ANTLR4 Listeners

Ultima atualização a 04 de maio de 2020

Um *listener* consiste num conjunto de métodos *void* invocados por um *walker* de forma sequencial (que não pode ser alterada).

Implementa o padrão ***callback***.

Conteúdo

Criar_listener

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

```
$ antlr4 <grammarName>.g4
# Ao compilar a gramática são gerados automaticamente os ficheiros
<grammarName>Listener.java e <grammarName>BaseListener.java
$ antlr4-listener <grammarName> <listenerName>
# Copia <grammarName>BaseListener.java para o ficheiro <visitorName>.java
# Em alternativa, podemos ser nós a copiá-lo
```

Criar_main_com_visitor

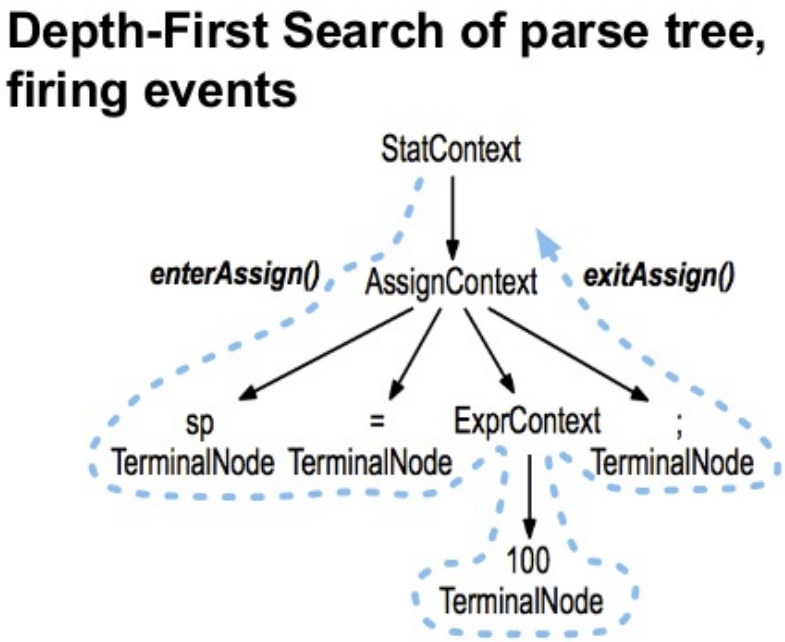
Para criar **main** com o visitor associado, devem ser executados os seguintes comandos:

```
$ antlr4-main -i <grammarName> <sintaticMainRuleName> -listener <listenerName>
$ antlr4-build <grammarName>
```

Contexto

No *listener*, para cada regra sintática `r: a b;`, são criados dois métodos: um método `enterR` e `exitR`, a primeira invocada quando o *walker* à medida que este percorrre a árvore em profundidade.

> A manipulação e processamento deve ser feita no exit, uma vez que só aí é garantido que o contexto dos nós inferiores foi gerado por completo!



Ambos os métodos têm como argumento o contexto da regra, 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()`.

No entanto e contrariamente ao *visitor*, no *listener*, os métodos não retornam qualquer valor (são *void*), pelo que há duas alternativas para passar este valor entre os métodos:

- **Adicionar atributos à gramática**

Ao adicionarmos à regra na gramática um valor de retorno, este passa a fazer parte do seu contexto e é assim acessível através do mesmo.

```
# Gramática
instrucao: numero '+' numero;
numero returns[Double p]: DOUBLE;

# Listener
public void exitInstrucao(ctx){
    Double n1 = ctx.numero(0).p;
    Double n2 = ctx.numero(1).p;
    System.out.println(String.format("%f + %f", n1, n2));
}
public void exitNumero(ctx){
    ctx.p = Double.parseDouble(ctx.DOUBLE().getText());
}
```

- Utilizar a classe **ParseTreeProperty**

Esta classe permite desacoplar o código da gramática, reduzindo a sua dependência da linguagem escolhida para a implementar

```
# Gramática
instrucao: numero '+' numero;
numero: DOUBLE;

# Listener
import org.antlr.v4.runtime.tree.ParseTreeProperty;
ParseTreeProperty<Double> map = new ParseTreeProperty<>();
public void exitInstrucao(ctx){
    Double n1 = map.get(ctx.numero(0));
    Double n2 = map.get(ctx.numero(0));
    System.out.println(String.format("%f + %f", n1, n2));
}
public void exitNumero(ctx){
    map.put(ctx, Double.parseDouble(ctx.DOUBLE().getText()));
}
```

> **ParseTreeProperty** (*org.antlr.v4.runtime.tree.ParseTreeProperty*)
>
> Implementa um Map (*protected*) que ao contexto (*ParseTree*) associa um valor
>
> void .put(ParseTree node, V value) Guarda um valor para o contexto dado
>
> V .get(ParseTree node) Devolve o valor associado ao contexto
>
> V .removeFrom(ParseTree node) Remove o valor associado a um contexto

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

Ver *visitor*

Adicionar_labels

Ver *visitor*

Percorrer_a_árvore_sintática

Ao contrário dos *visitors*, não pode haver qualquer manipulação à ordem pela qual os nós da árvore sintática são visitados, sendo todos visitados pelo *walker*.