


# Skywalking-07：OAL原理——解释器实现

 **Switch** 发布于 8 月 23 日

## OAL 解释器实现

OAL 解释器是基于 **Antlr4** 实现的，我们先来了解下 **Antlr4**

### Antlr4 基本介绍

#### Antlr4 使用案例

参考[Antlr4的使用简介](#)这篇文章，我们实现了一个简单的案例：[antlr案例：简单的计算器](#)，下面来讲讲这个案例。

首先，装好[ANTLR v4\(IDEA插件\)](#)插件，这个之后验证语法树的时候会用到。

在 `pom.xml` 中配置 `antlr4` 的依赖和插件

```
<dependency>
  <groupId>org.antlr</groupId>
  <artifactId>antlr4-runtime</artifactId>
  <version>4.7.1</version>
</dependency>
```

```
<plugin>
  <groupId>org.antlr</groupId>
  <artifactId>antlr4-maven-plugin</artifactId>
  <version>${antlr.version}</version>
  <executions>
    <execution>
      <id>antlr</id>
      <goals>
        <goal>antlr4</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

在 `src/main/antlr4/com/switchvov/antlr/demo/calc` 目录下添加一个 `Calc.g4` 文件

```
grammar Calc;

root : expr EOF;    //解决问题: no viable alternative at input '<EOF>'

expr
  : expr '+' expr    #add    //标签会生成对应访问方法方便我们实现调用逻辑编写
  | expr '-' expr    #sub
  | INT              #int
  ;

INT : [0-9]+         //定义整数
    ;

WS : [ \r\n\t]+ -> skip //跳过空白类字符
    ;
```

执行一下： `mvn compile -Dmaven.test.skip=true` ， 在 `target/generated-sources/antlr4` 会生成相应的 `Java` 代码。

使用方式默认是监听器模式，也可以配置成访问者模式。

监听器模式：主要借助了 `ParseTreeWalker` 这样一个类,相当于是一个 `hook` ， 每经过一个树的节点,便会触发对应节点的方法。好处就算是比较方便,但是灵活性不够,不能够自主性的调用任意节点进行使用。

访问者模式：将每个数据的节点类型高度抽象出来够,根据你传入的上下文类型来判断你想要访问的是哪个节点,触发对应的方法

<font color="red">PS：结论，简单语法监听器模式就可以了，如果语法比较灵活可以考虑使用访问者模式。</font>



继承 `com.switchvov antlr.demo.calc.CalcBaseListener` ， 实现计算器相应功能

```
package com.switchvov.antlr.demo.calc;

import java.util.ArrayDeque;
import java.util.Deque;

/**
 * @author switch
 * @since 2021/6/30
 */
public class CalcExecuteListener extends CalcBaseListener {

    Deque<Integer> queue = new ArrayDeque<>(16);

    @Override
    public void exitInt(CalcParser.IntContext ctx) {
        queue.add(Integer.parseInt(ctx.INT().getText()));
    }

    @Override
    public void exitAdd(CalcParser.AddContext ctx) {
        int r = queue.pop();
        int l = queue.pop();
        queue.add(l + r);
    }

    @Override
```

测试一下

```
package com.switchvov.antlr.demo.calc;

import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CodePointCharStream;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.ParseTreeWalker;
import org.junit.Test;

/**
 * @author switch
 * @since 2021/6/30
 */
public class CalcTest {

    public static int exec(String input) {
        // 读入字符串
        CodePointCharStream cs = CharStreams.fromString(input);
        // 词法解析
        CalcLexer lexer = new CalcLexer(cs);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // 语法解析
        CalcParser parser = new CalcParser(tokens);

        // 监听器触发获取执行结果
```

## Antlr4 IDEA 插件使用

在 Calc.g4 语法定义文件中，鼠标右击可以选择 Test Rule root，然后在 ANTLR Preview 的输入框中填入 1 + 2 就可以校验语法文件是否 OK，并且也可以看到相应的语法树

## Antlr4 在 Skywalking 的应用

通过“ Antlr4 基本介绍”一节，基本上对 Antlr4 使用有了个大概的认识。下面来看看 Skywalking 中 Antlr4 是如何使用的。

### 词法定义

在 oap-server/oal-grammar/src/main/antlr4/org/apache/skywalking/oal/rt/grammar/OALLexer.g4 文件中，我们能看到 OAL 的词法定义

```
// Observability Analysis Language lexer
lexer grammar OALLexer;

@Header {package org.apache.skywalking.oal.rt.grammar;}

// Keywords

FROM: 'from';
FILTER: 'filter';
DISABLE: 'disable';
SRC_ALL: 'All';
SRC_SERVICE: 'Service';
SRC_SERVICE_INSTANCE: 'ServiceInstance';
SRC_ENDPOINT: 'Endpoint';
SRC_SERVICE_RELATION: 'ServiceRelation';
SRC_SERVICE_INSTANCE_RELATION: 'ServiceInstanceRelation';
SRC_ENDPOINT_RELATION: 'EndpointRelation';
SRC_SERVICE_INSTANCE_JVM_CPU: 'ServiceInstanceJVMCPU';
SRC_SERVICE_INSTANCE_JVM_MEMORY: 'ServiceInstanceJVMMemory';
SRC_SERVICE_INSTANCE_JVM_MEMORY_POOL: 'ServiceInstanceJVMMemoryPool';
SRC_SERVICE_INSTANCE_JVM_GC: 'ServiceInstanceJVMGC';
SRC_SERVICE_INSTANCE_JVM_THREAD: 'ServiceInstanceJVMThread';
SRC_SERVICE_INSTANCE_JVM_CLASS: 'ServiceInstanceJVMClass';
SRC_DATABASE_ACCESS: 'DatabaseAccess';
SRC_SERVICE_INSTANCE_CLR_CPU: 'ServiceInstanceCLRCPU';
SRC_SERVICE_INSTANCE_CLR_GC: 'ServiceInstanceCLRGC';
```

## 语法定义

在 `oap-server/oal-grammar/src/main/antlr4/org/apache/skywalking/oal/rt/grammar/OALParser.g4` 文件中，我们能看到 OAL 的语法定义

```
parser grammar OALParser;

@Header {package org.apache.skywalking.oal.rt.grammar;}

options { tokenVocab=OALLexer; }

// Top Level Description

root
    : (aggregationStatement | disableStatement)*
    ;

aggregationStatement
    : variable (SPACE)? EQUAL (SPACE)? metricStatement DelimitedComment? LineComment? (SEMI|EOF)
    ;

disableStatement
    : DISABLE LR_BRACKET disableSource RR_BRACKET DelimitedComment? LineComment? (SEMI|EOF)
    ;

metricStatement
    : FROM LR_BRACKET source (sourceAttributeStmt+) RR_BRACKET (filterStatement+)? DOT aggregateFunction
    ;

filterStatement
```

### Antlr4 生成 Java 代码

在 `oap-server/oal-grammar` 下执行 `mvn compile -Dmaven.test.skip=true` 会在 `oap-server/oal-grammar/target/generated-sources/antlr4` 目录下生成相应的 Java 代码



## 在 Skywalking 的使用

通过“ Antlr4 使用案例”一节，可以知道 Antlr4 有两种功能实现方式：监听器或者访问器。

通过“ Antlr4 生成 Java 代码”一节，知道 Skywalking 使用的是监听器模式。

Skywalking 关于 OAL 的相应的代码都在 oap-server/oal-rt 模块中。

org.apache.skywalking.oal.rt.grammar.OALParserBaseListener 的继承类坐标是  
org.apache.skywalking.oal.rt.parser.OALListener

```
package org.apache.skywalking.oal.rt.parser;

import java.util.Arrays;
import java.util.List;
import org.antlr.v4.runtime.misc.NotNull;
import org.apache.skywalking.oal.rt.grammar.OALParser;
import org.apache.skywalking.oal.rt.grammar.OALParserBaseListener;
import org.apache.skywalking.oap.server.core.source.DefaultScopeDefine;

public class OALListener extends OALParserBaseListener {
    private List<AnalysisResult> results;
    private AnalysisResult current;
    private DisableCollection collection;

    private ConditionExpression conditionExpression;

    private final String sourcePackage;

    public OALListener(OALScripts scripts, String sourcePackage) {
        this.results = scripts.getMetricsStmts();
        this.collection = scripts.getDisableCollection();
        this.sourcePackage = sourcePackage;
    }

    @Override
    public void enterAggregationStatement(@NotNull OALParser.AggregationStatementContext ctx) {
```

简单来说，就是通过监听器封装了个 org.apache.skywalking.oal.rt.parser.OALScripts 对象

```
package org.apache.skywalking.oal.rt.parser;

import java.util.LinkedList;
import java.util.List;
import lombok.Getter;

@Getter
public class OALScripts {
    // 解析出来的分析结果集合
    private List<AnalysisResult> metricsStmts;
    // 禁用表达式集合
    private DisableCollection disableCollection;

    public OALScripts() {
        metricsStmts = new LinkedList<>();
        disableCollection = new DisableCollection();
    }
}
```

org.apache.skywalking.oal.rt.parser.ScriptParser 类读取 oal 文件，使用 Antlr 生成的 Java 类进行解析

```
package org.apache.skywalking.oal.rt.parser;

import java.io.IOException;
import java.io.Reader;
import org antlr.v4.runtime.CharStreams;
import org antlr.v4.runtime.CommonTokenStream;
import org antlr.v4.runtime.tree.ParseTree;
import org antlr.v4.runtime.tree.ParseTreeWalker;
import org.apache.skywalking.oal.rt.grammar.OALLexer;
import org.apache.skywalking.oal.rt.grammar.OALParser;

/**
 * Script reader and parser.
 */
public class ScriptParser {
    private OALLexer lexer;

    private String sourcePackage;

    private ScriptParser() {

    }

    public static ScriptParser createFromFile(Reader scriptReader, String sourcePackage) throws IOException {
        ScriptParser parser = new ScriptParser();
        parser.lexer = new OALLexer(CharStreams.fromReader(scriptReader));
    }
}
```

## 参考文档

- [ANTLR官网](#)
- [antlr4 GitHub](#)
- [antlr4 语案例](#)
- [Antlr4的使用简介](#)
- [antlr案例：简单的计算器](#)
- [某小伙的Antlr4学习笔记](#)
- [ANTLR v4\(IDEA插件\)](#)

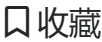
分享并记录所学所见



阅读 71 • 发布于 8 月 23 日



赞



收藏



分享

本作品系原创，采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议



Switch

9 声望

0 粉丝

关注作者

0 条评论

得票数

最新



撰写评论 ...



提交评论

你知道吗？

我没有写过诗，但有人说我写过的代码，像诗一样优雅。

注册登录

继续阅读

聊聊skywalking的lettuce-plugin

skywalking-6.6.0/apm-sniffer/optional-plugins/lettuce-5.x-plugin/src/main/resources/skywalking-plugin.def

codecraft · 阅读 598

聊聊skywalking的rocketmq-plugin

skywalking-6.6.0/apm-sniffer/apm-sdk-plugin/rocketMQ-4.x-plugin/src/main/resources/skywalking-plugin.def

codecraft · 阅读 797

解释器模式

描述：对一个表达式进行解释时，将表达式分为终结符、非终结符、运算环境，这样区分可以把表达式的各个部分独立出来扩展。...

k00baa · 阅读 2.1k

聊聊skywalking的sharding-sphere-plugin

skywalking-6.6.0/apm-sniffer/apm-sdk-plugin/sharding-sphere-4.x-plugin/src/main/resources/skywalking-plugin.def

codecraft · 阅读 751

聊聊skywalking的http-async-client-plugin

skywalking-6.6.0/apm-sniffer/apm-sdk-plugin/httpasyncclient-4.x-plugin/src/main/resources/skywalking-plugin.def

[codecraft](#) · 阅读 563

### 解释器模式

解释器模式(Interpreter Pattern)：定义一个语言的文法，并且建立一个解释器来解释该语言中的句子，这里的“语言”是指使用规定...

[lijingyulee](#) · 阅读 396

### 聊聊skywalking的log4j2-activation

skywalking-6.6.0/apm-sniffer/apm-toolkit-activation/apm-toolkit-log4j-2.x-activation/src/main/resources/skywalking-plugin.def

[codecraft](#) · 阅读 891

### 解释器模式 (Interpreter)

解释器模式 一. 解释器模式 1.1 定义 给定一种语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释...

[JadeQYuan](#) · 阅读 2.7k

#### 产品

热门问答

热门专栏

热门课程

最新活动

技术圈

酷工作

#### 课程

Java 开发课程

PHP 开发课程

Python 开发课程

前端开发课程

移动开发课程

#### 资源

每周精选

用户排行榜

勋章

帮助中心

声望与权限

社区服务中心

建议反馈

#### 合作

关于我们

广告投放

职位发布

讲师招募

联系我们

合作伙伴

#### 关注

产品技术日志

社区运营日志

市场运营日志

团队日志

社区访谈

#### 条款

服务协议

隐私政策

下载 App

Copyright © 2011-2021 SegmentFault. 当前呈现版本 21.09.09

浙ICP备15005796号-2 浙公网安备33010602002000号 ICP 经营许可 浙B2-20201554

杭州堆栈科技有限公司版权所有

