executados os seguintes comandos:

```
$ antlr4-main -i <grammarName> <sintaticMainRuleName> -visitor <visitorName>
$ antlr4-build <grammarName>
```

Contexto

No *visitor*, para cada regra sintática `r: a b;`, é criado um método `visitR`, que tem como argumento o seu contexto, definido no ficheiro `<grammarName>Parser.java` pela classe `RContext`.

> Os objetos de contexto têm a si associada toda a informação revelante da análise sintática (*tokens*, referência aos nós filhos, etc.).

Caso a e b sejam não terminais, a classe de contexto de r contém métodos para devolver os contextos destes dois, respetivamente `AContext` e `BContext`, acessíveis através dos métodos `a()` e `b()`: `ctx.a()` e `ctx.b()`.

Para obter o retorno dos seus *visitors* deve recorrer-se à função `visit()`, passando como argumento o contexto do elemento.

Caso algum destes elementos seja terminal, este é acessível através do mesmo método do seu contexto, mas que agora retorna um objeto da classe **TerminalNode**.

> **TerminalNode** (*org.antlr.v4.runtime.tree.TerminalNode*)
>
> String .getText() Devolve texto combinado de todas as folhas

Identificadores_repeditos

Quando uma regra sintática tem um identificador repetido, ou que se pode repetir (quantificadores + ou *), o método que devolvia o seu contexto em circuntâncias normais vai agora devolver um objeto do tipo List, tendo como argumentos *TerminalNode* ou a respetiva classe do contexto.

> **List** (*java.util.List*)
>
> int .size() Devolve o número de elementos da lista
>
> E .get() Devolve o elemento na posição dada

Em alternativa à operação `ctx.TOKEN().get(<index>)` pode ser realizada a operação `ctx.TOKEN(<index)`.

Para manipular o valor retornado, podemos recorrer à classe **Iterator**.

```
import java.util.Iterator;
import org.antlr.v4.runtime.tree.TerminalNode;
...
Iterator<TerminalNode> iter = ctx.TOKEN().iterator();
while (iter.hasNext())
    iter.next();
```

Adicionar_labels

Por vezes assumimos que numa regra um token pode assumir vários valores. Para facilitar o seu acesso, podemos adicionar-lhes um *label*.

```
# Grammar
<sintaticRule>: ... op=('+'|'-') ...
# Acima foi criado o label op

# Visitor
ctx.op.getText()
```

Os *labels* não são **acedidos** como métodos, mas sim **como atributos**, sendo devolvido um objeto do tipo **Token**.

> Token (*org.antlr.v4.runtime.Token*)
>
> String .getText() Devolve o texto do *token*

De forma semelhante aos identificadores repetidos, se existir mais do que um *token* na regra sintática, é devolvido um objeto do tipo **List**, uma coleção de elementos da classe referida.

Percorrer_a_árvore_sintática

Para percorrer a árvore sintática com *visitors* é necessário fazer **chamadas explícitas à visita** do contexto em cada nó para o seu filho. Por defeito, a visita é feita em profundida e percorre todos os nós.

Ao criarmos *visitors*, no entanto, os seus métodos vão fazer *override* aos métodos por defeito e esta ordem pode ser alterada, para além de a visita a determinadas partes da árvore deixar de ser feita.

Para visitar o contexto de um identificador, devemos recorrer ao método *visit*.

```
visit(ctx.identifier())
```

Os métodos que não são alterados podem ser apagados, pois estes já estão implementados por defeito.