

拉勾教育

— 互联网人实战大学 —

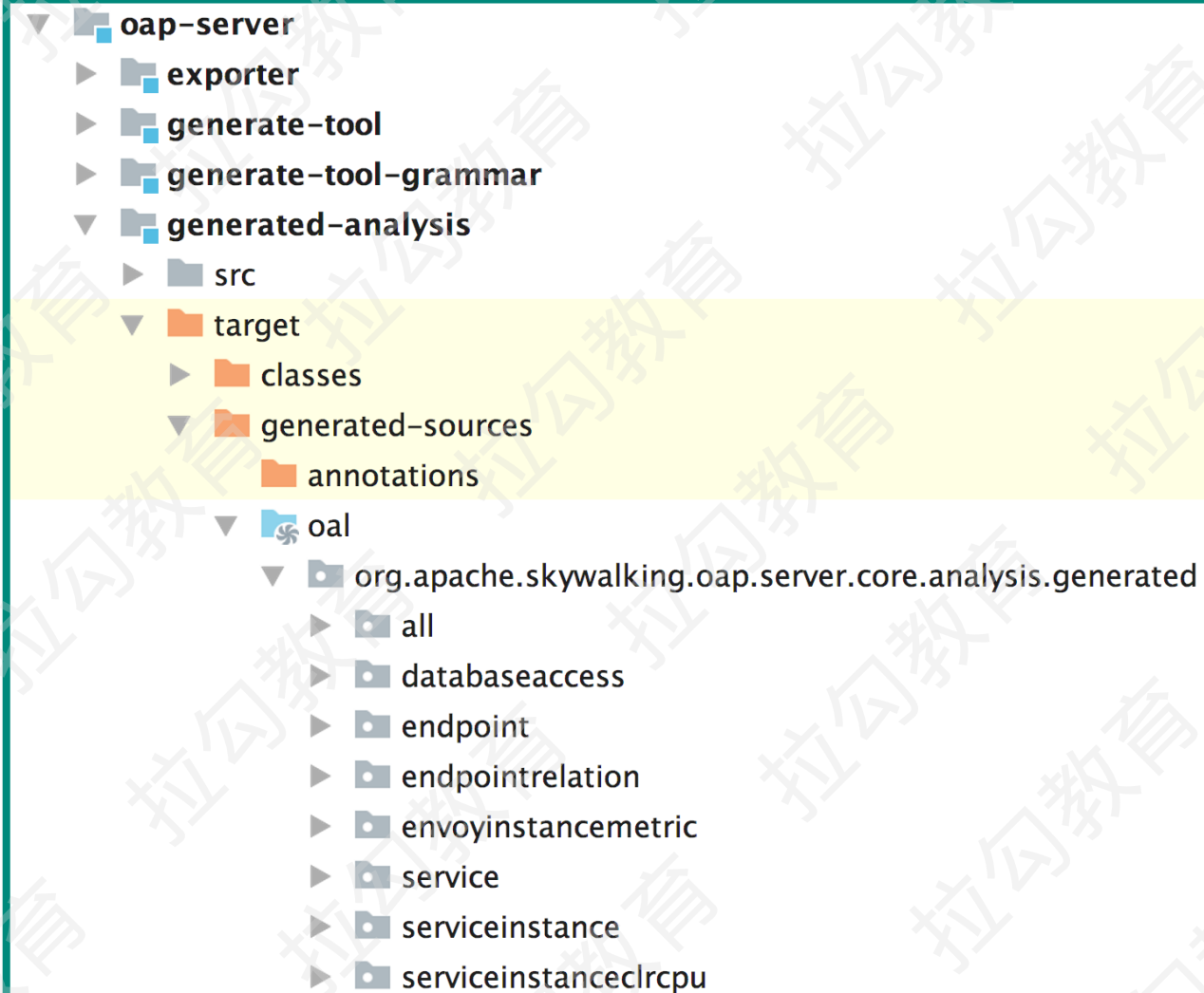
# 《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

# 第31讲：OAL 语言 原来定义创造一门新语言如此轻松（上）

```
/**  
 * This class is auto generated. Please don't change this class manually.  
 *  
 * @author Observability Analysis Language code generator  
 */
```



```
▼ oap-server
  ► exporter
  ► generate-tool
  ► generate-tool-grammar
  ▼ generated-analysis
    ► src
    ▼ target
      ► classes
      ▼ generated-sources
        ► annotations
      ▼ oal
        ▼ org.apache.skywalking.oap.server.core.analysis.generated
          ► all
          ► databaseaccess
          ► endpoint
          ► endpointrelation
          ► envoyinstancemetric
          ► service
          ► serviceinstance
          ► serviceinstanceclrcpu
```

**Antlr4 (Another Tool for Language Recognition)** 是一款强大的语法生成器工具

可以根据输入的字节流自动生成语法树

作为一款开源语法分析器，可用于读取、处理、执行和翻译结构化的文本或二进制文件



- 在很多大数据系统中都使用 Antlr4

例如 Hadoop 生态系统中的 Hive、Spark 数据仓库和分析系统所使用的语言，都用到了 Antlr4

- Hibernate 对象-关系映射框架（ORM）使用 Antlr 来处理 HQL 语言
- Oracle 公司在 SQL 开发者 IDE 和迁移工具中使用了 Antlr4
- NetBeans 使用 Antlr4 来解析 C++



**编译**的目的是将程序员日常使用的高级编程语言翻译成物理机或是虚拟机可以执行的二进制指令

**词法分析器**的工作是读取、解析程序员写出来的代码文件，这些文件基本都是文本文件

词法分析器通过读取代码文件中的字节流，就可以将其翻译成一个一个连续的、编程语言预先定义好的 Token

一个 Token 可以是关键字、标识符、符号（symbols）和操作符等等

下面的语法分析器将通过这些 Token 构造抽象语法树（Abstract Syntax Tree, AST）



在分析读取到的字符流时

**词法分析器 (Lexer)** 并不关心所生成的单个 Token 的语法意义及其与上下文之间的关系

**语法分析器 (Parser)** 将收到的所有 Token 组织起来，并转换成为目标语言语法定义所允许的序列

本质上是类似的东西，而只是在分工上有所不同而已

```
sp = 100;
```

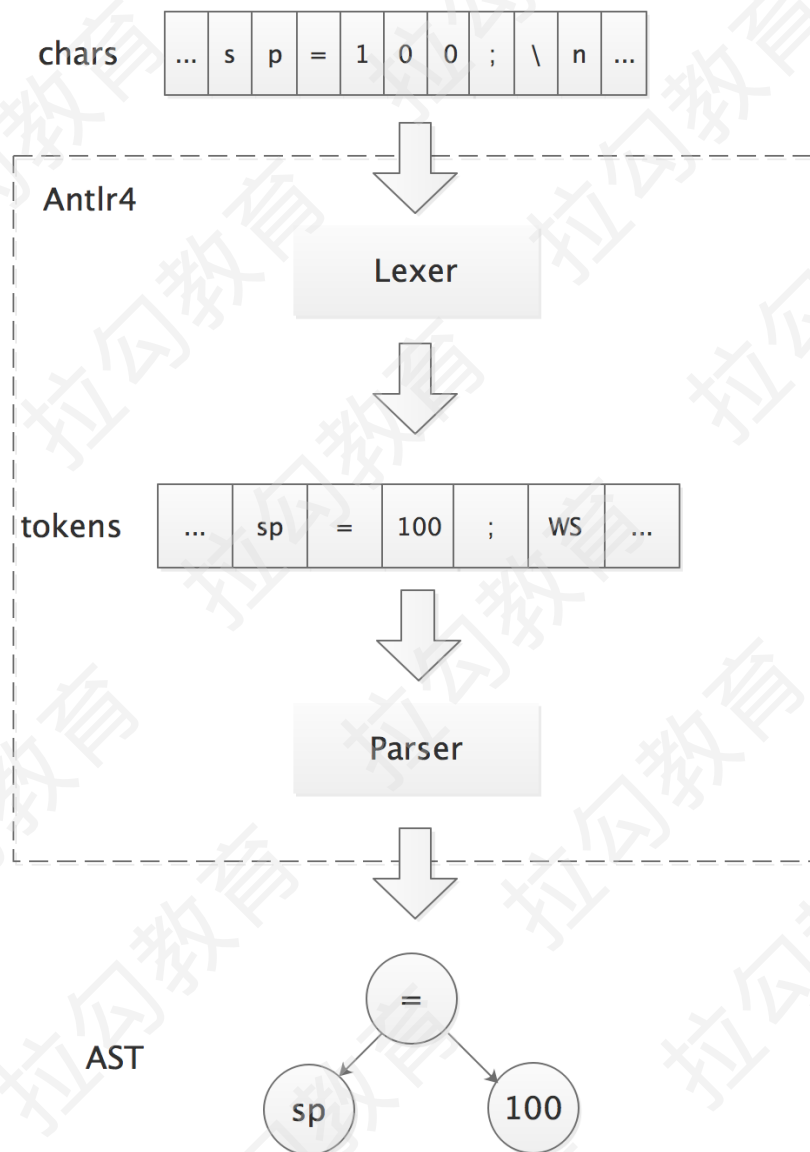




# Antlr4 基础入门

拉勾教育

— 互联网人实战大学 —



```
grammar Calculator;

expr : '(' expr ')'
      | '*' expr
      | '/' expr
      | '+' expr
      | '-' expr
      | FLOAT
      ;

line : expr EOF ;
WS : [ \t\n\r ]+ -> skip;
FLOAT : DIGIT+ '.' DIGIT* EXPONET?
       | '.' DIGIT+ EXPONET?
       | DIGIT+ EXPONET?
       ;

fragment DIGIT : '0'..'9';
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+;
```

grammar Calculator;

```
expr : '(' expr ')'
      | '*' '/' expr
      | '+' '-' expr
      | FLOAT
```

```
line : expr EOF ;
WS : [ \t\n\r ]+ -> skip;
FLOAT : DIGIT+ '.' DIGIT* EXPONET?
       | '.' DIGIT+ EXPONET?
       | DIGIT+ EXPONET?
```

```
fragment DIGIT : '0'..'9';
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+;
```

```
grammar Calculator;
```

```
expr : '(' expr ')'
      | '*' expr
      | '/' expr
      | '+' expr
      | '-' expr
      | FLOAT
```

```
line : expr EOF ;
```

```
WS : [ \t\n\r ]+ -> skip;
```

```
FLOAT : DIGIT+ '.' DIGIT* EXPONET?
```

```
       | '.' DIGIT+ EXPONET?
```

```
       | DIGIT+ EXPONET?
```

```
fragment DIGIT : '0'..'9' ;
```

```
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+ ;
```

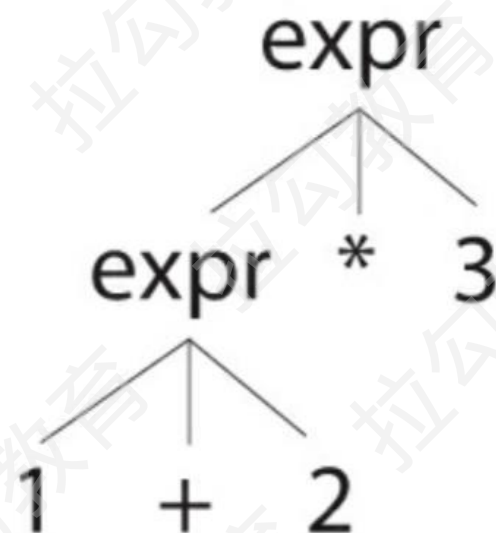
expr 由 4 个备选分支，不同的备选分支由 “|” 分割，expr 规则的含义分别是：

- 第一个备选分支表示 expr 语句可以由另一个 expr 加上左右两个括号构成
- 第二个备选分支表示 expr 语句可以是  $\text{expr} * \text{expr}$  或是  $\text{expr} / \text{expr}$  格式
- 第三个备选分支表示 expr 语句可以是  $\text{expr} + \text{expr}$  或是  $\text{expr} - \text{expr}$  格式
- 第四个备选分支表示 expr 语句可以是 FLOAT



例如这里的 `expr` 规则在处理 `1 + 2 * 3` 这个表达式的时候

因为 `expr * expr` 的分支在前，生产的语法树如下



```
grammar Calculator;
```

```
expr : '(' expr ')'
      | '*' expr
      | '/' expr
      | '+' expr
      | '-' expr
      | FLOAT
```

```
line : expr EOF ;
```

```
WS : [ \t\n\r ]+ -> skip;
```

```
FLOAT : DIGIT+ '!' DIGIT* EXPONET?
```

```
       | '!' DIGIT+ EXPONET?
```

```
       | DIGIT+ EXPONET?
```

```
fragment DIGIT : '0'..'9' ;
```

```
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+ ;
```

```
grammar Calculator;
```

```
expr : '(' expr ')'
      | '*' '/' expr
      | '+' '-' expr
      | FLOAT
```

```
line : expr EOF ;
```

```
WS : [ \t\n\r ]+ -> skip;
```

```
FLOAT : DIGIT+ '!' DIGIT* EXPONET?
```

```
       | '!' DIGIT+ EXPONET?
```

```
       | DIGIT+ EXPONET?
```

```
fragment DIGIT : '0'..'9' ;
```

```
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+ ;
```



```
grammar Calculator;
```

```
expr : '(' expr ')'
      | '*' expr
      | '/' expr
      | '+' expr
      | '-' expr
      | FLOAT
```

```
line : expr EOF ;
```

```
WS : [ \t\n\r ]+ -> skip;
```

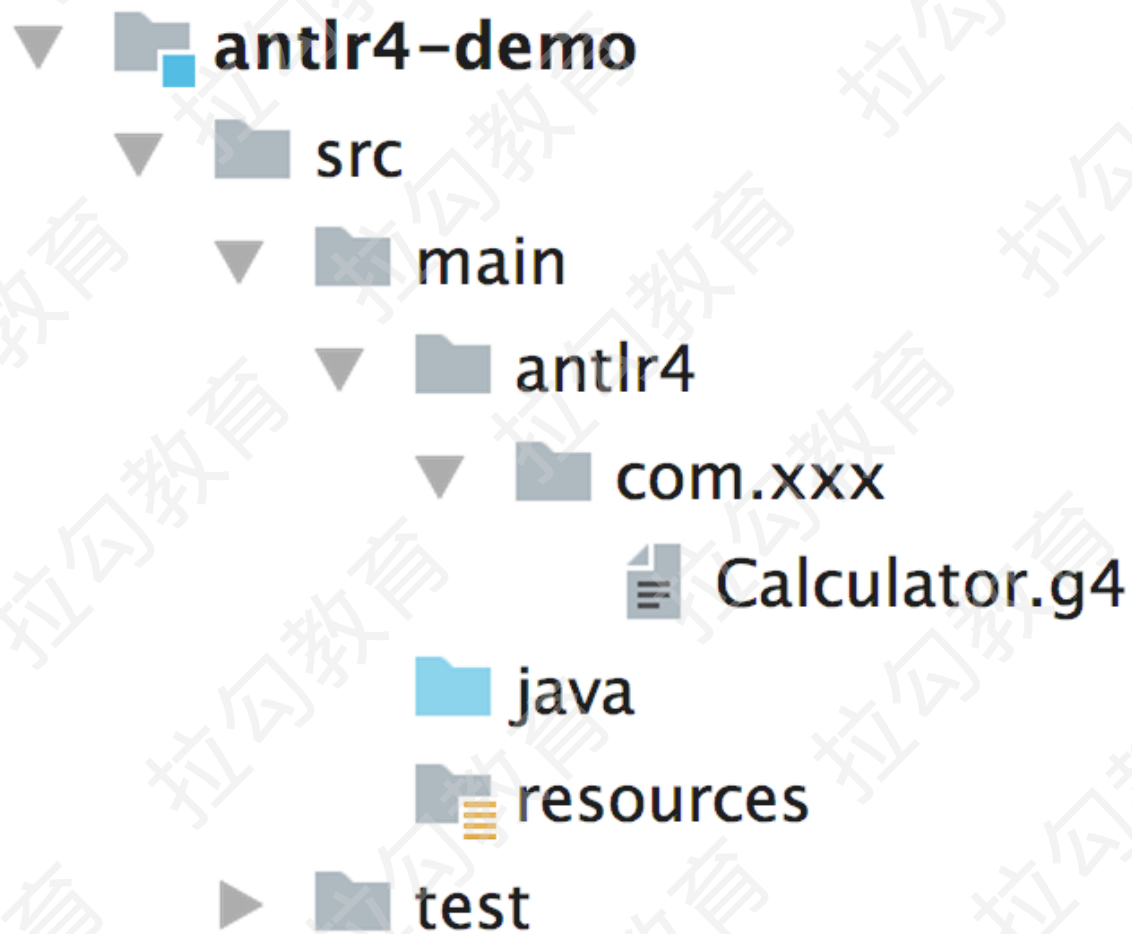
```
FLOAT : DIGIT+ '.' DIGIT* EXPONENT?
```

```
       | '.' DIGIT+ EXPONENT?
```

```
       | DIGIT+ EXPONENT?
```

```
fragment DIGIT : '0'..'9' ;
```

```
fragment EXPONENT : ('e'|'E') ('+'|'-')? DIGIT+ ;
```



```
<dependencies>
  <dependency>
    <groupId>org antlr</groupId>
    <artifactId>antlr4</artifactId>
    <version>4.7.1</version>
  </dependency>
</dependencies>

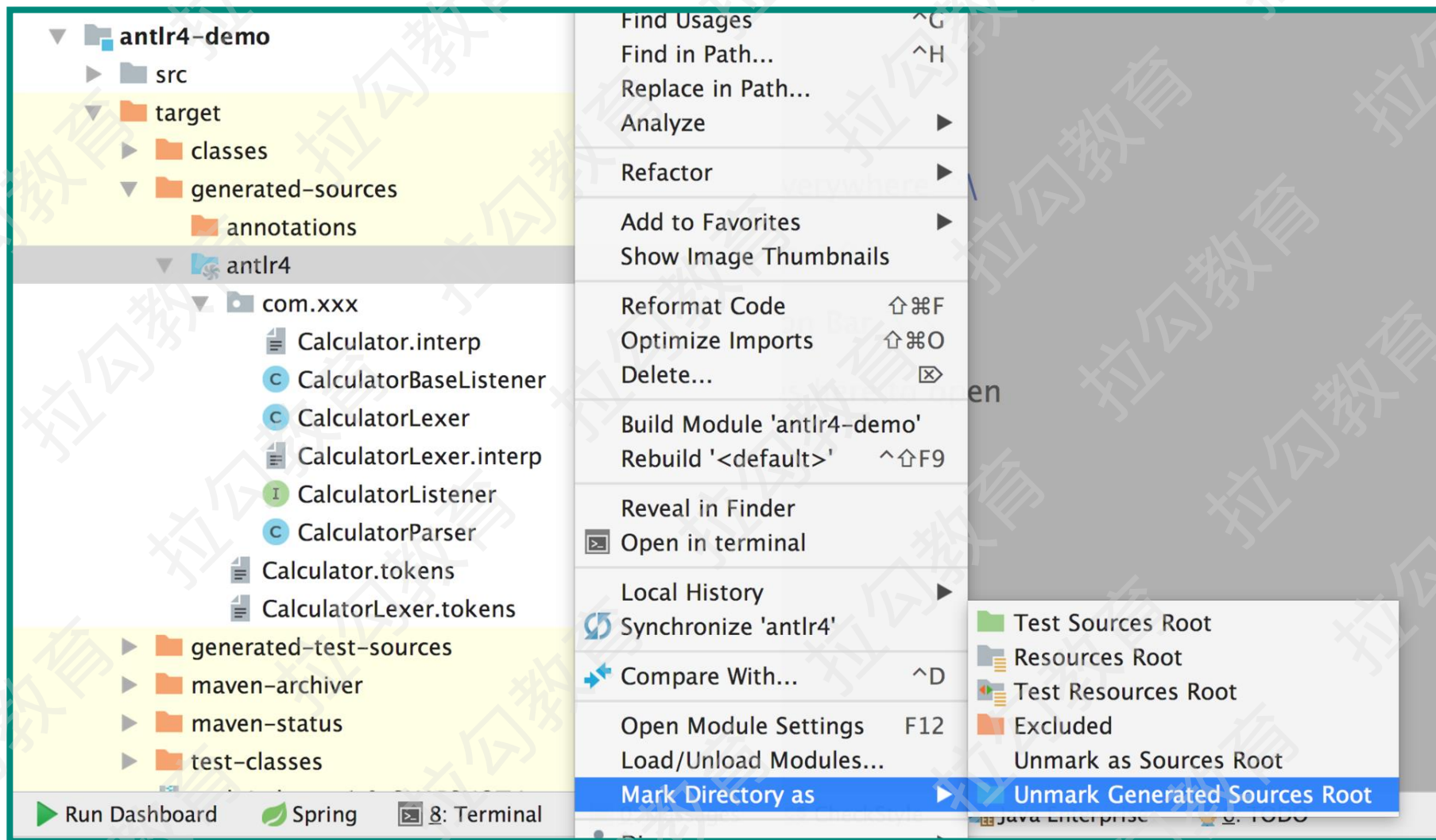
<build>
  <plugins>
    <plugin>
      <groupId>org antlr</groupId>
      <artifactId>antlr4-maven-plugin</artifactId>
      <version>4.7.1</version> 
      <executions>
        <execution>
          <id>antlr</id>
          <goals>
```

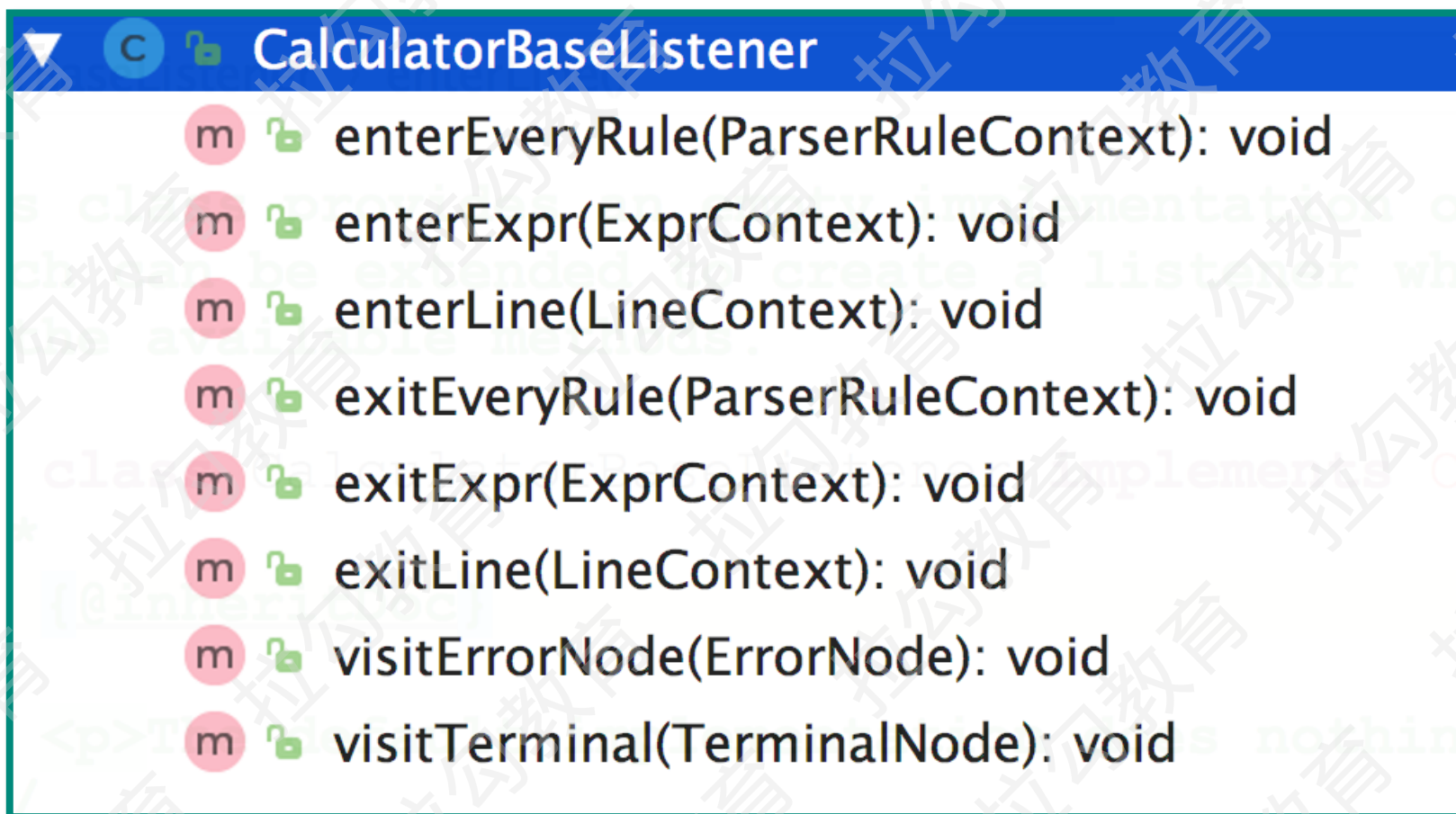
```
<build>
  <plugins>
    <plugin>
      <groupId>org antlr</groupId>
      <artifactId>antlr4-maven-plugin</artifactId>
      <version>4.7.1</version <!-- 与jar包版本相同 -->
      <executions>
        <execution>
          <id>antlr</id>
          <goals>
            <goal>antlr4</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

# Antlr4 基础入门

拉勾教育

— 互联网人实战大学 —







```
public class PrintListener extends com.xxx.CalculatorBaseListener {  
    @Override  
    public void enterLine(com.xxx.CalculatorParser.LineContext ctx) {  
        System.out.println("enterLine:" + ctx.getText());  
    }  
    ..... // 省略其他方法的实现  
}
```

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        //这里处理的是"1+2"这一行计算器语言，读取得到字节流  
        ANTLRInputStream input = new ANTLRInputStream("1+2");  
        //创建CalculatorLexer，词法分析器(Lexer)识别字节流得到Token流  
        CalculatorLexer lexer = new CalculatorLexer(input);  
        CommonTokenStream tokens = new CommonTokenStream(lexer);  
        //创建CalculatorParser，语法分析器(Parser)识别Token流得到AST  
        CalculatorParser parser = new CalculatorParser(tokens);  
        ParseTree tree = parser.line();  
        //遍历AST中各个节点回调PrintListener中相应的方法  
        ParseTreeWalker walker = new ParseTreeWalker();  
        walker.walk(new PrintListener(), tree);  
        //将整个AST转换成字符串输出  
        System.out.println(tree.toStringTree(parser));  
    }  
}
```



```
enterEveryRule:1+2<EOF>
enterLine:1+2<EOF> # ??é??? line: 1+2
enterEveryRule:1+2
enterExpr:1+2 # ??é??? expr: 1+2
enterEveryRule:1
enterExpr:1 # ??é??? expr: 1
visitTerminal:1
exitExpr:1 # ??é??? expr: 1
exitEveryRule:1
visitTerminal:+
enterEveryRule:2
enterExpr:2 # ??é??? expr: 2
visitTerminal:2
exitExpr:2 # ??é??? expr: 2
exitEveryRule:2
exitExpr:1+2 # ??é??? expr: 1+2
exitEveryRule:1+2
visitTerminal:<EOF>
```

```
enterEveryRule:1+2
enterExpr:1+2 # ??é??? expr: 1+2
enterEveryRule:1
enterExpr:1 # ??é??? expr: 1
visitTerminal:1
exitExpr:1 # ??é??? expr: 1
exitEveryRule:1
visitTerminal:+
enterEveryRule:2
enterExpr:2 # ??é??? expr: 2
visitTerminal:2
exitExpr:2 # ??é??? expr: 2
exitEveryRule:2
exitExpr:1+2 # ??é??? expr: 1+2
exitEveryRule:1+2
visitTerminal:<EOF>
exitEveryRule:1+2<EOF>
(line (expr 1) + (expr 2)) <EOF>
```

# Antlr4 基础入门

拉勾教育

— 互联网人实战大学 —

ANTLR Preview:

**Calculator.g4 start rule: expr**

File

1	1+2

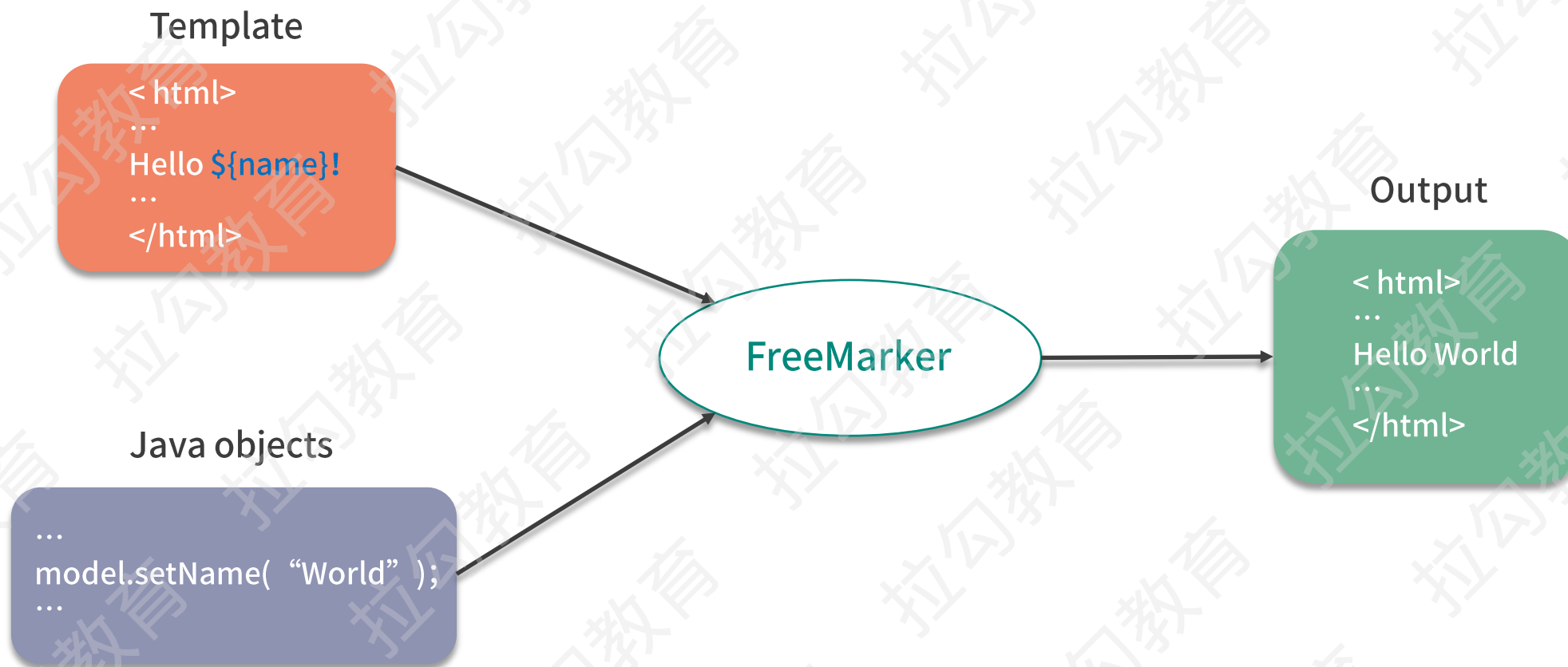
Parse tree Hierarchy Profiler

```
graph TD
    expr3["expr:3"] --> expr4_1["expr:4"]
    expr3 --> plus["null: '+'"]
    expr3 --> expr4_2["expr:4"]
    expr4_1 --> float1["FLOAT: '1'"]
    expr4_2 --> float2["FLOAT: '2'"]
```

# FreeMarker 基础入门

拉勾教育

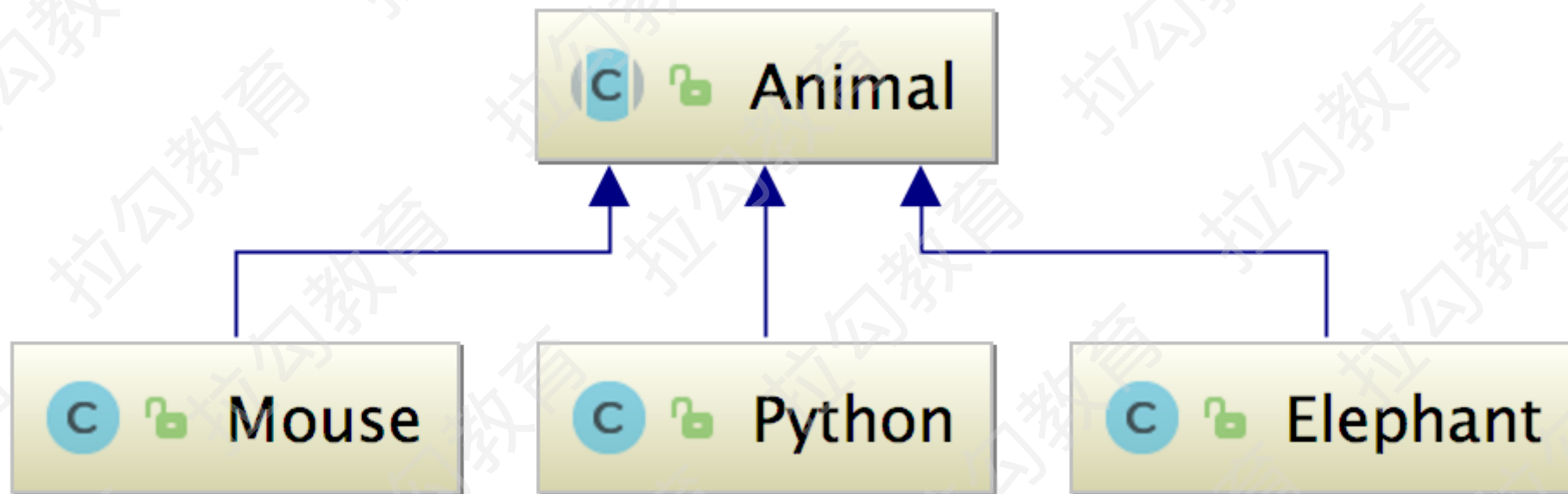
— 互联网人实战大学 —



# FreeMarker 基础入门

拉勾教育

— 互联网人实战大学 —



```
<html>
<body>
<h1>
  <!-- 使用 ${...} 展示一个变量 -->
  Welcome ${user}
  <!-- 使用<#if><#else>标签实现条件分支 -->
  <#if user == "freemarker-user">
    our leader<#else>
    out user</#if>
</h1>
<p>We have these animals:
<table border=1>
  <tr>
    <td>Animal Name</td>
    <td>Price</td>
    <td>Size</td>
  </tr>
  <!-- 使用 <#list ... as> 标签实现对 List 集合的遍历 -->
  <#list animals as animal>
  <tr>
    <td>${animal.name}</td>
```

```
our leader<#else>
out user</#if>
</h1>
<p>We have these animals:
<table border=1>
  <tr>
    <td>Animal Name</td>
    <td>Price</td>
    <td>Size</td>
  </tr>
  <!-- 使用 <#list ... as> 标签实现对 List 集合的遍历 -->
  <#list animals as animal>
    <tr>
      <td>${animal.name}</td>
      <td>${animal.price}</td>
      <td>${animal.size}</td>
    </tr>
  </#list>
</table>
</body>
</html>
```

```
public static void main(String[] args) throws Exception {  
    //1.初始化并配置Configuration对象  
    Configuration configuration =  
        new Configuration(Configuration.getVersion());  
    //2.设置模板文件所在的目录  
    configuration.setClassForTemplateLoading(Main.class, "/template");  
    //3.设置字符集  
    configuration.setDefaultEncoding("utf-8");  
    //4.加载模板文件  
    Template template = configuration.getTemplate("test.ftl");  
    //5.创建数据模型  
    Map<String, Object> result = createData();  
    //6.创建Writer对象  
    FileWriter writer = new FileWriter(  
        new File("/Users/xxx/Documents/log/test.html"));  
    //7.输出数据模型到文件中  
    template.process(result, writer);  
    //8.关闭Writer对象  
    writer.close();  
}
```

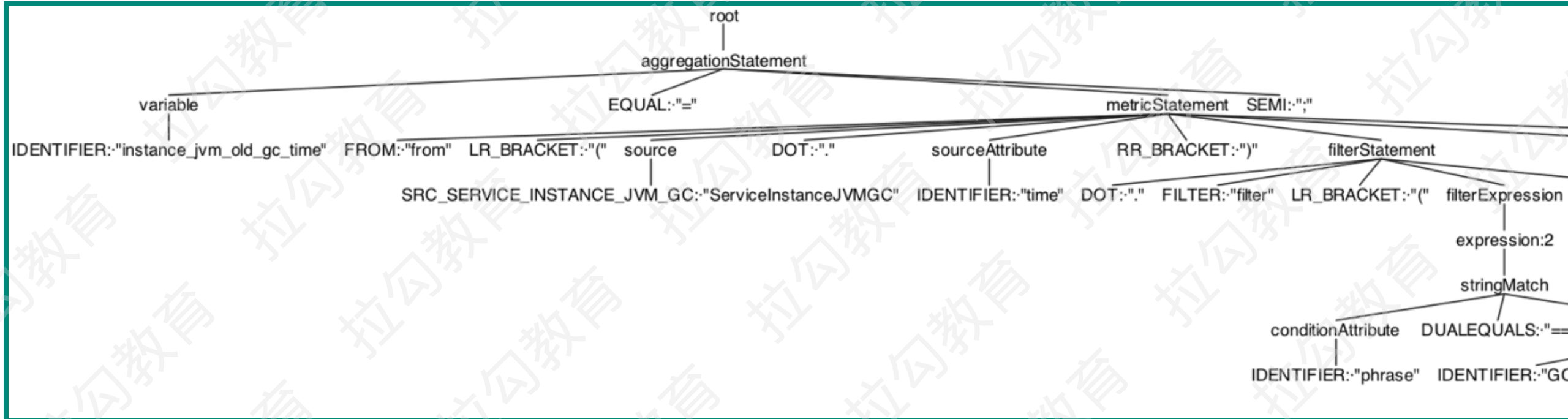


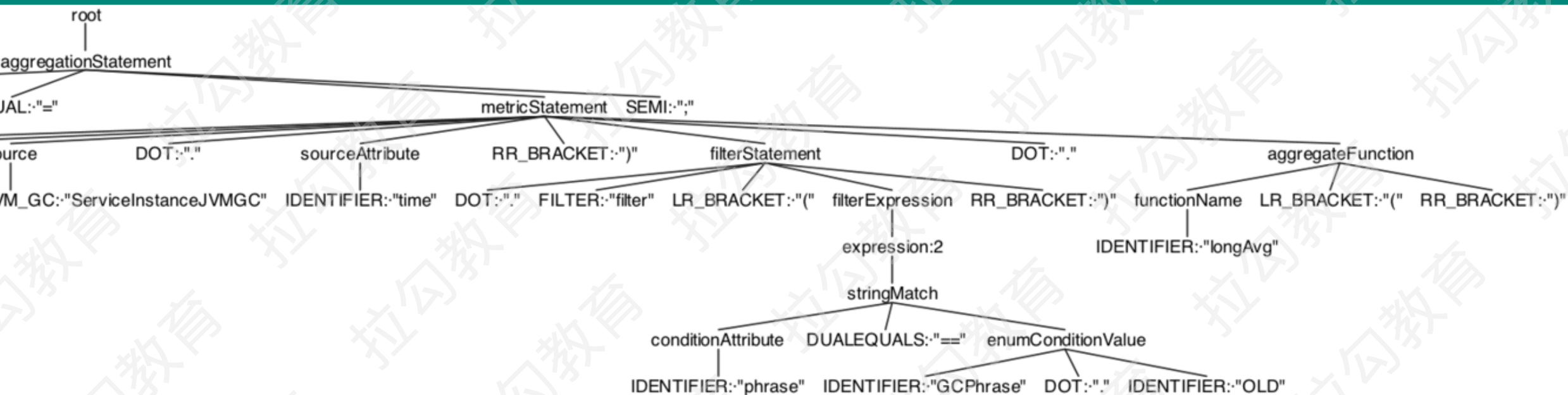
# Welcome, freemarker-user our leader!

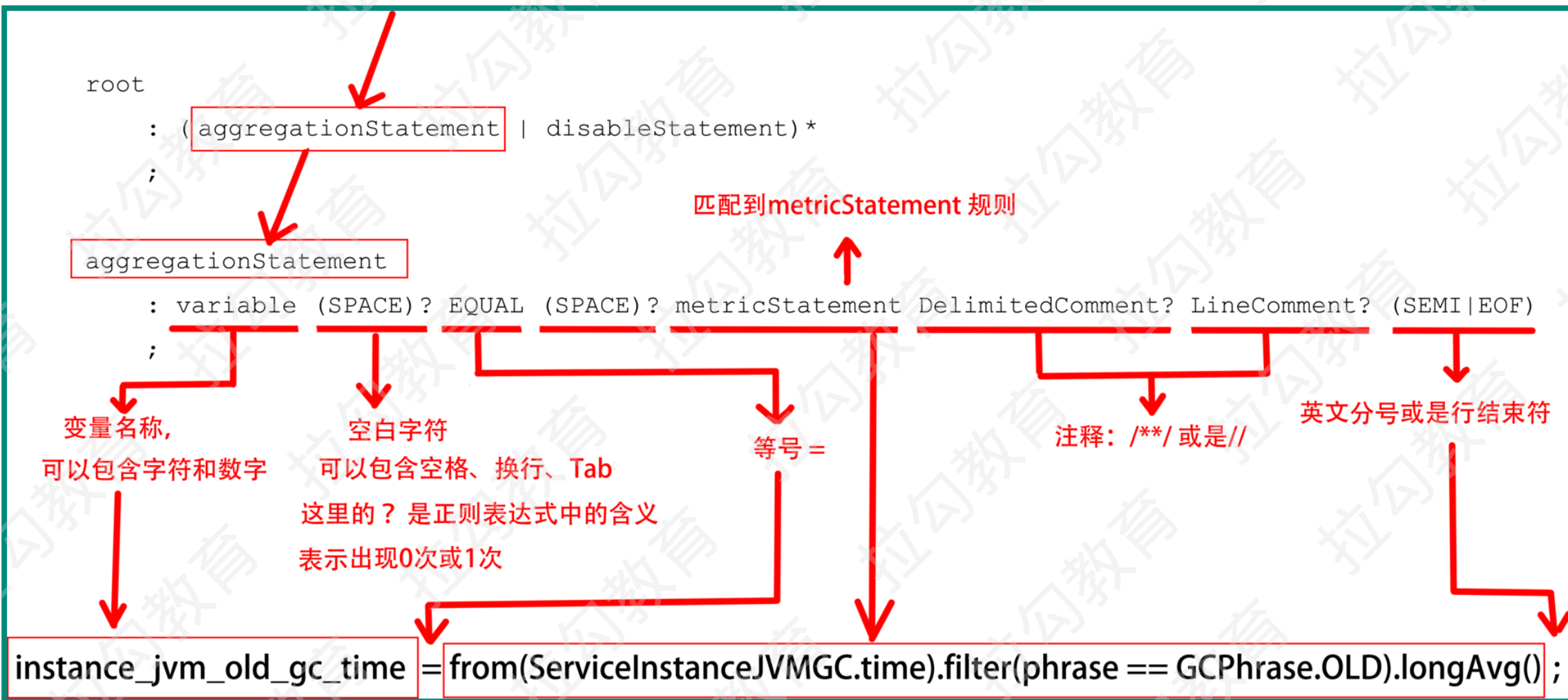
We have these animals:

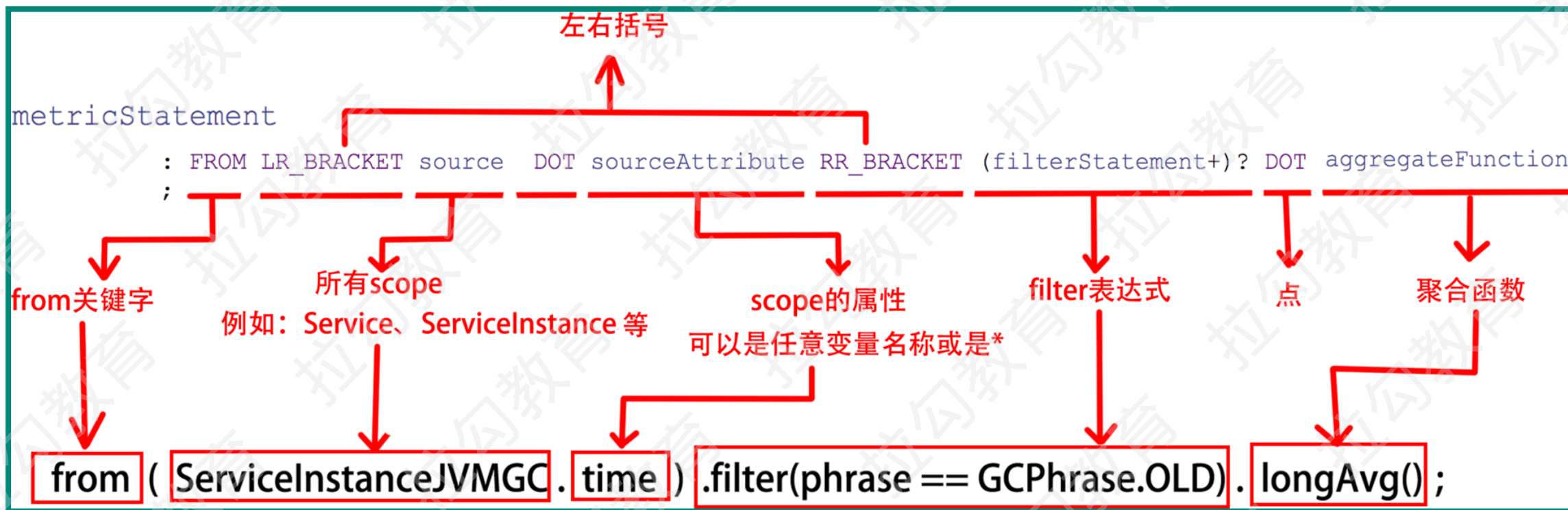
Animal Name	Price	Size
Mouse	20	little
Elephant	10,000	large
Python	1,000	middle

```
instance_jvm_old_gc_time =  
    from(ServiceInstanceJVMGC.time)  
        .filter(phrase == GCPhrase.OLD).longAvg();  
  
service_cpm = from(Service.*).cpm();  
  
service_p99 = from(Service.latency).p99(10);  
  
service_relation_server_cpm =  
    from(ServiceRelation.*)  
        .filter(detectPoint == DetectPoint.SERVER).cpm();
```









Next: 第31讲 《OAL 语言，原来定义创造一门新语言如此轻松（上）》



# 拉勾教育

— 互联网人实战大学 —



关注拉勾「教育公众号」  
获取更多课程信息