AIQL说明文档

1 简介

满足从简单条件到ES（ElasticSearch）查询语句的翻译。

2 使用介绍

2.1 条件表达式

条件表达式由多个条件通过连接词组合而成。

```
(age <= 18 AND school == "文泽路男子职业技术学院") OR (age == 19 AND home CONTAIN "zhejiang" )
```

注：条件为举例用，忽略现实意义。

2.1.1 表达式连接词

逻辑词	释义	举例	举例释义	优先级
AND	逻辑 “与”	A AND B	条件A与条件B同时满足	1
OR	逻辑“或”	A OR B	条件A满足或条件B满足	2
NOT	逻辑“非”	NOT（A）	不满足条件A	——

####

2.1.2 条件

格式：变量 运算符 值

```
age <= 18
变量:    age
运算符:  <=
值:     18
```

2.1.2.1 变量规范

支持以数字、字母、下划线开头的字符组合。

2.1.2.2 运算符

操作符	释义	举例1	举例2
<	小于	age < 18	createTime < 2022-07-26 08:30:00
>	大于	age > 18	createTime < 2022-07-26 08:30:00
==	等于	a == 1	name == “张三”
<=	小于等于	a <= 18	
>=	大于等于		
!=	不等于	a != 18	
=~	目标字段正则匹配	a =~ /[a-z]/	
CONTAIN	字段包含值中的信息	message CONTAIN "你是个好人"	
NOT_CONTAIN	字段不包含值中的信息	message NOT_CONTAIN "好的"	
IN	匹配字段值属于其中多个值	age IN [18, 30, 45]	name IN ["张三", "罗翔" , "法外狂徒"]

操作符	释义	举例1	举例2
EXIXT	字段存在	destAddress EXIXT	
NOT_EXIST	字段不存在		

2.1.2.3 值类型

类型	释义	举例
ipv4	ipv4地址，支持子网掩码和通配符	192.*.30.2/24
ipv6	ipv6地址，支持子网掩码和通配符	
String	字符串	"你过来啊！"
Number	带符号（带符号int类型）	123，-123
Time	时间	2020-07-26 08:32:00
Regex	正则表达式	/[a-z]/
Array	数组	[1,2,3] 或 ["啦", "啊", "吖"]

1. ipv4 允许通配符（"*"）与子网掩码同时出现,优先匹配高位
2. Time支持2020-07-26 08:32:00、2020-07-26 08:32:00.000两种格式，自动补全。
3. 数组仅用于IN、NOT_IN搭配

2.2 函数表达式

a.group()

对字段a进行聚合操作

2.3 优先级与表达式递归

优先优先级较高

优先最长匹配

优先左结合

	释义	优先级（小值级高）
()	小括号包围的子表达式	0
A AND B	子表达式与关系	1
AOR B	子表达式或关系	2

1. a OR b AND

优先匹配b AND c

```
{
  "query": {
    "bool": {
      "should": [
        {
          "term": {
            "a": {
              "value": 1,
              "boost": 1.0
            }
          }
        },
        {
          "bool": {
            "must": [
              {
                "term": {
                  "b": {
                    "value": 2,
                    "boost": 1.0
                  }
                }
              },
              {
                "term": {
                  "c": {
                    "value": 3,
                    "boost": 1.0
                  }
                }
              }
            ],
            "adjust_pure_negative": true,
            "boost": 1.0
          }
        }
      ],
      "adjust_pure_negative": true,
      "boost": 1.0
    }
  }
}
```

2. (a OR b) AND c

优先匹配小括号 (a OR b)

```
{
  "query": {
    "bool": {
      "must": [
```

```
        {
          "term": {
            "c": {
              "value": 3,
              "boost": 1.0
            }
          }
        },
        {
          "bool": {
            "should": [
              {
                "term": {
                  "a": {
                    "value": 1,
                    "boost": 1.0
                  }
                }
              },
              {
                "term": {
                  "b": {
                    "value": 2,
                    "boost": 1.0
                  }
                }
              }
            ],
            "adjust_pure_negative": true,
            "minimum_should_match": "1",
            "boost": 1.0
          }
        }
      ],
      "adjust_pure_negative": true,
      "boost": 1.0
    }
  }
}
```

3 实现方式

3.1 ANTLR 4 简介

3.2 *.g4 文件

3.2.1 词法分析

词法分析：讲字符聚集为单词或者符号(词法符号，token)。

如关于NUMBER类型的词法符号的描述。

```
NUMBER
    :   '-'? INT '.' INT EXP?   // 1.35, 1.35E-9, 0.3, -4.5
    |   '-'? INT EXP            // 1e10 -3e4
    |   '-'? INT                // -3, 45
    ;
```

3.2.2 语法分析

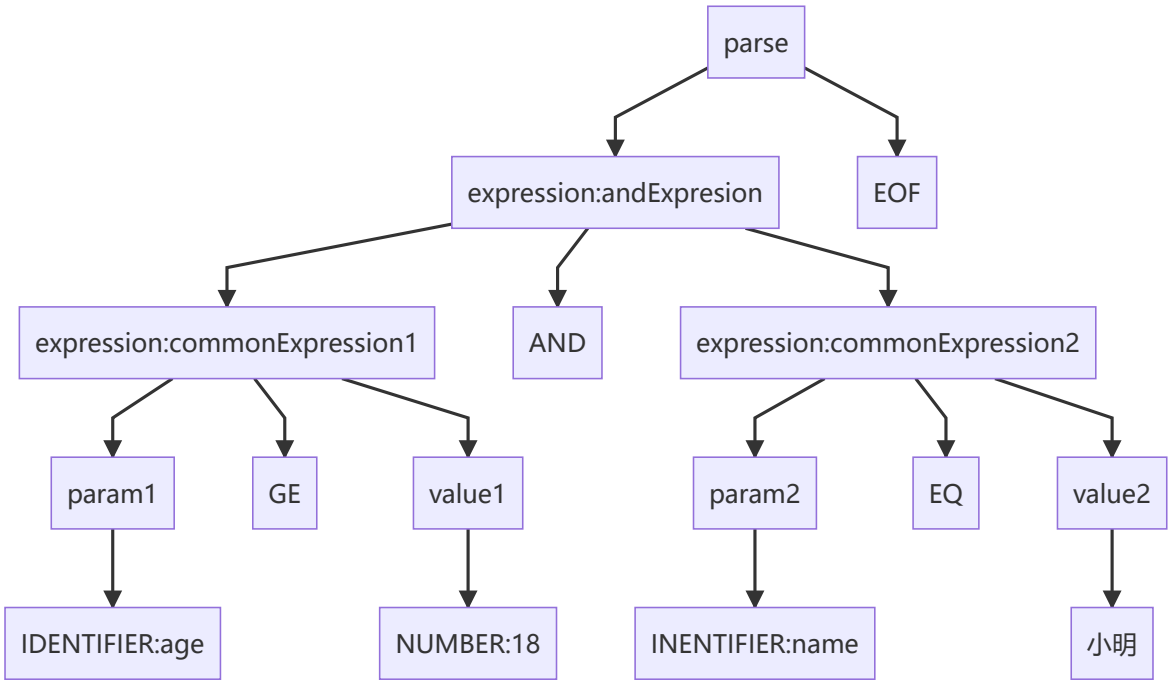
语法分析：消费输入的词法符号来识别语句结构，建造语法分析树(ParseTree)的数据结构。

如关于表达式的语法描述。

```
expression
    :   '('expression')'    #parenExpression
    |   NOT expression      #notExpression
    |   aggexpr             #aggreExpression
    |   expression AND expression   #andExpression
    |   expression OR expression    #orExpression
    |   expr                #commonExpression
    ;
```

3.2.3 语法分析树

age >= 18 AND name == "小明"



注：图中的expression:commonExpression1和expression:commonExpression2 应该同为expression:commonExpression

param1 和 param2 同为param

value1 和 value2 同为value

3.2.4 ANTLR 4 生成文件

*.interp

*.tokens

*BaseListener.java

*BaseVisitor.java

*Lexer.interp

*Lexer.java

*Lexer.tokens

*Listener.java

*Visitor.java

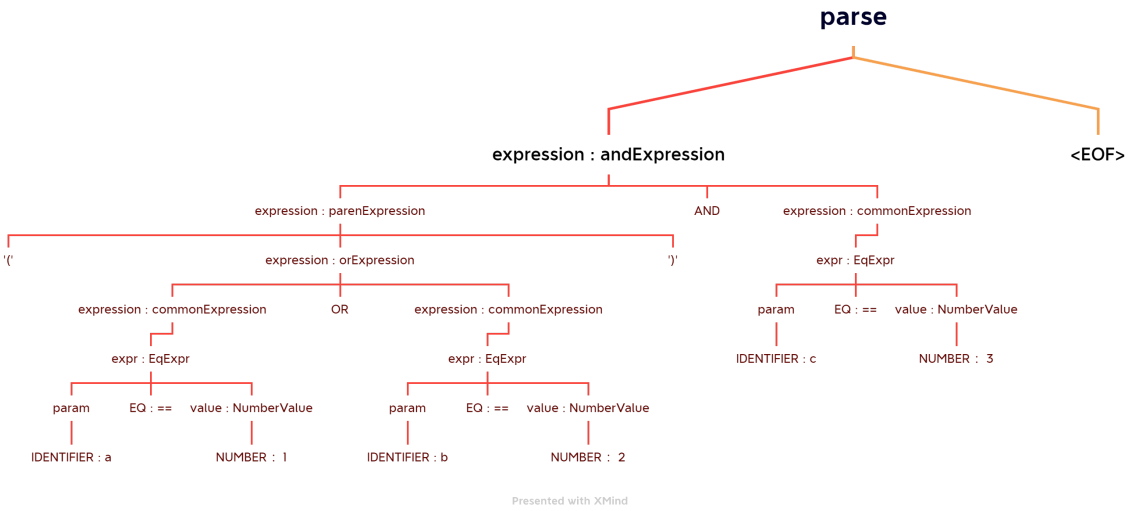
*Parser.java

*Visitor.java

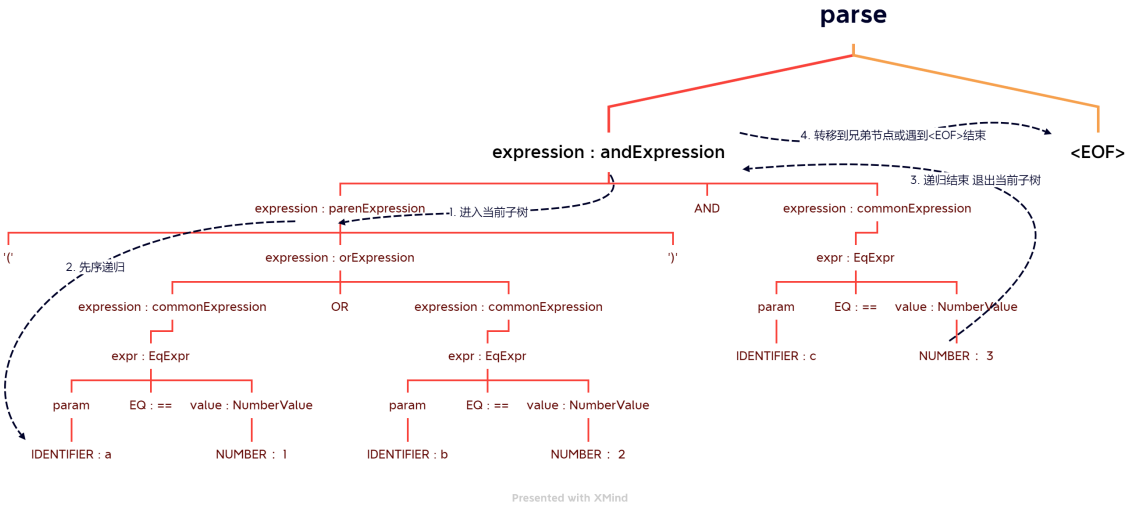
3.3 生成树遍历

(a == 1 OR b == 2) AND c ==3

生成树



遍历顺序



3.3.1 访问者模式

访问者模式相关方法有返回值，即进入到子树时可获取到子树的相关值。

1. 继承 *Visitor.java 接口

   接口部分方法：

```
package com.power.es.gen.boolquery;
import org.antlr.v4.runtime.tree.ParseTreeVisitor;

/**
 * @param <T> The return type of the visit operation. Use {@link Void}
for
 * operations with no return type.
 */
public interface EsInitVisitor<T> extends ParseTreeVisitor<T> {

    T visitParse(EsInitParser.ParseContext ctx);

    T visitRegex(EsInitParser.RegexContext ctx);
}
```

2. 直接对 *BaseVisitor.java 类经行改写。

```
package com.power.es.gen.boolquery;
import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;

/**
 * This class provides an empty implementation of {@link EsInitVisitor},
 * which can be extended to create a visitor which only needs to handle a
subset
 * of the available methods.
 * @param <T> The return type of the visit operation. Use {@link Void}
for
 * operations with no return type.
 */
public class EsInitBaseVisitor<T> extends AbstractParseTreeVisitor<T>
implements EsInitVisitor<T> {

    @Override public T visitParse(EsInitParser.ParseContext ctx) {
        return visitChildren(ctx);
    }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitOrExpression(EsInitParser.OrExpressionContext
ctx) {
        return visitChildren(ctx);
    }
```

3.3.2 监听者模式

针对每个子节点有两个函数：

- 进入子节点（子节点均为遍历）
- 退出子节点（子节点均已遍历）

实现方式：

- 实现*Listener.java接口

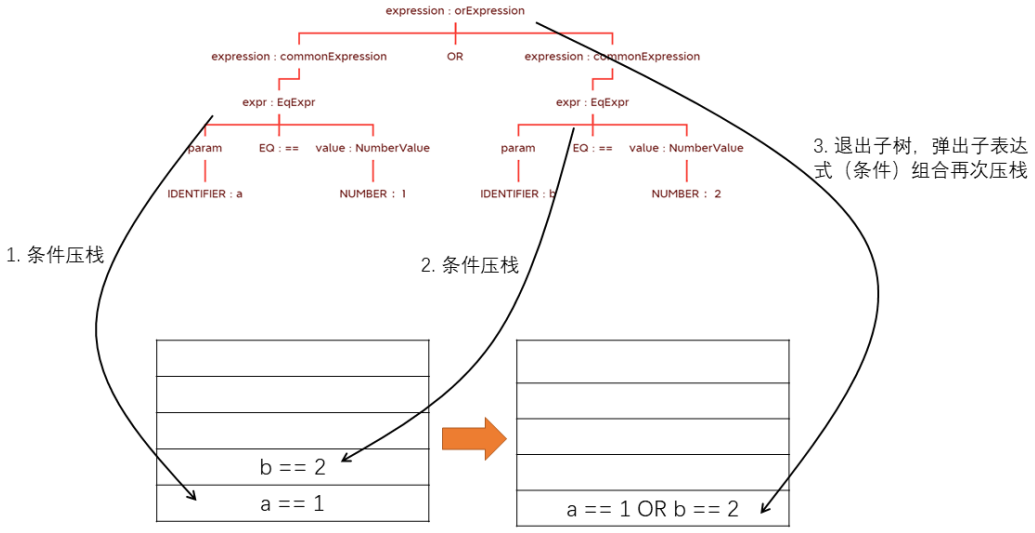
```
package com.power.es.gen.boolquery;
import org.antlr.v4.runtime.tree.ParseTreeListener;

/**
 * This interface defines a complete listener for a parse tree produced
by
 * {@link EsInitParser}.
 */
public interface EsInitListener extends ParseTreeListener {
    /**
     * Enter a parse tree produced by {@link EsInitParser#parse}.
     * @param ctx the parse tree
     */
    void enterParse(EsInitParser.ParseContext ctx);
    /**
     * Exit a parse tree produced by {@link EsInitParser#parse}.
     * @param ctx the parse tree
     */
    void exitParse(EsInitParser.ParseContext ctx);
    /**
     * Enter a parse tree produced by the {@code orExpression}
     * labeled alternative in {@link EsInitParser#expression}.
     * @param ctx the parse tree
     */
}
```

- 修改*BaseListener.java文件

```
package com.power.es.gen.boolquery;

import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.TerminalNode;

/**
 * This class provides an empty implementation of {@link EsInitListener},
 * which can be extended to create a listener which only needs to handle
a subset
 * of the available methods.
 */
public class EsInitBaseListener implements EsInitListener {
    /**
     * {@inheritDoc}
     *
```

```
     * <p>The default implementation does nothing.</p>
     */
    @Override
    public void enterParse(EsInitParser.ParseContext ctx) { }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation does nothing.</p>
     */
    @Override
    public void exitParse(EsInitParser.ParseContext ctx) { }
}
```

监听者遍历方式并没有返回值，但拥有退出访问节点的相关方法，即在退出该子树的根节点时，子树所有节点均已被访问过。可结合栈先进后出的特性来构建查询语句。



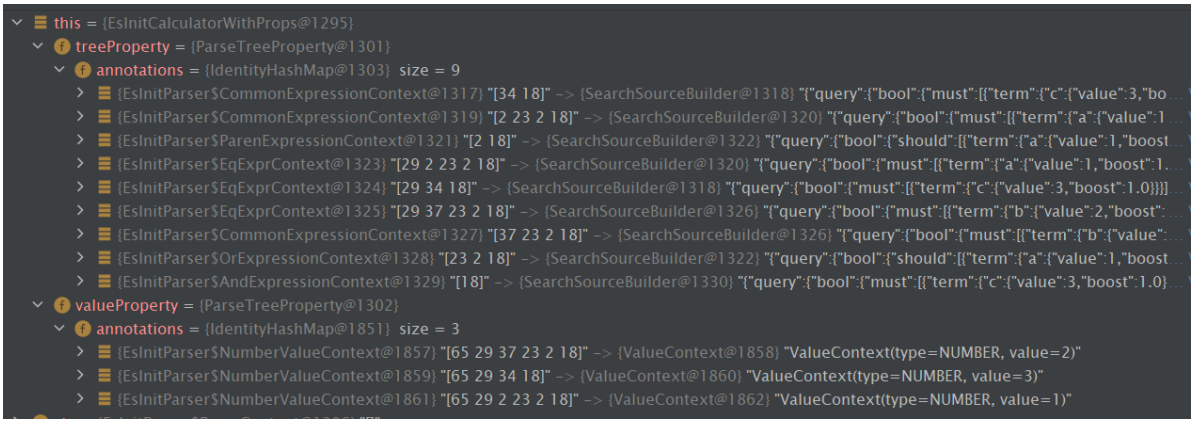
3.3.3 表注语法分析树

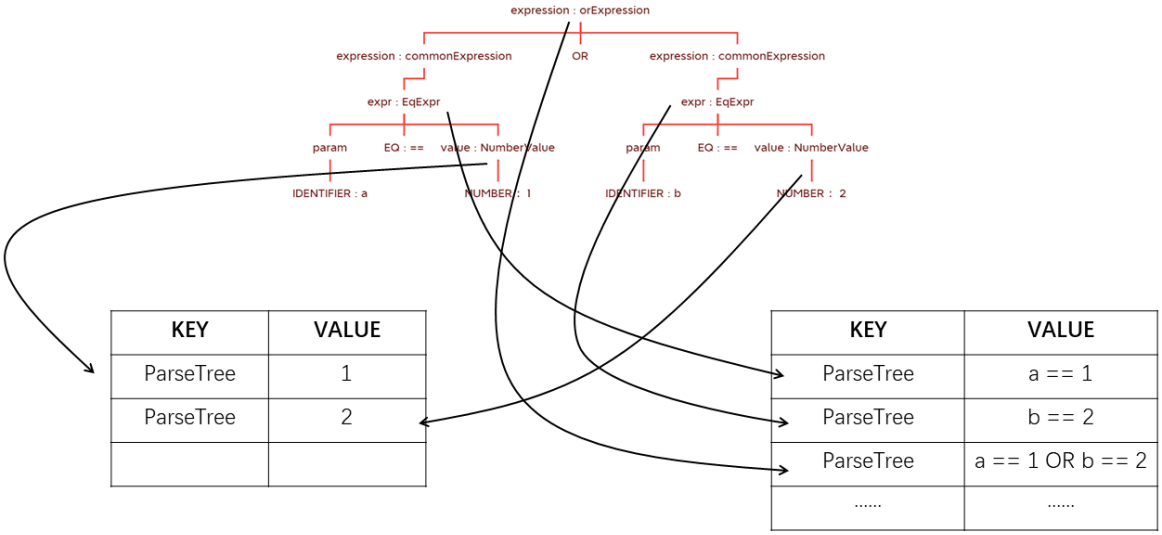
ParseTreeProperty.java

```
/*
 * Copyright (c) 2012-2017 The ANTLR Project. All rights reserved.
 * Use of this file is governed by the BSD 3-clause license that
 * can be found in the LICENSE.txt file in the project root.
 */

package org.antlr.v4.runtime.tree;

import java.util.IdentityHashMap;
import java.util.Map;

/**
 * Associate a property with a parse tree node. Useful with parse tree listeners
 * that need to associate values with particular tree nodes, kind of like
 * specifying a return value for the listener event method that visited a
 * particular node. Example:
 *
```

```
 * <pre>
 * ParseTreeProperty&lt;Integer&gt; values = new
ParseTreeProperty&lt;Integer&gt;();
 * values.put(tree, 36);
 * int x = values.get(tree);
 * values.removeFrom(tree);
 * </pre>
 *
 * You would make one decl (values here) in the listener and use lots of times
 * in your event methods.
 */
public class ParseTreeProperty<V> {
    protected Map<ParseTree, V> annotations = new IdentityHashMap<ParseTree, V>
();

    public V get(ParseTree node) { return annotations.get(node); }
    public void put(ParseTree node, V value) { annotations.put(node, value); }
    public V removeFrom(ParseTree node) { return annotations.remove(node); }
}
```

单独采用Map的形式来存储已经访问过的节点的信息以及各类值。





在遍历树的过程中，通过Map可获取到已存储的子表达式数据从而有利于构建查询语句。