ANTLR4介绍

2023编译原理项目

王立友

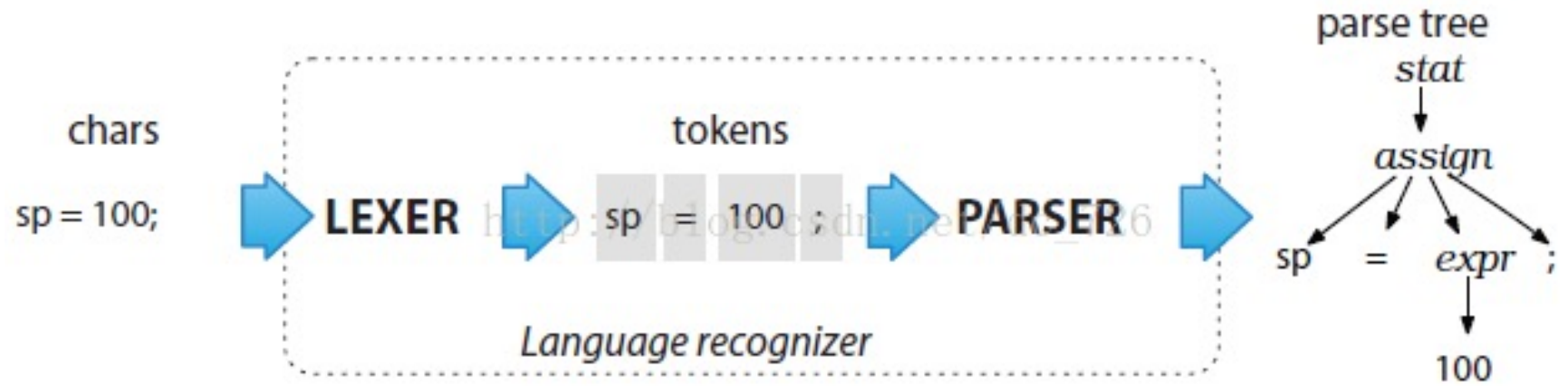
2023.12.07

同济大學軟件學院

ANTLR4介绍

ANTLR（全称：ANother Tool for Language Recognition）是目前非常流行的语言识别工具，使用Java语言编写，基于LL(*)解析方式，使用自上而下的递归下降分析方法。**通过输入语法描述文件来自动构造自定义语言的词法分析器、语法分析器和树状分析器等各个模块。**ANTLR使用上下无关文法描述语言，文法定义使用类似EBNF的方式。

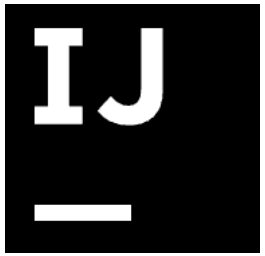
简单的讲，ANTLR是基于用户提供的语法规则文件，自动生成相应词法/语法分析器的一个工具，并提供给用户后续改造、加工的接口。



ANTLR4安装与使用

基于ANTLR4的开发文档，安装使用ANTLR4的方法有很多，有基于命令行的使用方式，同样也存在基于工具插件的使用方式，在此我以IntelliJ为例，展示如何将ANTLR4作为IntelliJ插件使用。

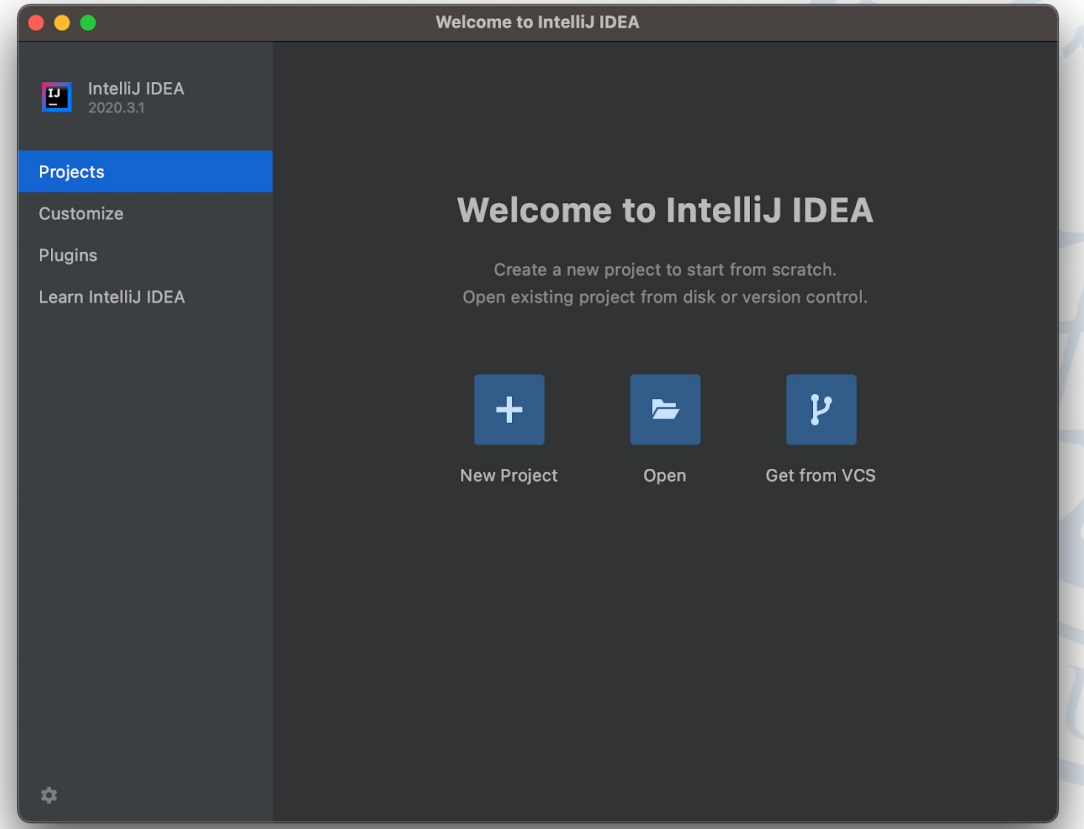
1. 下载IDEA

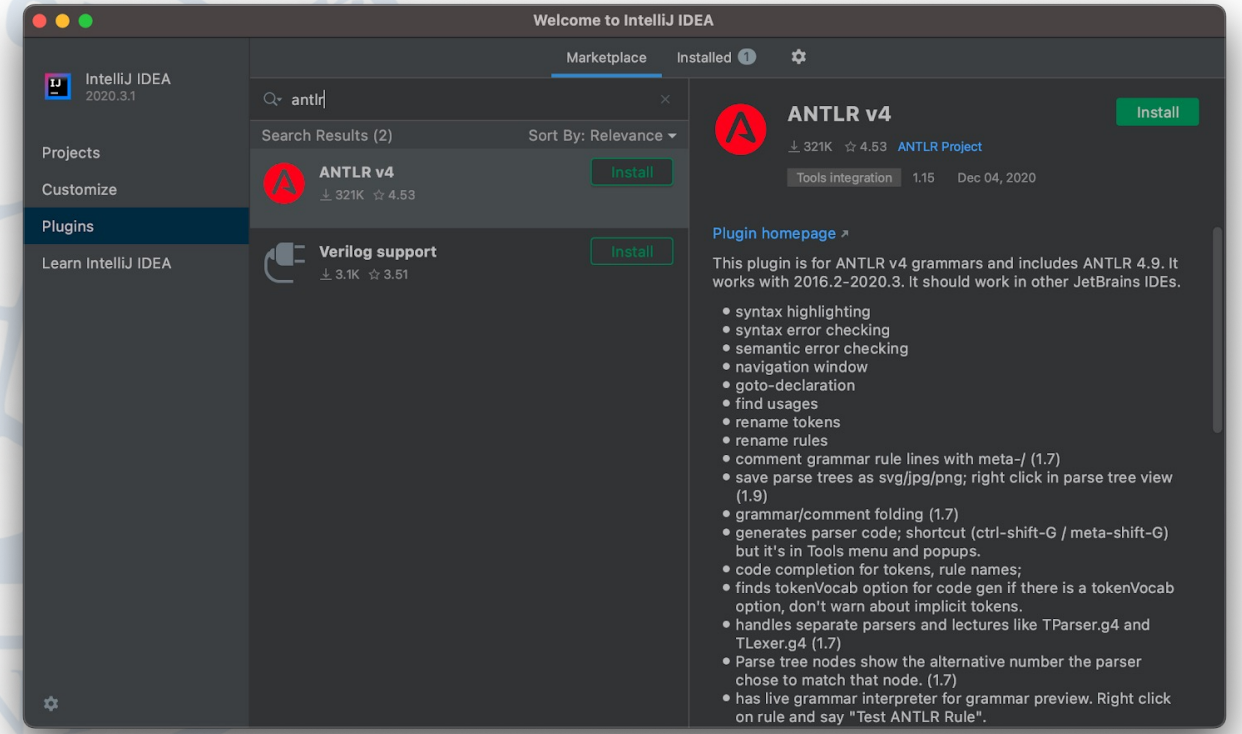


IntelliJ IDEA是一款功能完备、使开发更高效的JAVA开发IDE工具，接触过JAVA开发的同学应该对此相当熟悉了（未了解的同学也可以很快上手）

ANTLR4安装与使用

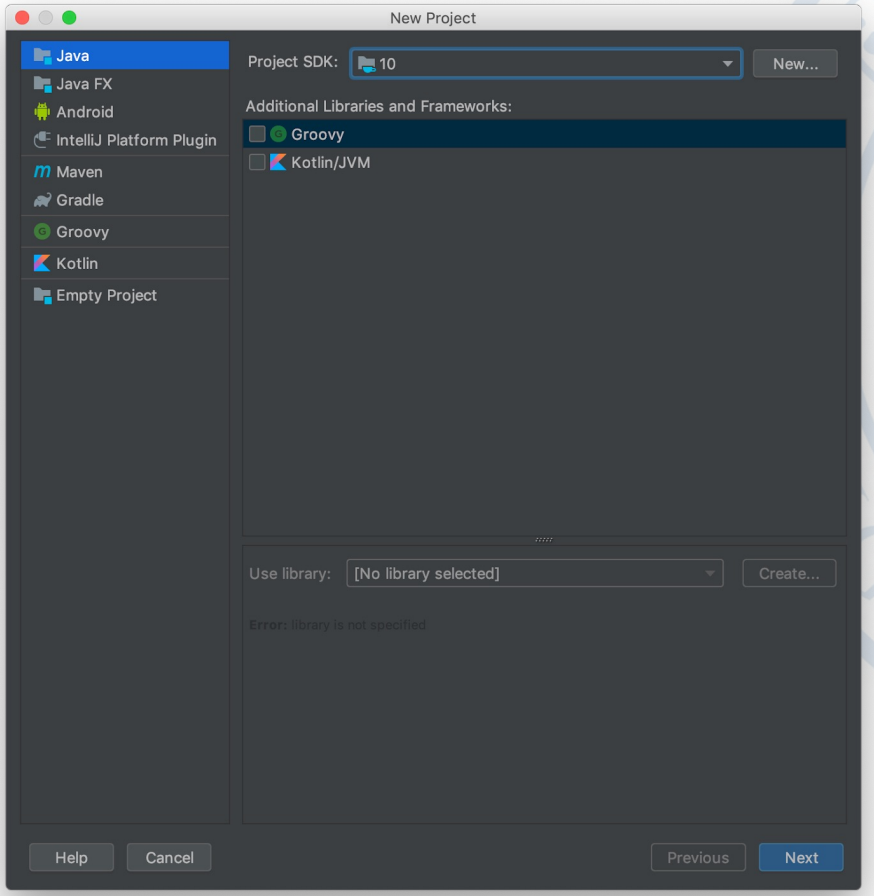
2. 安装ANTLR4插件

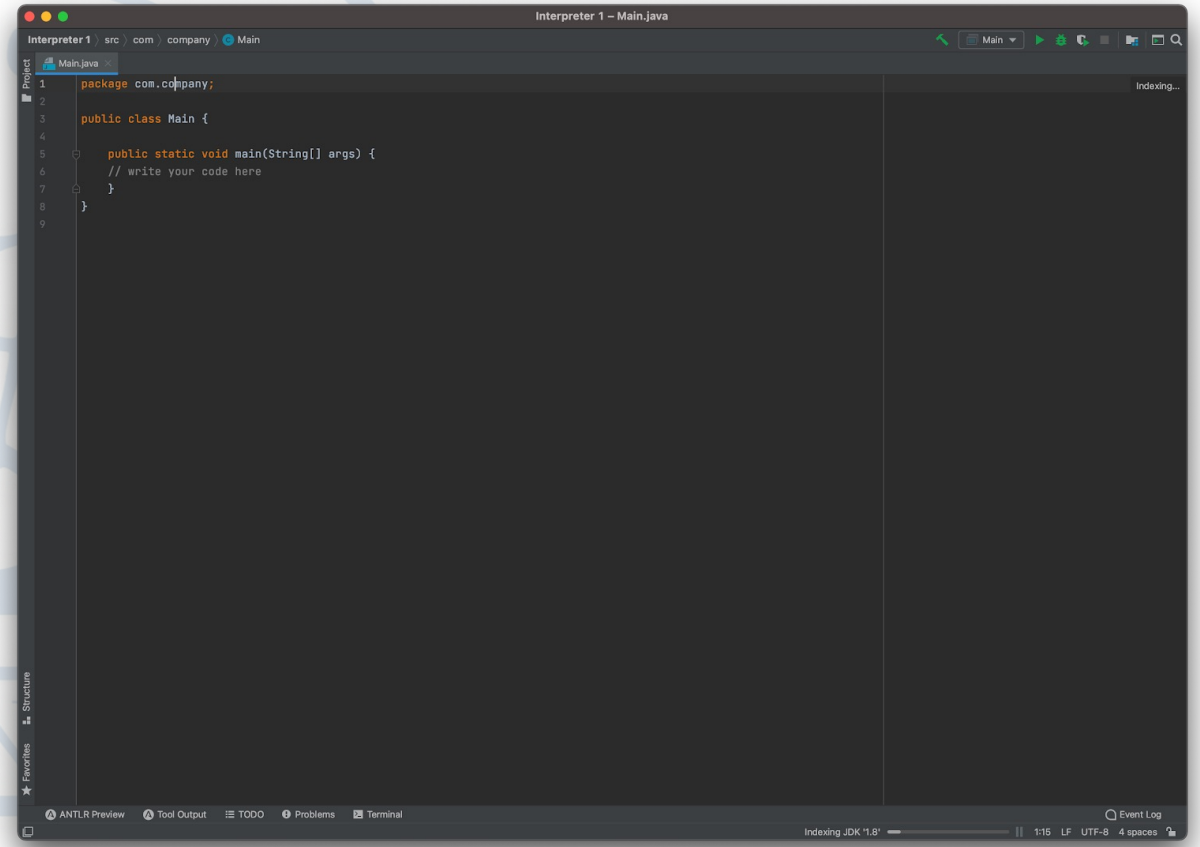




ANTLR4安装与使用

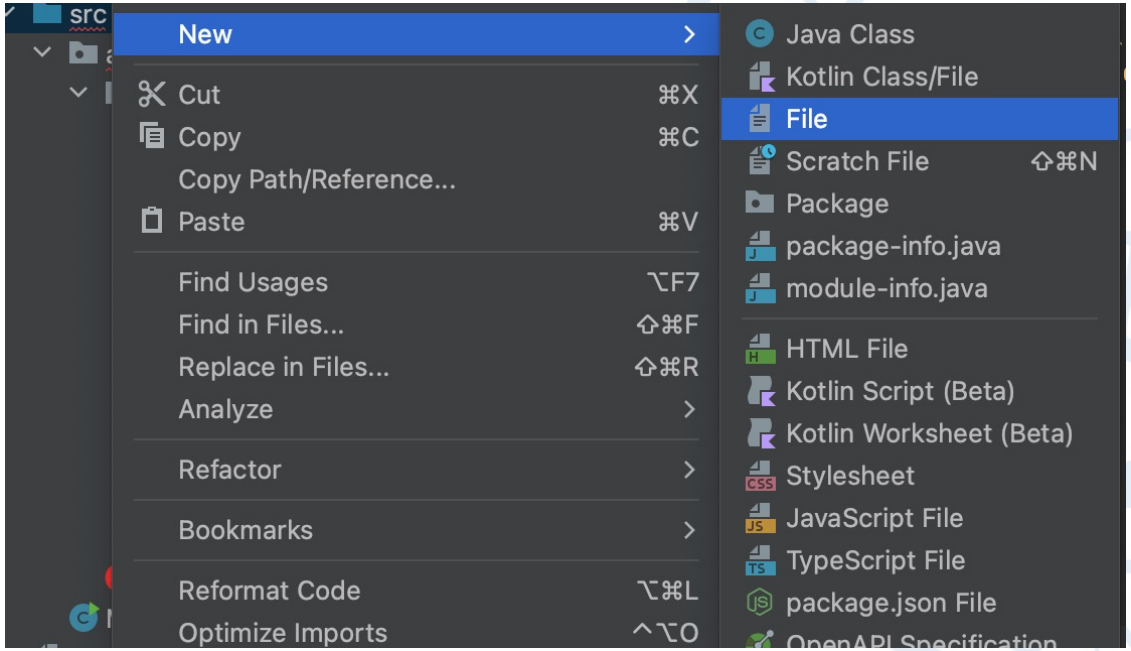
3. 创建一个项目





ANTLR4安装与使用

4. 创建ANTLR4语法文件（以加减乘除为例）



ANTLR4的语法文件以g4作为扩展名，在此例中，我们命名为Calculator.g4

ANTLR4安装与使用

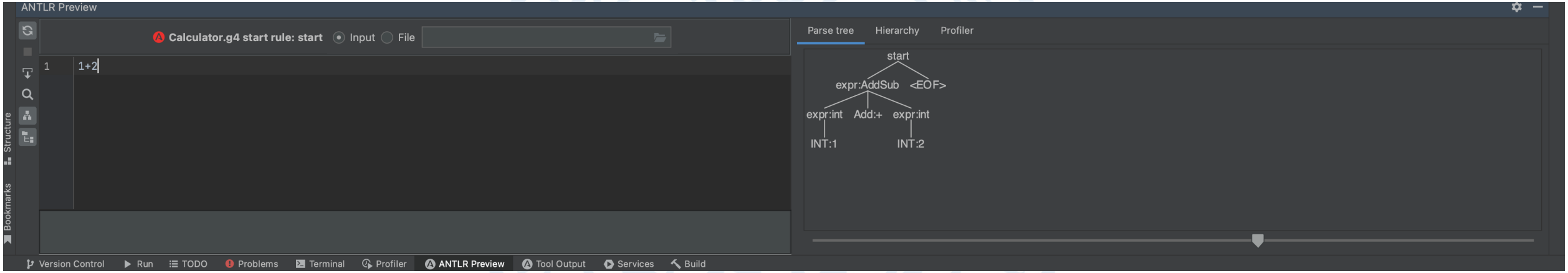
4. 创建ANTLR4语法文件（以加减乘除为例）

```
grammar Calculator;

/** 起始规则 语法分析器起点 */
start  :
     expr  EOF
  ;

expr:   expr op=('*'|'/') expr  # MulDiv
    |   expr op=('+'|'-') expr  # AddSub
    |   INT                     # int
    |   '(' expr ')'            # parens
    ;

INT     : [0-9]+ ;       // 匹配整数

MUL     : '*' ;
DIV     : '/' ;
Add     : '+' ;
SUB     : '-' ;
```

此处为Calculator.g4文件的具体内容，上部分定义了此例的语法、下部分定义了此例的词法，具体的grammar的规则，可以查看上述给出的ANTLR4的语法

ANTLR4安装与使用

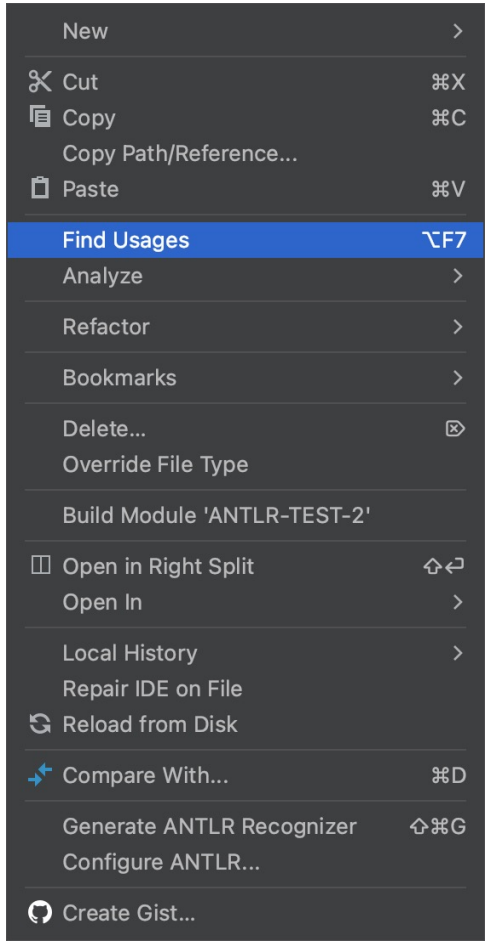
5. 测试语法是否正确



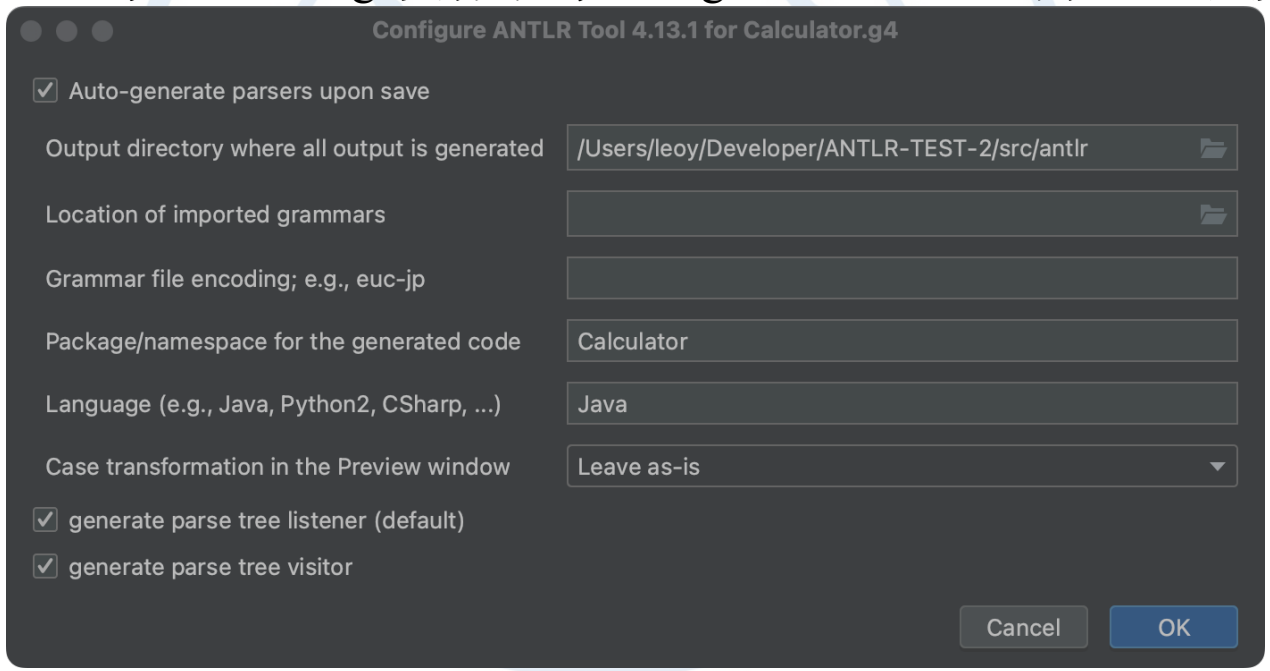
可以通过IDEA下部分的ANTLR Preview工具栏来查看构建的文法所形成的Parse tree，检查是否如我们预期所写的内容

ANTLR4安装与使用

6. 构建ANTLR语法器与词法器



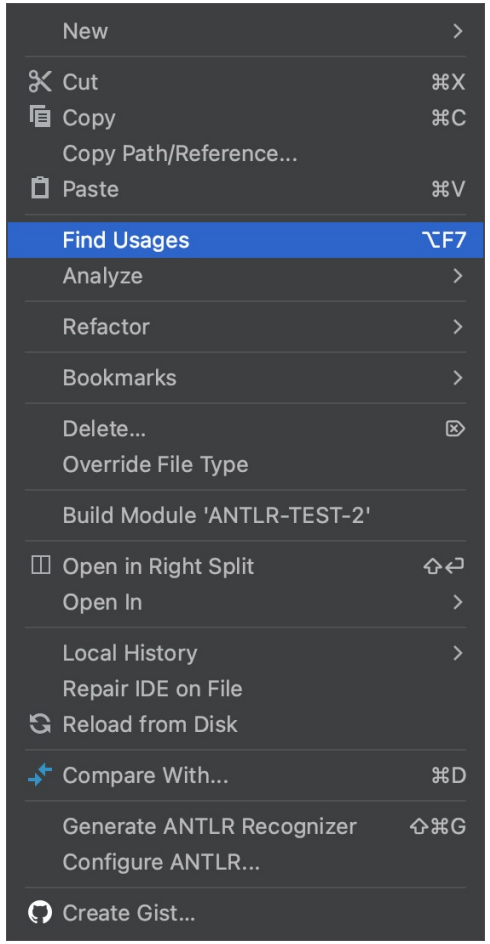
右击Calculator.g4文件下的Configure ANTLR… 来配置生成文件的属性



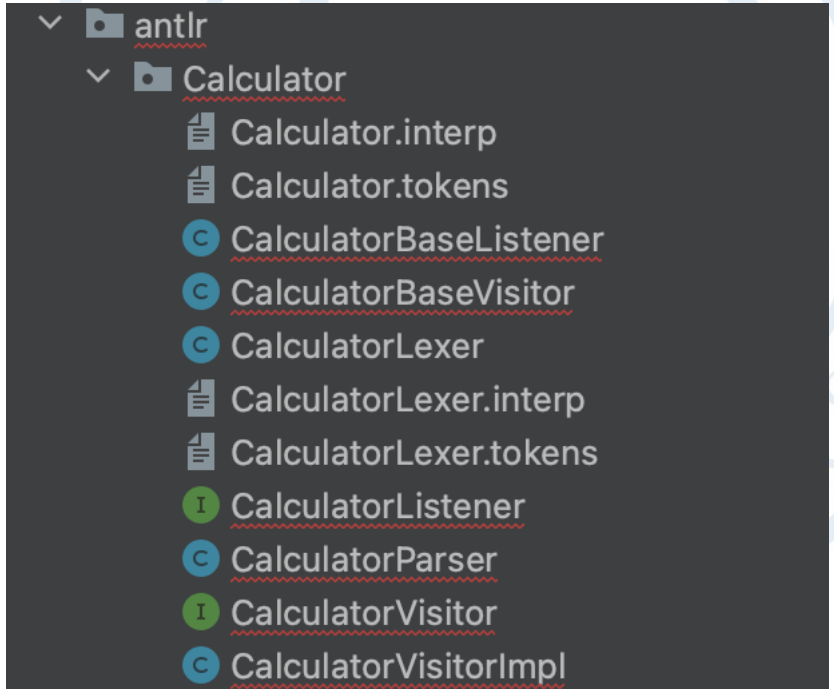
最值得关注的是，Language代表你期望生成的词法器和语法器是什么语言编写的（如Java、Python3等等）

ANTLR4安装与使用

7. 生成ANTLR语法器与词法器



右击Calculator.g4文件下的Generate ANTLR Recognizer来生成词法器与语法器



如CalculatorLexer即自动生成的词法器，我们可以根据这个类对输入的charstream分析转为tokens，CalculatorParser即语法分析器

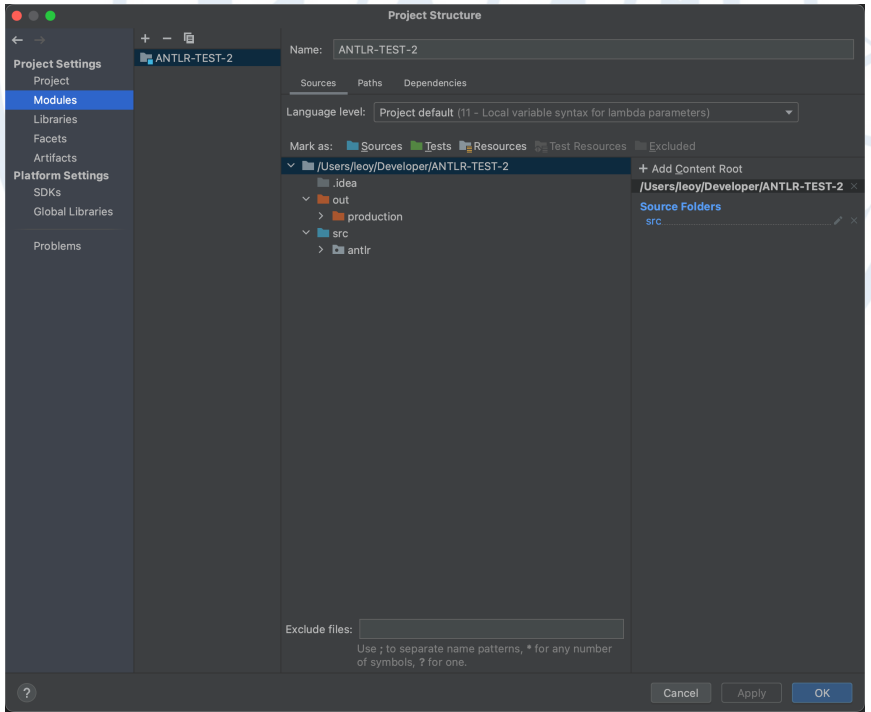
ANTLR4安装与使用

8. 导入Antlr依赖

首先需要下载ANTLR的jar包。该文件是一个“Jar”文件，这基本上意味着它是一个压缩文件，包含我们可以执行的代码，所有这些代码都以Java理解的方式捆绑在一起。我们需要把代码放在某个地方。

要将其添加到我们的项目中，请转到“File”->“Project Structure”。从那里，我们可以看到这样一个窗口：

在该窗口中，使用Dependencies选项卡，单击小加号，然后选择“JAR或目录...”，然后浏览到我们从Antlr站点下载的文件的存储位置。然后我们将列出依赖项。



ANTLR4安装与使用

9. 运行Java程序

修改Main函数代码，执行对grammar的测试

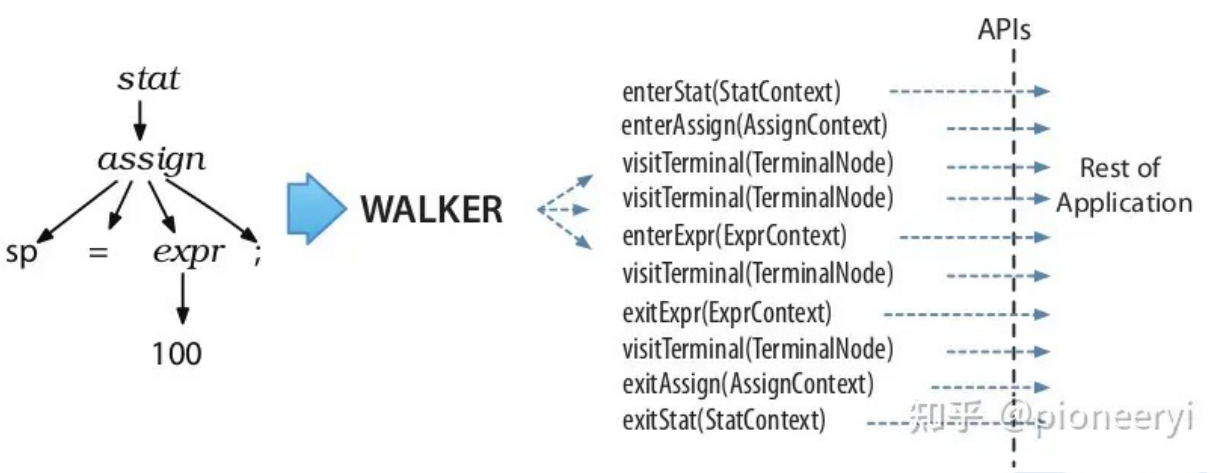
```
import antlr.Calculator.CalculatorVisitorImpl;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import Calculator.CalculatorLexer;
import Calculator.CalculatorParser;
import Calculator.CalculatorBaseVisitor;

public class Main {
    public static void main(String[] args) {

        CalculatorLexer lexer = new CalculatorLexer(CharStreams.fromString( s: "5+4+2/3+5-7*9"));
        CalculatorParser parser = new CalculatorParser((new CommonTokenStream(lexer)));
        parser.start();
//        parser.setBuildParseTree(true);
//        CalculatorParser.StartContext tree = parser.start();
//        CalculatorBaseVisitor<String> visitor = new CalculatorVisitorImpl();
//        visitor.visit(tree);
        System.out.println("parser has executed");

    }
}
```

中间代码生成简单示例

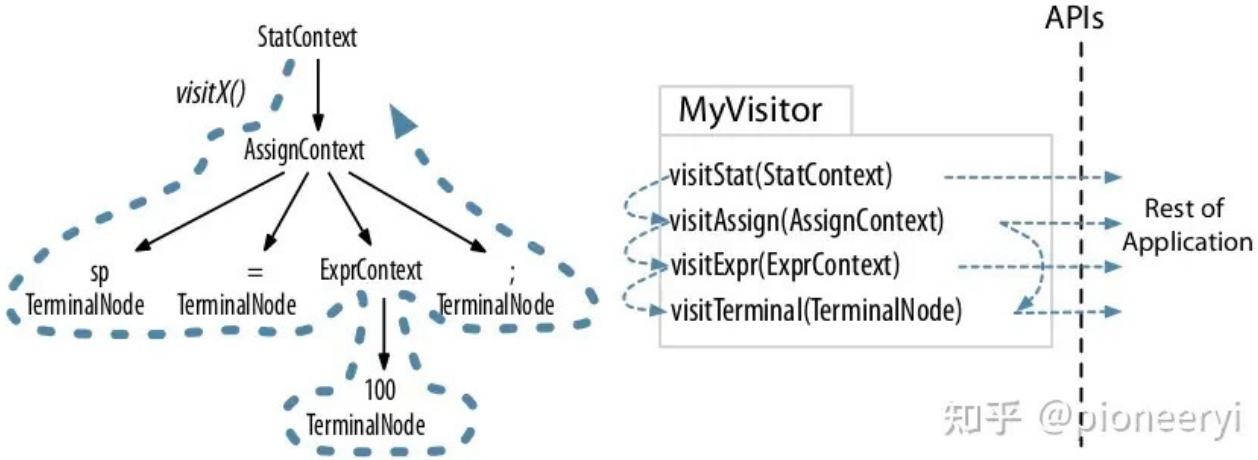
1. 理解Listener和Visitor

可以看到，我们在生成词法器Lexer和语法器的时候，同时还有一些额外的代码文件的产生，如Listener和Visitor，这两种方式即我们访问Parse Tree的途径，也就是在词法与语法分析后，需要自行进行扩展的代码部分。

Listener



Visitor



中间代码生成简单示例

2. 利用Visitor写一个简单的中间代码生成

在此例中，我们通过Visitor方式来访问Parse tree，实现对一个表达式的中间代码生成。具体的方法是实现一个自定义的Visitor类（CalculatorVisitorImpl 来extends扩展原始的基类CalculatorBaseVisitor，其中该基类需要设置返回值类型，在该例中，我希望返回某一节点代表的临时变量名，因此设置为CalculatorBaseVisitor<String>，具体问题具体分析。

```java
public class CalculatorVisitorImpl extends CalculatorBaseVisitor<String> {
    1 usage
    private int tempVarCounter = 0;

    2 usages
    private String newTempVar() {
        return "t" + tempVarCounter++;
    }

    2 related problems
    @Override
    public String visitMulDiv(CalculatorParser.MulDivContext ctx) {
        String left = visit(ctx.expr( i: 0));
        String right = visit(ctx.expr( i: 1));
        String op = ctx.op.getText();
        String tempVar = newTempVar();
        System.out.println(tempVar + " = " + left + " " + op + " " + right);
        return tempVar;
    }
```

中间代码生成简单示例

3. 中间代码生成测试

```java
import antlr.Calculator.CalculatorVisitorImpl;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import Calculator.CalculatorLexer;
import Calculator.CalculatorParser;
import Calculator.CalculatorBaseVisitor;

public class Main {
    public static void main(String[] args) {

        CalculatorLexer lexer = new CalculatorLexer(CharStreams.fromString( s: "5+4+2/3+5-7*9"));
        CalculatorParser parser = new CalculatorParser((new CommonTokenStream(lexer)));
//        parser.start();
        parser.setBuildParseTree(true);
        CalculatorParser.StartContext tree = parser.start();
        CalculatorBaseVisitor<String> visitor = new CalculatorVisitorImpl();
        visitor.visit(tree);
        System.out.println("parser has executed");

    }
}
```

```
/Users/leoy/Library/Java/JavaVirtualMachines/corretto-11.0.19/Con
t0 = 5 + 4
t1 = 2 / 3
t2 = t0 + t1
t3 = t2 + 5
t4 = 7 * 9
t5 = t3 - t4
parser has executed
```

感谢观看

2023编译原理项目

王立友

2023.12.07