词法分析器的完整设计

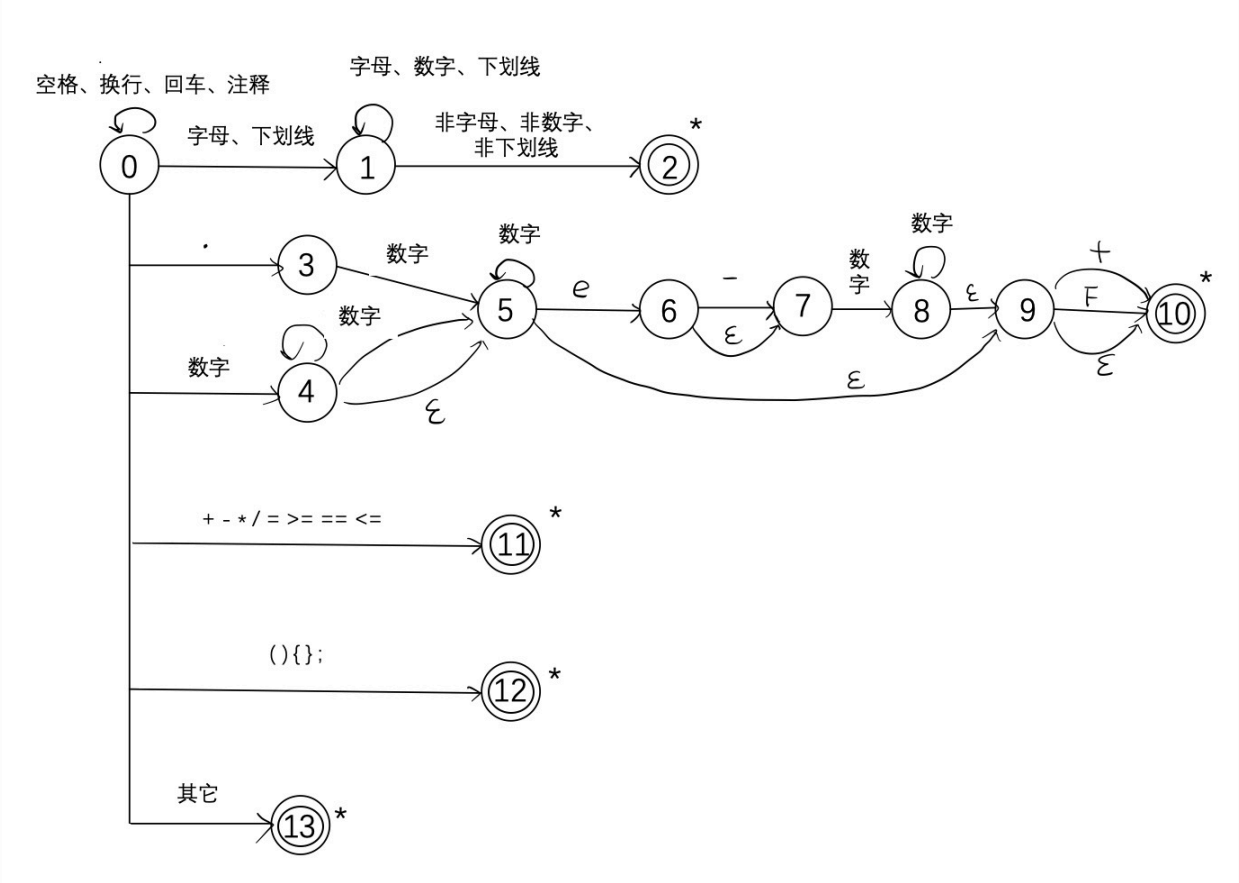
Java 词法分析器 状态转换图

1. 开发环境

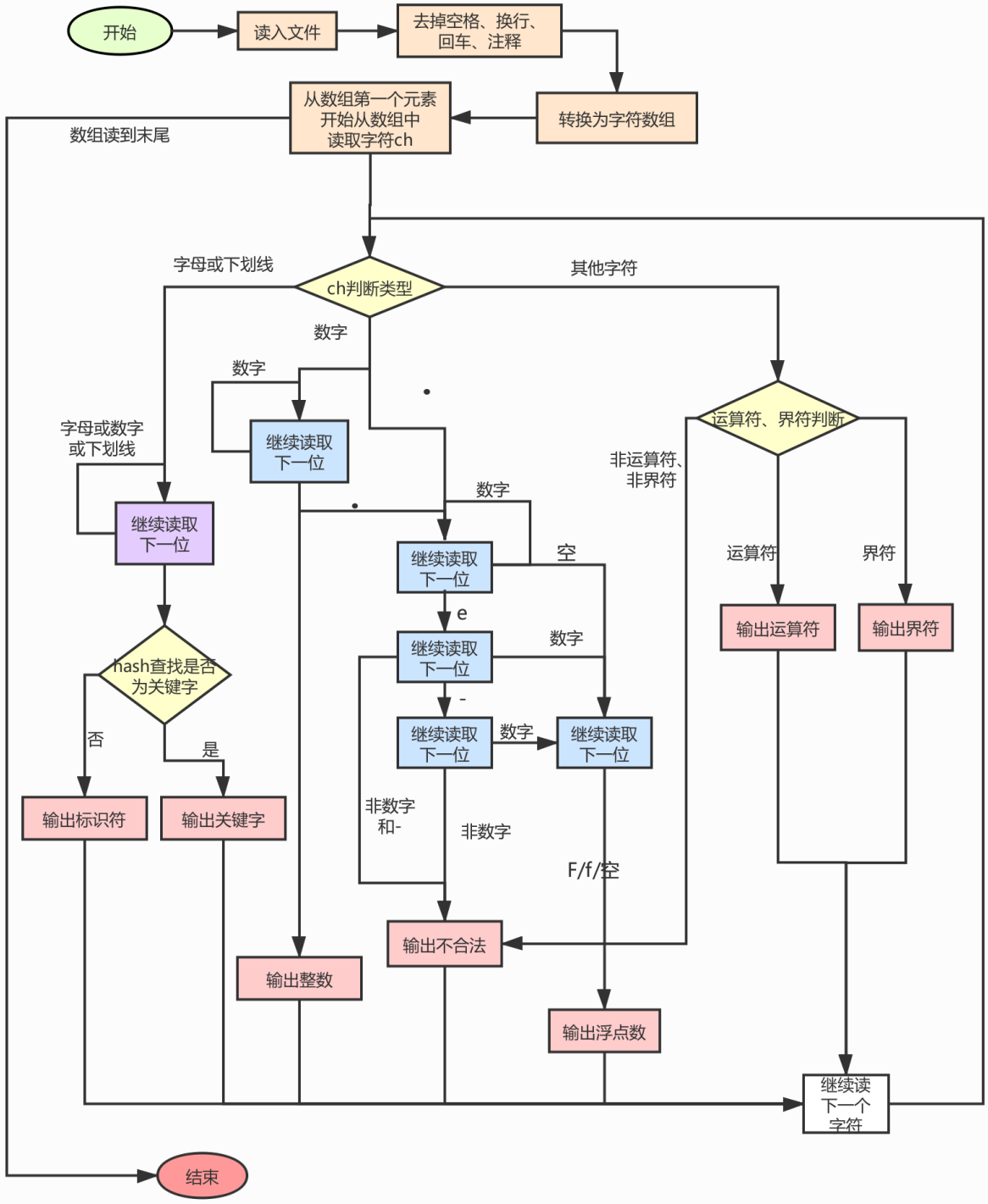
- 编程语言：Java8
- IDE：IntelliJ IDEA
- 版本管理：git + GitHub
- 仓库地址：https://github.com/chenshuyuhhh/LexicalAnalyzer.git

未使用第三方库

2. 状态转换图



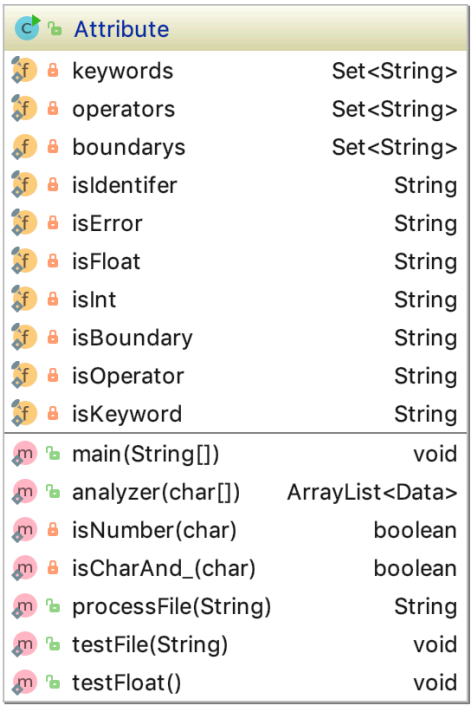
3. 程序核心流程图


开始
读入文件
去掉空格、换行、回车、注释
转换为字符数组
从数组第一个元素开始从数组中读取字符ch
数组读到末尾
结束
ch判断类型
字母或下划线
其他字符
数字
数字
字母或数字或下划线
继续读取下一位
继续读取下一位
数字
空
e
数字
-
数字
继续读取下一位
继续读取下一位
继续读取下一位
继续读取下一位
hash查找是否为关键字
否
是
输出标识符
输出关键字
非数字和-
非数字
F/f/空
输出不合法
输出整数
输出浮点数
运算符、界符判断
非运算符、非界符
运算符
界符
输出运算符
输出界符
继续读下一个字符

4. 词法分析器的程序设计

关键类为 `Attribute` 。

- **analyzer**静态方法根据字符数组进行词法分析并返回分析结果
- **testFile**根据文件名，对文件内容进行词法分析并打印出分析结果。
- **processFile** 过滤原始的文件内容为字符串。
- 使用**hashSet** 存关键字、界符、算符，便于后续查找判断。



3.1读取文件

根据原文件路径按字节读入文件内容，将整个文件内容转化为字符串。

```
InputStream is = new FileInputStream(file);
            int iAvail = is.available();
            byte[] bytes = new byte[iAvail];
            is.read(bytes);
            // 获取文件内容
            String content = new String(bytes);
```

3.2 过滤内容

利用字符串分割过滤掉原内容中的“/*..*/”、“//”注释以及回车符和换行符。没有清除空格，因为空格可以帮助断句。

`特殊情况` 当注释"/* */没有成对时，将/*后面紧跟着的所有*/作为注释内容，比如/*abc*/def*/g*/hijk*/，则abcdefghijk均为注释内容。

```
public static String processFile(String content) {
        //去掉/* */型注释
        String[] strs = content.split("/\\*");
        StringBuilder newContent = new StringBuilder(strs[0]);
        for (int i = 1; i < strs.length; i++) {
            String[] temp = strs[i].split("\\*/");
            // 只有第一个*/ 算
            for (int j = 1; j < temp.length; j++) {
                newContent.append(temp[j]);
            }

        }

        // 去掉//型注释，去掉空格,换行
        return newContent.toString().
                replaceAll("//.*\r\n", "").
                        replaceAll("\n", " ").
                        replaceAll("\r", " ");
}
```

3.3 字符级别词法分析

1. 构造Data类记录分析结果

```
public class Data {
    private String content;// 字符串内容
    private String kind; // 字符串类型

    public Data(String content, String kind) {
        this.content = content;
        this.kind = kind;
    }

    public String getContent() {
        return content;
    }

    public String getKind() {
        return kind;
    }
}
```

2. 将过滤后的字符串转化为字符数组，依次遍历分析每个字符。从数组中的第一个字符开始，作为判断的激活字符，根据不同情况往后连续读取，分析出一个词法之后，将连续读取的字符串的下一个字符作为激活字符，继续分情况分析，这样不断遍历整个数组。可分为以下几种情况。

- 数字判断 —— 浮点数和整数，其中浮点数的判断最为复杂。

> float的表现形式，
>
> - 指数形式，如3.45e6，.7e-19
> - 十进制小数形式，如1.08，.98，18.；
> - 最后加f/F，只有带前面两种形式的才能加这个尾缀
>
> 10f 、10d、10.0d 中的f、d算词法错误
>
> float a = 10 这个10算int

```
 if (isNumber(c) || c == '.') { // 如果是数字，可能是整数或者浮点数
                // 浮点数.号前后的数字部分，至少有一个部分有数字
                boolean beforedian = false; // 小数点前是否有数字，有数字
则true
                boolean afterdian = false; // 小数点后是否有数字

                boolean hase = false; // 是否有e
                boolean aftere = true; // e后面应当有数字,默认合法
                boolean hasDian = false;// 是否有小数点

                // 浮点数小数点前面的数字部分
                if (isNumber(c)) {
                    // 如果c是一个数字，小数点前面肯定有数字
                    beforedian = true;
                    // 继续往后找数字
                    while (true) {
                        if (++i >= ln) break; // 如果越界，直接跳出循环
                        c = chars[i]; // 没有越界，向后访问
                        if (isNumber(c)) {
                            // 是数字再加入字符串
                            str.append(c);
                        } else {
                            // 不是数字跳出循环
                            break;
                        }
                    }
                }
                // 浮点数.
                // 保证“.”前或者后必然有数字
                // 这个c可能是第一次读到的，也可能是刚开始是数字，往后读
                if (c == '.') {
                    if (beforedian) str.append(c);
                    hasDian = true;
                    // 往后继续识别数字，可能没有
                    while (true) {
                        if (++i >= ln) break; // 如果越界，直接跳出循环
                        c = chars[i]; // 没有越界，向后访问
                        if (isNumber(c)) {
                            afterdian = true;
```

```
                    // 是数字再加入字符串
                    str.append(c);
                } else {
                    // 不是数字跳出循环
                    break;
                }
            }
            // 回退
        }

        // 浮点数应当考虑.指数形式，如3.45e6
        // 这个c=='e' 可能是经过了第二个if (c == '.') , 也没有经过
小数点
        // 如果没有经过小数点，需要前面有数字
        if ((beforedian || afterdian) && c == 'e') {
            str.append(c);
            hase = true;
            // 往后继续识别数字，有才合法
            boolean judge = false; // 标示是否合法，默认不合法
            if (++i < ln) {
                c = chars[i];
            }
            if (c == '-') {
                // 指数可能是负数，就往后读
                str.append(c);
            } else {
                --i;
            }
            while (true) {
                // 往后识别数字
                if (++i >= ln) break; // 如果越界，直接跳出循环
                c = chars[i]; // 没有越界，向后访问
                if (isNumber(c)) {
                    // e后面是否合法
                    judge = true;
                    // 是数字再加入字符串
                    str.append(c);
                } else {
                    // 不是数字跳出循环
                    break;
                }
            }
            if (!judge) {
                aftere = false;
            }
            // 数字读取完毕，回退
        }

        // 没有点，有数字，没有e，则是整数
        if (!hasDian && beforedian && !hase) {
            result.add(new Data(str.toString(), isInt));
        }
```

```
                    // 保证e的合法性的前提下
                    // 小数点后面有数字也可以
                    // 小数点后面没有数字，则需要前面有数字和小数点
                    // 小数点后面没有数字，也没有小数点，则需要有e
                    else if (aftere && (afterdian || (beforedian &&
hasDian) || (beforedian && hase))) {
                        if ((c == 'f' || c == 'F')) {
                            str.append(c);
                            ++i;
                        }
                        result.add(new Data(str.toString(), isFloat));
                    } else {
                        result.add(new Data(str.toString(), isError));
                    }
                    // 整体回退一格，便于进行下一个字符识别
                    --i;
                }
```

- 关键字和标识符

读到的开始字符为字母或下划线，此后连续读到字母、下划线、或者数字。

```
else if (isCharAnd_(c)) {

                    while (true) {
                        if (++i >= ln) {
                            --i;
                            break;
                        } // 越界直接跳出
                        c = chars[i];
                        // 如果是字母(A-Z,a-z)、数字(0-9)、下划线“_”
                        if (isCharAnd_(c) || isNumber(c)) {
                            str.append(c);
                        } else {
                            --i;
                            break;
                        }
                    }
                    String temp = str.toString();
                    if (keywords.contains(temp)) {
                        result.add(new Data(temp, isKeyword));
                    } else {
                        result.add(new Data(temp, isIdentifer));
                    }
 }
```

- 算符和界符

读取一个字符，先判断是否为算符，如果不是再判断是否为界符。界符中的">="、"<="、"=="、"="特殊，如果该字符为'>'、'<'、'='应当再往后读一位判断

```
else if (c != ' ') {
                // 先判断运算符
                // 先从特殊的开始判断
                if (c == '=' || c == '>' || c == '<') {
                    // special: ">=" "==" "<="
                    if (++i < ln) {
                        // 如果能往后取一位
                        c = chars[i];
                        // 是不是=，因为有==运算符和=运算符
                        if (c == '=') {// >=,<=,==
                            str.append(c);
                            result.add(new Data(str.toString(),
isOperator));
                            ++i; // 实际往后取了，需要加上去
                        } else if (chars[i - 1] == '=') { // i-1为上一
位
                            // = 运算符，不是==
                            result.add(new Data(str.toString(),
isOperator));
                        } else {
                            // 不是>=,<= ,只有>或者<
                            result.add(new Data(str.toString(),
isError));
                        }
                        --i; // 再减回去
                    } else {
                        // 不能就直接是=
                        result.add(new Data(str.toString(),
isOperator));
                    }
                } else if (operators.contains("" + c)) {//别的操作符，
均为一位，可直接判
                    result.add(new Data(str.toString(), isOperator));
                }
                // 不是算符，就可能是界符
                else if (boundarys.contains("" + c)) {
                    result.add(new Data(str.toString(), isBoundary));
                } else {
                    result.add(new Data(str.toString(), isError));
                }
            }
```

- 错误

在上述判断过程中，若出现非规定字符，则为错误。

3.4 输出

在判断过程中将切割出来的字符串和字符串对应的类型实例化为data，存入数组中，遍历整个数组打印出分析结果。同时应当关闭文件流。

```
ArrayList<Data> datas = analyzer(content.toCharArray());
for (Data data : datas) {
    System.out.println(data.getContent() + ": " + data.getKind());
}
System.out.println();
is.close();
```

作业要求中的测试用例样本输出结果

```
void: keyword
main: keyword
(: boundary
): boundary
{: boundary
int: keyword
i: identifier
=: operator
2: integer
;: boundary
int: keyword
j: identifier
=: operator
4: integer
;: boundary
int: keyword
result: identifier
=: operator
0: integer
;: boundary
while: keyword
(: boundary
i: identifier
<=: operator
j: identifier
): boundary
{: boundary
result: identifier
=: operator
result: identifier
+: operator
j: identifier
;: boundary
i: identifier
+: operator
+: operator
```

```
;: boundary
}: boundary
printf: identifier
(: boundary
): boundary
;: boundary
return: keyword
result: identifier
;: boundary
}: boundary
```