倒排索引和布尔检索

——莎士比亚全集搜索

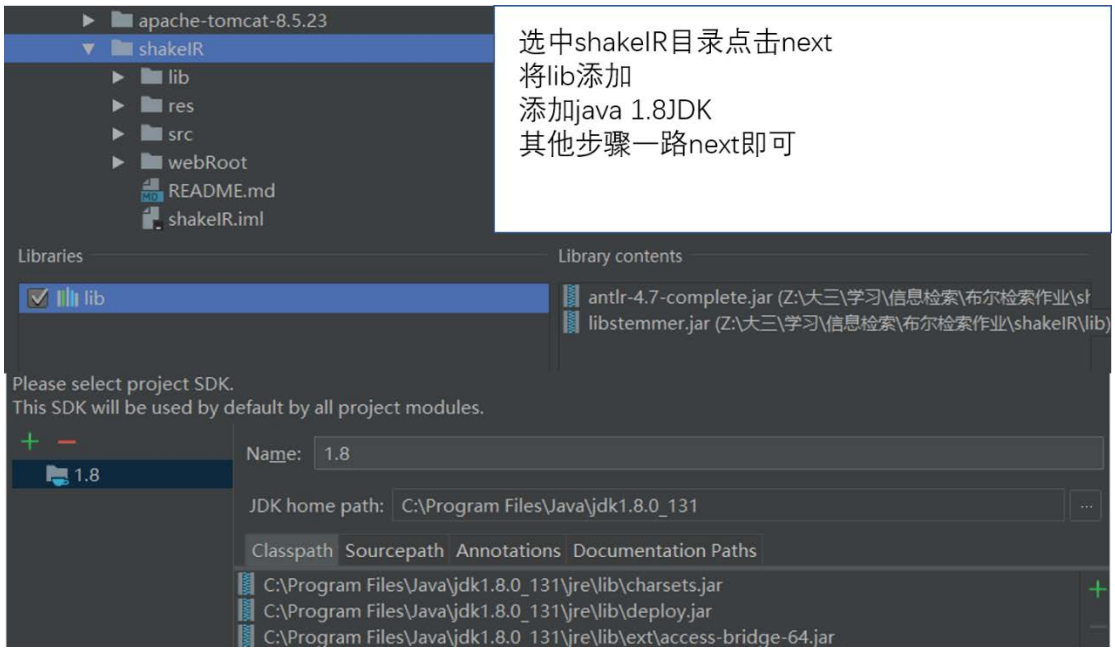
1511274 朱钰颖 计算机科学与技术

一、实验环境

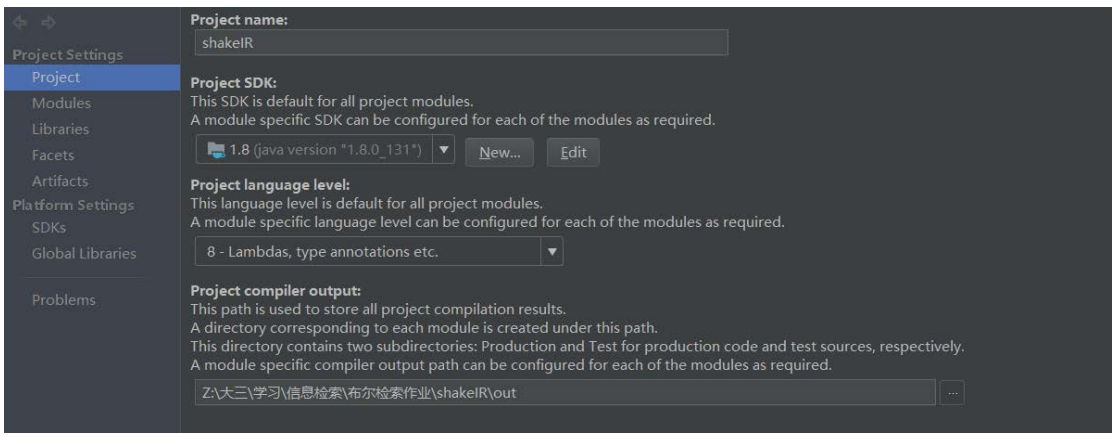
Java 1.8
IDE Intellij IDEA Ultimate 版本
apache-tomcat-8.5.23
antlr-4.7-complete
snowball.stemmer

二、环境配置步骤

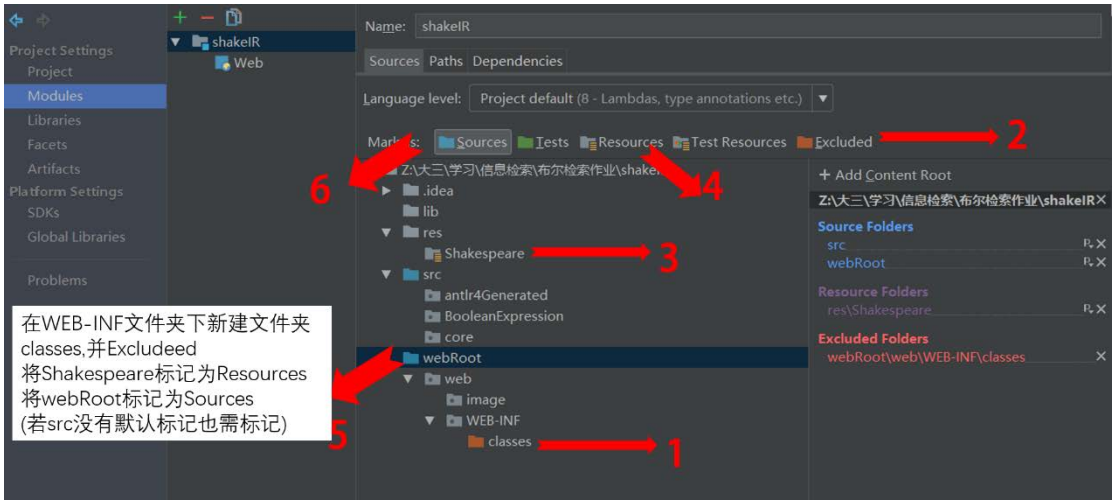
1、打开 intellij 后，import project，初始化步骤如下图所示：



2、成功进入 intellij 后，点击 File->Project Structure
2.1Project 设置如下



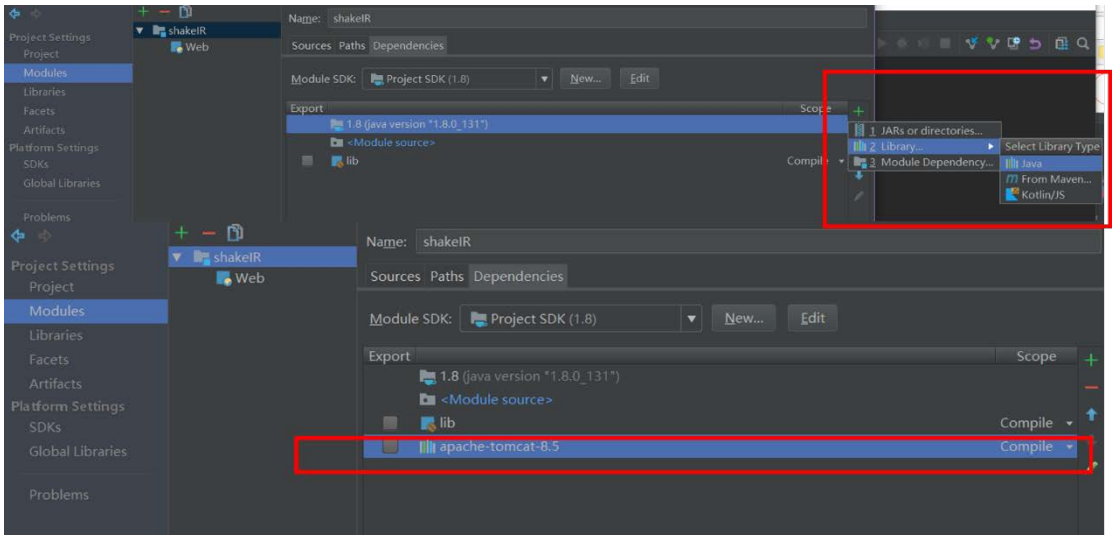
2.2Modules 模块设置如下



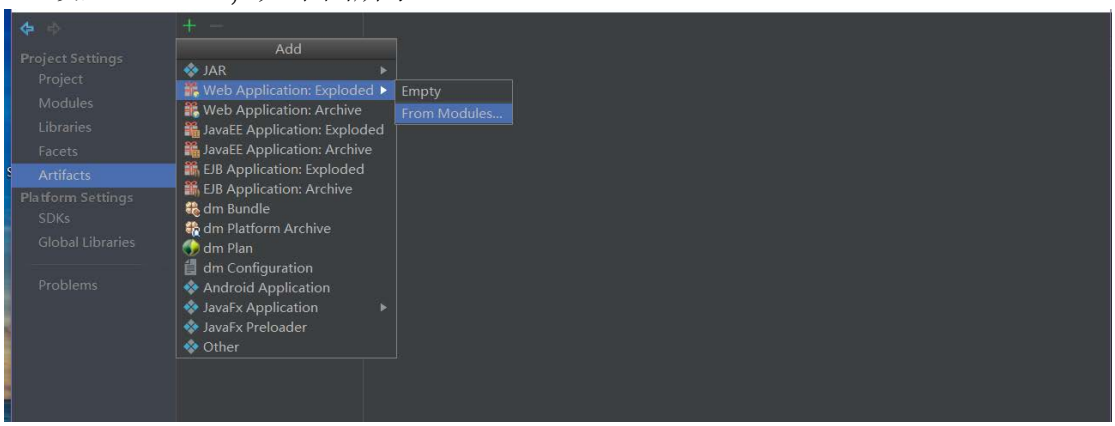
※更正：将 Shakespeare 的上层目录 res 标记为 Resources，而不是 Shakespeare 文件夹

设置 Path

设置 Dependencies

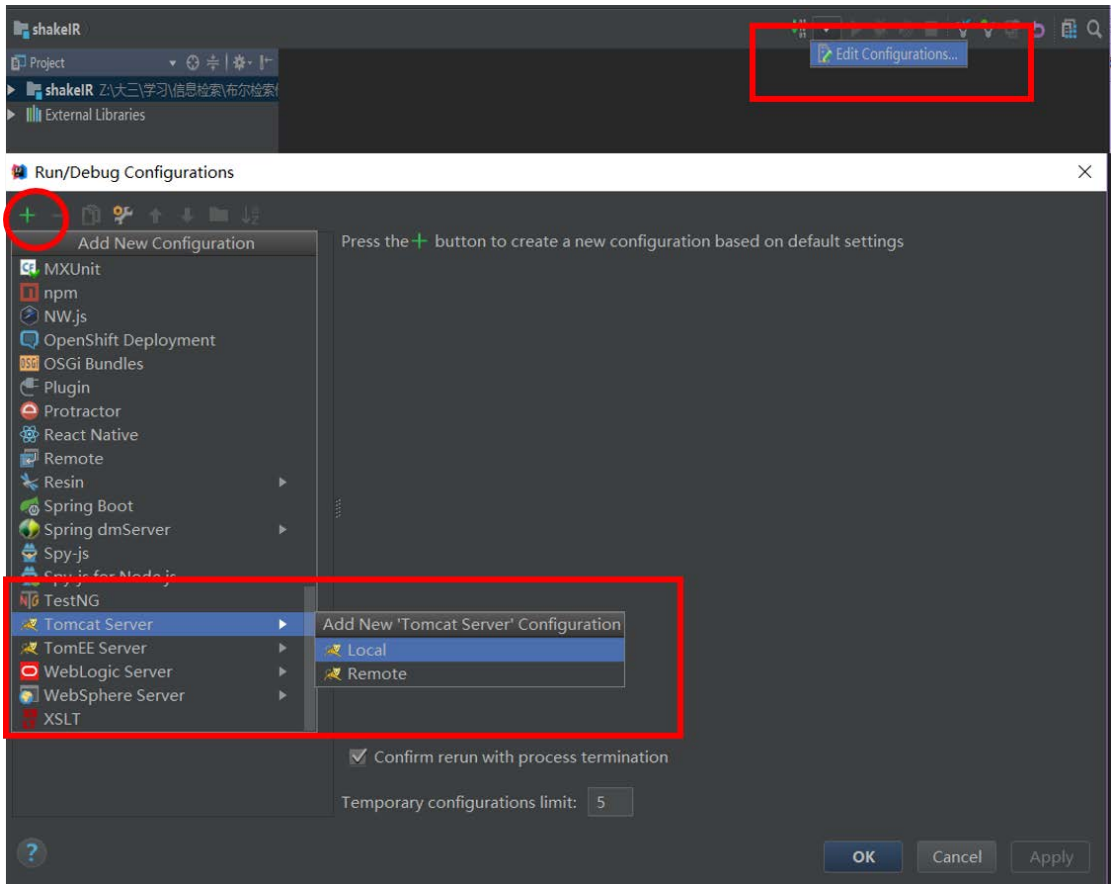


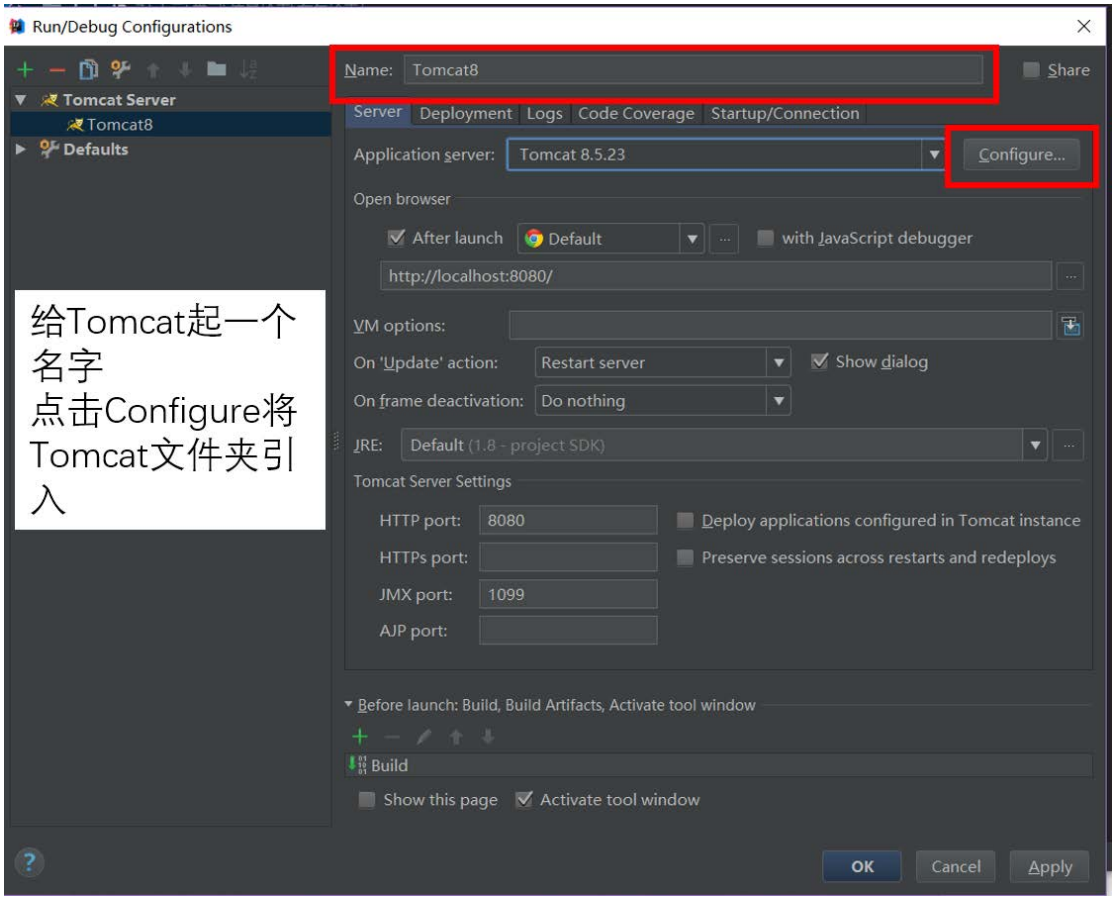
2.3 设置 Artifacts，如下图所示



3、配置 Tomcat

IDEA 右上角点击 Edit Configuration，按照图中所示添加一个本地 Tomcat

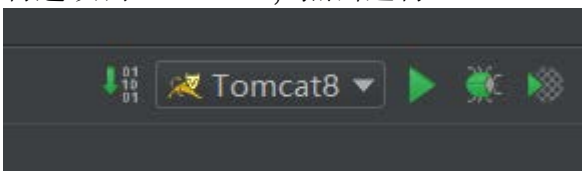




至此，环境配置完毕

三、程序运行

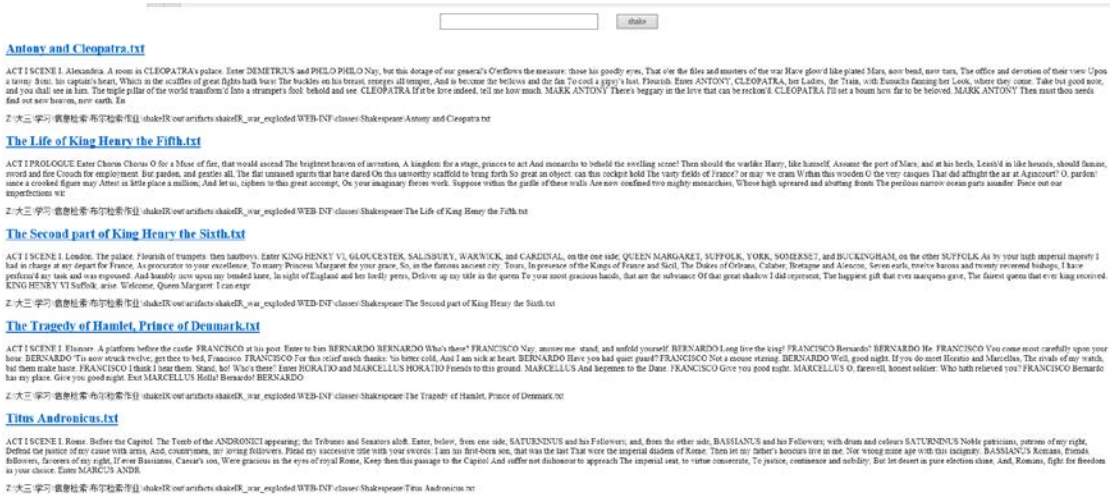
将选项调至 Tomcat，点击运行。



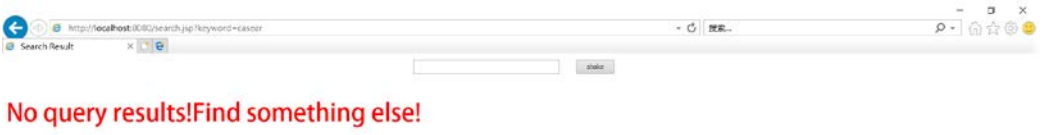
不出意外的话网页会自动跳出，若没有自动跳出，输入 http://localhost:8080 即可打开网页。

页面如下图所示，在底部文本框输入查询语句，并点击 shake 即可查询。

结果界面：输入 **Brutus AND Caesar AND NOT Calpurnia** 得到查询结果如下



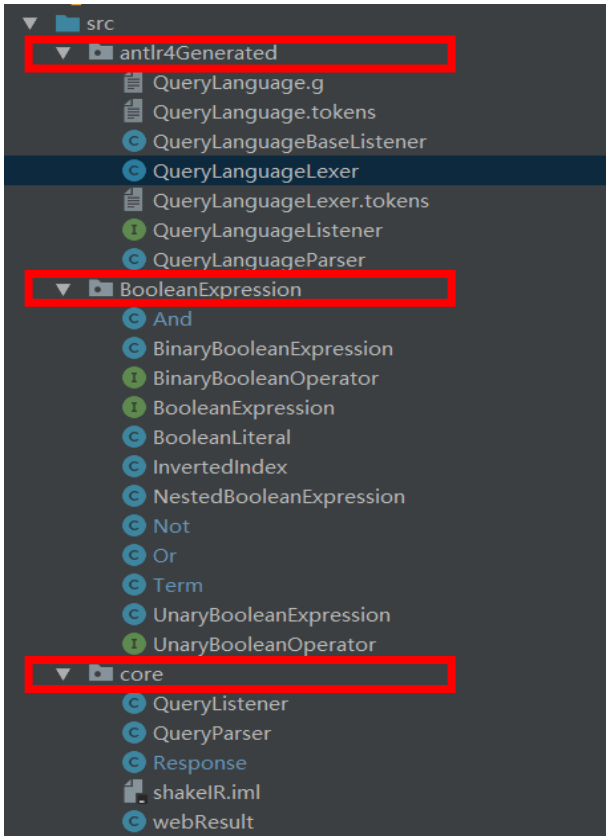
在顶部文本框可继续进行查询，若查询不到则显示以下界面：



四、程序实现

4.1 程序结构

antlr4Generated 文件夹中是编写的语法文件和利用 antlr 语法工具生成的代码。
BooleanExpression 文件夹中是倒排索引和布尔检索所需计算程序
core 是核心代码



4.2 倒排索引

4.2.1 数据结构
InvertedIndex 类中有两个变量 docIndex 和 InvertedIndex
docIndex 记录键值对<文件编号,文件名字>,使用 HashMap
InvertedIndex 记录<词语，词语所带的信息 Info>，使用 TreeMap
Info 是一个内部类，包含词语的一些信息。
在本程序中主要用到的变量是倒排表 posting，其他变量虽有记录但在布尔查询时并未使用（起初是为了进行优化功能所设）。

```
public class InvertedIndex {
    class Info {
        private int docfreq;//how many doc has this word
        private int totalfreq;//word total frequency in all files

        ArrayList<Integer> posting;
        TreeMap<Integer, Integer> posIndex;//docID+word position(last appearance)
        TreeMap<Integer, Integer> wordfreq;//word in each files,<docID,freq>


        public Info() {...}

        public Info(int docfreq, TreeMap<Integer, Integer> posIndex, TreeMap<Integer, Integer> wordfreq) {...}

    }

    public HashMap<Integer, String> docIndex;
    public TreeMap<String, Info> InvertedIndex;
```

4.2.2 方法

getFileIndex 为文件编号，并储存到数据结构中，且写到了 txt 文件中。

getWordFrequency 获得倒排索引表和词语频率

preprocess 利用 snowball 分词工具对文本进行预处理

```
    public void getFileIndex() {...}

    public void getWordFrequency() throws IOException {...}


    private void getWordFrequencyHelp(String wordOfDoc, Integer docID, Integer ByteRecord) {...}

    public static ArrayList<String> preprocess(String[] words) {...}
}
```

4.3 查询语句解析

参考资料：

http://yijun1171.github.io/2015/03/30/ANTLR4 学习笔记-语法字典-Grammar-Lexicon/

https://theendian.com/blog/antlr-4-lexer-parser-and-listener-with-example-grammar/

下载 ANTLR4 包后，编写文法文档.g 文件，使用 **java -jar path/ antlr-4.7-complete.jar /path/xx.g** 进行编译，即可自动生成该文法的解析器。

文法文档如下图所示：

```
grammar QueryLanguage;

@header{
package antlr4Generated;
}

TRUE:'true';
FALSE:'false';
AND:'and';
OR:'or';
NOT:'not';
LPAR:'(';
RPAR:')';

parse:expr EOF;

expr:nestedExpr|unaryExpr|expr binaryOp expr|literalExpr|termExpr;

nestedExpr:LPAR expr RPAR;

unaryExpr:unaryOp expr;

literalExpr:boolean_literal;

termExpr:TERM;

unaryOp:NOT;

binaryOp:AND|OR;

boolean_literal:TRUE|FALSE;

TERM:[a-zA-Z_][a-zA-Z_0_9]*;

WS:((' '|'\r'|'\t'|'\u000C'|'\n')* |EOF)->skip;
```

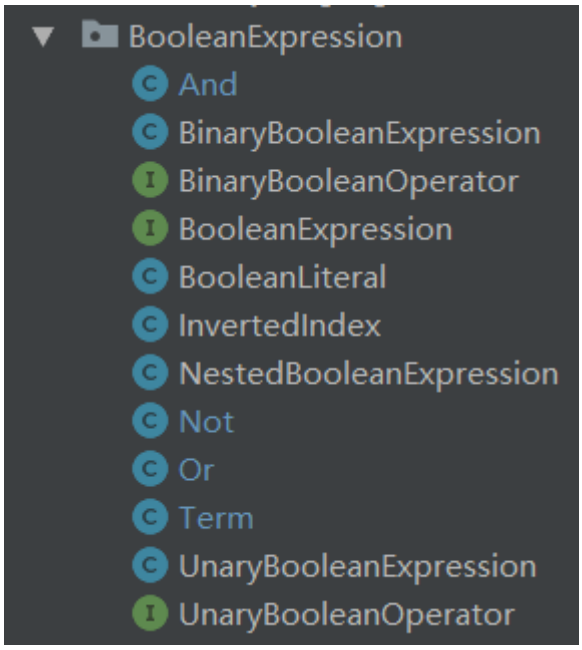
在自动生成的文件中我们只关心 QueryLanguageBaseListener，继承这个文件，重写里面的 enterxx 和 exitxx 方法，编写 QueryListener 类。
编写 QueryParser 类，将 query 传入此类中，由 parse () 函数进行解析，match () 函数进行查询。

```
public class QueryParser {
    private BooleanExpression booleanExpression;

    public BooleanExpression parse(String query){
        QueryLanguageLexer lexer=new QueryLanguageLexer(new ANTLRInputStream(query.toLowerCase()));
        QueryLanguageParser parser=new QueryLanguageParser(new CommonTokenStream(lexer));
        QueryListener queryListener=new QueryListener();
        parser.addParseListener(queryListener);
        parser.parse();
        booleanExpression=queryListener.getOperand();
        System.out.println("PARSED QUERY: " + booleanExpression + "\n");
        return booleanExpression;
    }

    public ArrayList<Integer> match(InvertedIndex invertedIndex){
        if(invertedIndex !=null&&invertedIndex.InvertedIndex.size()>0){
            return booleanExpression.evaluate(invertedIndex);
        }
       return null;
    }
}
```

4.4 布尔查询



三个接口： BooleanExpression，BinaryBooleanOperator，UnaryBooleanOperator 定义布尔表达式，二元运算和一元运算。其他类实现这三个接口。

三个基本布尔逻辑运算实现：

And

如果返回 list 含有-1（没有在文档集中找到此词），继续向上层返回-1

通过两个变量 i，j 对两个 list 进行合并

```java
public ArrayList<Integer> evaluate(BooleanExpression left, BooleanExpression right, InvertedIndex invertedIndex) {
    ArrayList<Integer> leftrtn = left.evaluate(invertedIndex);
    ArrayList<Integer> rightrtn = right.evaluate(invertedIndex);
    ArrayList<Integer> rtn = new ArrayList<>();
    /*have nonexistent word*/
    if (leftrtn.contains(-1) || rightrtn.contains(-1)) {
        rtn.add(-1);
    } else {

        int i = 0;
        int j = 0;

        while (i < leftrtn.size() && j < rightrtn.size()) {
            if (leftrtn.get(i) == rightrtn.get(j)) {
                rtn.add(leftrtn.get(i));
                i++;
                j++;
            } else if (leftrtn.get(i) < rightrtn.get(j)) {
                i++;
            } else {
                j++;
            }
        }
    }
    return rtn;
}
```

Or

同样考虑含有-1 的情况，通常返回两个 list 的并集。

```java
public ArrayList<Integer> evaluate(BooleanExpression left, BooleanExpression right, InvertedIndex invertedIndex) {
    ArrayList<Integer> leftrtn = left.evaluate(invertedIndex);
    ArrayList<Integer> rightrtn = right.evaluate(invertedIndex);
    ArrayList<Integer> rtn = new ArrayList<>();
/*have nonexistent word*/
    if (leftrtn.contains(-1)) {
        return rightrtn;
    } else if (rightrtn.contains(-1)) {
        return leftrtn;
    } else if (leftrtn.contains(-1) && rightrtn.contains(-1)) {
        rtn.add(-1);
        return rtn;
    } else {
        int i = 0;
        int j = 0;

        while (i < leftrtn.size() && j < rightrtn.size()) {
            if (leftrtn.get(i) == rightrtn.get(j)) {
                rtn.add(leftrtn.get(i));
                i++;
                j++;
            } else if (leftrtn.get(i) < rightrtn.get(j)) {
                rtn.add(leftrtn.get(i));
                i++;
            } else {
                rtn.add(rightrtn.get(j));
                j++;
            }
        }
        if (i < leftrtn.size()) {
            for (; i < leftrtn.size(); i++) {
                rtn.add(leftrtn.get(i));
            }
        }
        if (j < rightrtn.size()) {
            for (; j < rightrtn.size(); j++) {
                rtn.add(rightrtn.get(j));
            }
        }
        return rtn;
    }
```

Not

取全集的补集

```java
public ArrayList<Integer> evaluate(BooleanExpression booleanExpression, InvertedIndex invertedIndex) {

    ArrayList<Integer> posting = booleanExpression.evaluate(invertedIndex);
    ArrayList<Integer> rtn = new ArrayList<>();

    int flag = invertedIndex.docIndex.size();
    if (posting.contains(-1)) {
        for (int i = 0; i < flag; i++) {
            rtn.add(i);
        }
    } else {
        for (int i = 0; i < flag; i++) {
            if (posting.contains(i)) {
                continue;
            } else {
                rtn.add(i);
            }
        }
    }
    return rtn;
}
```

五、程序改进

1、没有对索引进行压缩

2、没有对搜索出的文档进行排序

3、网页搜索结果没有链接到文档本身，即无法打开链接结果

4、进行布尔逻辑运算时，没有首先判断大小，进行运算顺序的优化