JFLEX词法分析

# 安装与配置

1. 下载**jflex-1.4.3.zip**,解压缩到本地目录（c:/jflex）。
2. 找到**jflex\bin\jflex.bat**文件,配置**JAVA HOME**和**JFLEX HOME**
3. 把**x:\jflex\bin**写入系统环境变量path中

# 运行

## 可视化方式

直接运行jflex\bin\jflex.bat文件，打开可视化界面操作即可。

## 命令行方式

### 配置

把x:\jflex\bin以及x:\jflex\lib\ JFlex.jar配置到系统环境变量的CLASSPATH中。

### 格式

java JFlex.Main <options> <inputfiles>

### 运行参数

#### -d <directory>

<directory>生成文件的输出目录

#### --skel <file>

使用外部的骨架文件生成扫描器类，它大多数情况下用于JFLEX的维护和低级别定制。只有在你知道自己正在作什么时候才使用它。JFLEX的源码中带有一个骨架文件，预先编写骨架文件才能使用此命令。

#### --nomin

在扫描器生成的过程中，跳过DFA简化步骤。

#### --jlex

完全兼容jlex

#### --dot

为NFA, DFA and minimised DFA生成扩展名为.dot的graphviz图型文件。该参数还在最初阶段，尚未完全实现。

#### --dump

在控制台显示NFA转换表，初始DFA和最简DFA。

#### --verbose or –v

显示生成过程信息。

#### --quiet or –q

仅显示生成错误信息

#### --time

显示代码生成耗时信息（不十分精确）

#### --version

打印JFLex版本号

#### --info

打印系统以及JDK信息。

#### --pack

使用%pack代码生成策略

#### --table

使用%table代码生成策略

#### --switch

使用%switch代码生成策略

#### --help or -h

打印帮助信息，解释运行参数以及Jflex用法。

# JFLEX 配置文件编写

配置文件以.flex为扩展名，整个文档分为三个部分，使用%%划分

1. 用户代码
2. 选项与声明
3. 词法规则

形式形如：

|  |
| --- |
| 用户代码  …………………………………………………………………………….  …………………………………………………………………………….  %%  选项与声明  …………………………………………………………………………….  ……………………………………………………………………………..  %%  词法规则  ……………………………………………………………………………….. |

## 用户代码

JFLEX直接将这部分代码拷贝到生成词法分析器Java源文件中，通常在这里我们只定义一些类注释信息以及package和import的引用。

## 选项与声明

在这一部分，**选项**用来定制词法分析器，**声明**则是声明一些能够在第三部分(词法规则定义)使用的宏定义和词法状态，其中宏大多由正则表达式定义。

### 选项

所有选项都要由一个“%”符号开头，下面来列举一下所有的选项：

#### 类选项和用户代码

##### %class

定义生成词法分析器Java文件的文件名，如果不定义该选项，则默认生成 ”Yylex.java”。

**例子： %class MyScanner**

##### %implements

使得生成的词法分析器类实现特定的接口，可以同时实现多个接口。

**例子： %implements interface1,interface2**

##### %extends

使得生成的词法分析器类是某个类的子类，至多定义一个%extends选项。

**例子： %extends ParentClass**

##### %public

使得生成的类是public的，类似的还有**%final**和**%abstract**指令，他们分别生成的类是final和abstract类型的。

**例子： %public**

##### %apiprivate

使得生成的类文件中，所有生成的方法和变量都变为private,只有该类的构造方法和用户自定代码段除外。如果使用了%cup选项，那么next\_token方法也不会被设定为私有。这个方法如果没有特殊情况不推荐使用。

**例子： % apiprivate**

##### %{………用户代码…….%} 类代码指令

其中用户代码将被直接复制到生成类文件中，在这里你可以定义自己的成员变量和方法。此规范描述中出现多个类代码指令，那么JFLEX将根据这些类代码指令出现的先后顺序将他们拼接起来。

**例子：**

**%{**

**public String name;**

**public void test(){**

**System.out.println(“this is a test!”);**

**}**

**………………**

**………………**

**%}**

##### %init{………初始化代码………%init}

初始化代码将被直接复制生成类的构造函数中，我们可以在这里对类指令代码中声明的成员变量进行初始化工作。同类代码指令一样，如果出现多个初始化指令定义，那么JFLEX将根据这些类代码指令出现的先后顺序将他们拼接起来。

**例子：**

**%init{**

**name=”Benson”;**

**………………**

**………………**

**%init}**

##### %initthrow

使得生成的类的构造器方法抛出某种异常，也就说当我们实例化生成的词法分析器时需要捕获异常才可以。

**例子：**

**方式1：**

**%initthrow{**

**"exception1","exception2"**

**%initthrow}**

**方式2：**

**%initthrow "exception1", "exception2"**

##### %ctorarg

使得生成的类的构造器方法，包含参数，可以设置多个该选项，那么参数会按顺序排列。

**例子：**

**%ctorarg String ss**

##### %scanerror

定义当扫描出现错误时抛出的具体异常。

**例子：**

**%scannerror XXException**

##### %buffer

设置默认扫描缓冲区大小，默认是16384

**例子：**

**%buffer 16388**

#### 扫描函数设置

##### %function

用于设置词法扫描函数的名称，如果不设置该指令，那么默认的词法扫描函数名称为：yylex;注意该指令优先于%cup指令，因为%cup指令设置后，默认扫描函数被命名为：next token，也就是说当我们使用了%cup指令后，尽量就不要使用%function指令了。

**例子：**

**%function myScanner**

##### %integer，%int

这两条指令都使扫描函数返回java语言中的int类型，在这种设置下文件结尾返回YYEOF

它是生成类 中的一个public static final int 的常量。

**例子：**

**%interger**

##### %intwrap

这条指令使扫描函数返回Java语言中的Integer类型，在这种设置下文件结尾缺省值是null.

**例子：**

**%intwrap**

##### %type

这条指令用于设置扫描函数的返回类型，在这种设置下文件结尾缺省值是null.如果指定的类型不是java.lang.Object的子类那么应该使用%eofval指令或者《EOF》来指定其他文件结束值。

**例子：**

**%type MyClassSymbol**

##### %yylexthrow

可以使扫描函数声明抛出异常，可以抛出多个异常。

**例子：**

**%yylexthrow{**

**"exception1","exception2"**

**%yylexthrow}**

**或者：**

**%yylexthrow "exception1", "exception2"**

#### 扫描结束操作

当扫描函数扫描到文件结束时都有一个默认的返回值，当然在扫描到文件结束时你也可以定义一个具体的值被返回或者一段具体代码被执行。

##### %eofval{…用户代码…%eofval}

其中用户代码部分直接被复制到扫描函数中，并且在每次文件结束时执行。这个用户代码应该返回表示文件结束的值。

**例子：**

**%eofval{ return** new **MyS**ymbol()(sy**m.EOF, null)**; **% eofval }**

##### %eof{…用户代码...%eof}

其中用户代码遇到文件结束时只执行一次，用户代码将被放到**void** **yy do eof()**方法中，并且不返回任何值。如果需要返回值应该使用%eofval{…%eofval}指令或者《EOF》规则。如果出现多个该指令，则按照出现先后顺序被连接在一起。

**例子：**

**%eof{**

**System.out.println(" "+nlines+"\t"+nwords+"\t"+nchars);**

**%eof}**

##### %eofthrow

可以使函数**void** **yy do eof()**声明抛出异常，可以抛出多个异常。

**例子：**

**%eofthrow{**

**"exception1","exception2"**

**%eofthrow}**

**或者**

**%eofthrow "exception1","exception2"**

##### %eofclose

这条指令使JFLEX在文件结束处关闭输入流，代码yyclose()被追加到方法**void** **yy do eof()**中,并且在这个方法throw子句中声明java.io.IOException。

**例子：**

**%eofclose**

##### %eofclose false

关闭%eofclose的影响。

**例子：**

**%eofclose false**

#### 添加Main主函数

##### %debug

在生成类中生成一个Main方法，它从命令行获得输入文件名，然后对这个文件运行语法分析器，并向Java控制台打印每个返回记号的信息，直到遇到文件结束。所输出的信息包括：line，column号（如果设置了%line,%column），匹配文本，执行动作。

**例子：**

%debug

##### %standalone

在生成类中生成一个Main方法，它从命令行获得输入文件名，然后对这个文件运行语法分析器，扫描器的返回值将被忽略，任何不匹配的文本将被打印在Java控制台之上。应该避免使用额外的标记类，扫描方法将被声明返回默认的int类型。

**例子：**

**%standalone**

#### CUP 能力

##### %cup

该指令等同于以下指令集

%implements java\_cup.runtime.Scanner

%function next\_token

%type java\_cup.runtime.Symbol

%eofval{

return new java\_cup.runtime.Symbol(<CUPSYM>.EOF);

%eofval}

%eofclose

##### %cupsym

使用在%cup指令之前，定义cup包含token生成类/接口的名字，默认为sym.

**例子：**

**%cupsym MySym**

##### %cupdebug

在生成类中生成一个Main方法，它从命令行获得输入文件名，然后对这个文件运行语法分析器，打印行号，列号，匹配字符以及标准的返回的CUP symbol名称。

#### BYacc/J 能力

##### %byacc

该指令等同于以下指令集

%integer

%eofval{

return 0;

%eofval}

%eofclose

#### 代码生成算法

以下这些选项，将使得JFLEX产生词汇分析代码。当没有设置代码产生选项时，%pack是默认被使用的。

##### %switch

使用%swith指令可以使JFLEX词法分析器产生一个嵌套的开关结构的代码。这个方法可以在保证良好运行的前提下，对编译.class文件数量更好的压缩。如果你的扫描器有过多的状态（大于200个），你就可以考虑使用%table或者%pack.如果状态再多（大于300个），则可能Java编译时产生错误代码，以致于执行错误代码或者在java虚拟机检测过程中产生java.lang.VerifyError异常。当出现这种情况是将被强制使用%pack。

##### %table

%table指令将产生一个经典的表格驱动扫描器，将使用数组编码DFA表格。JFLEX只进行数量较小的表格压缩。

##### %pack

％pack是默认设置，使用一个或者多个字符串生成DFA装配表。当没有使用代码生成方法的具体规定时默认使用。

#### 字符集

##### %7bit

支持字符集中0-127#字符，如果超出范围将抛出**ArrayIndexOutofBoundsException**异常。

##### %full,%8bit

支持字符集中0-255#字符，如果超出范围将抛出**ArrayIndexOutofBoundsException**异常。

##### %unicode,%16bit

支持字符集中0-65535#字符，使用该字符集不会出现运行时的溢出现象。

##### %caseless,%ignorecase

对字符的大小写忽略的设置。也就是说字母a可以对a进行匹配，也可以匹配字母A.

#### 行，列，字符设置

##### %char

字符计数器，yychar记录从输入开始到当前记号开始出处字符数（从0开始计数）。

##### %line

行计数器，yyline记录当前行数

##### %column

列计数器,yycolumn记录当前列数

======================================================================

### 声明

#### 状态

##### % states

包含状态------定义可能出现的词法状态

##### % xstates

排除状态------需要排除的词法状态。

##### 示例说明

%states A, B

%xstates C

%%

expr1 { yybegin(A); action }

<YYINITIAL, A> expr2 { action }

<A> {

expr3 { action }

<B,C> expr4 { action }

}

解释：

1. 首先确认A，B状态是包含状态，C是排除状态，默认状态YYINITIAL总是隐式的不需要被声明。
2. expr1不存在状态列表，他可以匹配任何状态，除了排除状态C。状态跳转到Ａ状态。
3. expr2只能匹配YYINITIAL和Ａ状态。
4. expr3只能匹配Ａ状态
5. expr4能够匹配Ａ，Ｂ，Ｃ三个状态
6. 总而言之，包含状态和排除状态的只有在规则前没有状态列表时才体现出来。那些状态列表为空的规则只能匹配除排除状态以外的所有规则。

#### 宏定义

宏定义规则：

宏标示符＝正则表达式

按照这种形式定义的宏标识符可以再第三部分引用，右边的正则表达式必须是合式，并且不能包含^, / 或者 $等运算符。

## 词法规则

词法规则部分包括一组正则表达式和动作行为，也就是当正则表达式匹配成功后要执行的Java代码。

### 语法

#### 了解BNF范式

语法结构使用BNF范式形式给出，所以我们先做一个简单了解。

在双引号中的字("word")代表着这些字符本身。而double\_quote用来代表双引号。  
在双引号外的字（有可能有下划线）代表着语法部分。   
尖括号( < > )内包含的为必选项。   
方括号( [ ] )内包含的为可选项。   
大括号( { } )内包含的为可重复0至无数次的项。   
竖线( | )表示在其左右两边任选一项，相当于"OR"的意思。   
::= 是“被定义为”的意思。

#### 结构定义

**LexicalRules** ::= Rule+

**Rule** ::= [StateList] [’^’] RegExp [LookAhead] Action

| [StateList] ’<<EOF>>’ Action

| StateGroup

**StateGroup** ::= StateList ’{’ Rule+ ’}’

**StateList** ::= ’<’ Identifier (’,’ Identifier)\* ’>’

**LookAhead** ::= ’$’ | ’/’ RegExp

**Action** ::= ’{’ JavaCode ’}’ | ’|’

**RegExp** ::= RegExp ’|’ RegExp

| RegExp RegExp //正则表达式连接运算

| ’(’ RegExp ’)’

| (’!’|’~’) RegExp //”~”匹配任何文本直到第一次匹配RegExp

| RegExp (’\*’|’+’|’?’)

| RegExp "{" Number ["," Number] "}" //RegExp重复次数

| ’[’ [’^’] (Character|Character’-’Character)\* ’]’

| PredefinedClass

| ’{’ Identifier ’}’

| ’"’ StringCharacter+ ’"’

| Character

**PredefinedClass** ::= ’[:jletter:]’ //java.lang.Character.isJavaIdentifierStart( )决定的字符类

| ’[:jletterdigit:]’ //由java.lang.Character.isJavaIdentifierPart( )决定的字符类

| ’[:letter:]’ //由java.lang.Character.isLetter( )决定的字符类

| ’[:digit:]’ //由java.lang.Character.isDigit( )决定的字符类

| ’[:uppercase:]’ //由java.lang.Character.isUpperCase( )决定的字符类

| ’[:lowercase:]’ //由java.lang.Character.isLowerCase( )决定的字符类

| ’.’ //包含除\n 外的所有字符

上述 EBNF 文法中使用了以下终结符：

**JavaCode**：表示Java 语言中的语句序列。

**Number**：表示一个非负的十进制整数。

**Identifier**：表示一个标识符，它是以字母（用[a-zA-Z]表示）开头，后跟0 个或多

个字母、数字或下划线（即[a-zA-Z0-9\_]）。

**转义序列包括**：

n、\r、\t、\f 和\b；

**\x** 后跟两个十六进制数字（即[a-fA-f0-9]），或者反斜杠后跟从000 到377 的三

个八进制数字，表示标准的ASCII 转义序列；

**\u** 后跟四个十六进制数字（即[a-fA-f0-9]），表示unicode 转义序列；

反斜杠后跟其他任何 unicode 字符，代表这个unicode 字符。

**Character**：是不包含下面字符之一的转义序列或任何unicode 字符：

| ( ) { } [ ] < > \ . \* + ? ^ $ / "

**StringCharacter**：是不包含下面字符之一的转义序列或任何unicode 字符：\ "

#### <<EOF>>规则

[StateList] <<EOF>> { 动作代码}

这条<<EOF>>规则与%eofval 指令十分类似，其不同在于<<EOF>>规则前可以放可

选的StateList。在遇到文件结束并且词法分析器当前处于StateList 中的某一词法状

态时，执行动作代码。

### 语义

**Character**：匹配这个字符；

**‘[’ (Character | Character ‘-’ Character)\* ‘]’**：匹配任意一个出现在方括号内的字符或

者由字符范围Character ‘-’ Character 所界定的字符。例如，[a0-2\n]可以匹配字符a、

0、1、2 或者\n。[]匹配空集，而不是空串。

**‘[^’ (Character | Character ‘-’ Character)\* ‘]’** ：匹配所有未列在方括号中的字符。[^]

匹配任意字符。

**‘"’ StringCharacter+ ‘"’**：匹配包含在双引号内的文本。

**‘{’ Identifier ‘}’**：匹配由名为Identifier 的宏的RHS 所匹配的输入。

**JFlex 在正规式中使用以下标准运算符（优先级从高到低）**：

**一元后缀运算符**（‘\*’、‘+’、‘?’、{n}、{n, m}），假设a 是正规式，则

**a\***---表示匹配 0 个或多个由a 匹配的输入；

**a+**---表示匹配 1 个或多个由a 匹配的输入；

**a?**---表示匹配 0 个或1 个由a 匹配的输入；

**a{n}**---表示匹配 n 个由a 匹配的输入；

**a{n, m}**---表示至少匹配 n 个、至多匹配m 个由a 匹配的输入。

**一元前缀运算符**（‘!’、‘~’），假设a 是正规式，则

**!a**---表示匹配除a 所匹配的串之外的输入；

**~a**---表示匹配任何文本直到第一次匹配a，它等价于正规式!([^]\*a[^]\*)。

**连接**（RegExp ::= RegExp RegExp）

**联合**（RegExp ::= RegExp ‘|’ RegExp）

**正规式 r 前面加上‘^’运算符**，表示r 只在输入流的每行开始进行匹配。

**正规式 r 后跟上‘$’运算符**，表示r 只在输入流的每行结尾进行匹配。

**假设 r1 和r2 是正规式，则r1/r2 表示r1 匹配的文本必须是定长的，或者r2 的内容**

**的开始不匹配r1 的尾部。**例如，"abc" / "a"|"b" 是合法的，因为"abc"是定长的；

"a"|"ab" / "x"\* 也是合法的，因为"x"\*的前缀不匹配"a"|"ab"的后缀；"x"|"xy" / "yx"

是非法的，因为"x"|"xy"的后缀"y"也是"yx"的前缀。

### 在动作代码中可以访问的应用编程接口（API）

生成的词法分析器类中的方法和成员变量名以“yy”为前缀，表示它们是自动生成的，避免与复制到这个类中的用户代码有名字冲突。由于用户代码也是类中的一部分，JFlex 没有像private 修饰符这样的语言手段来指示哪些方法和成员是内部的，哪些属于应用编程接口(API)。取而代之，JFlex 遵循一种命名约定：以“zz”为名字前缀的方法或域将被认为是内部使用的，在JFlex 的各发布版本之间不会通告对这些方法或域的变化；生成类中不以“zz”为名字前缀的方法或域就属于提供给用户在动作代码中使用的API，在JFlex 的各发布版本之间会尽可能地支持它们并保持稳定不变。

#### API方法和域组成

以下方法由JFLEX自动生成，并且提供给用户使用。

##### String yytext()

返回匹配的输入文本串

##### int yylength()

返回所匹配的输入文本串的长度

##### char yycharat(int pos)

返回位于匹配的文本中第pos 个字符，这等价于

yytext( ).charAt(pos)，但是执行得会更快一些。pos 的取值范围是0 到yylength( )-1。

##### void yyclose()

关闭输入流

##### void yyreset(java.io.Reader reader)

关闭当前的输入流，并复位词法分析器，使之读取一个新的输入流。所有的内部变量将被复位，原先的输入流不能被重用(内部缓冲区的内容被丢弃)。词法状态被设置为YY\_INITIAL。

##### void yypushStream(java.io.Reader reader)

将当前的输入流保存到一个栈中，并从一个新的输入流中读取。词法状态以及行、字符和列的计数信息保持不变。可以用yypopstream(通常放在<<EOF>>动作中)恢复当前的输入流。

**"#include" {FILE} { yypushStream(new FileReader(getFile(yytext()))); }**

**..**

**<<EOF>> { if (yymoreStreams()) yypopStream(); else return EOF; }**

**该方法仅当使用**--skel <file>生成词法分析器时可用。

##### void yypopStream()

关闭当前的输入流，从输入流栈出栈，并从弹出的输入流中继续读取。

**该方法仅当使用**--skel <file>生成词法分析器时可用。

##### boolean yymoreStreams()

如果输入流栈中还有输入流，则返回true。

**该方法仅当使用**--skel <file>生成词法分析器时可用。

##### int yystate()

返回当前的词法状态。

##### void yybegin(int lexicalState)

进入词法状态**lexicalState**。

##### void yypushback(int number)

将所匹配的文本中**number** 个字符退回到输入流中。这些被退回的字符将在下次调用扫描方法时被再次读入。在调用**yypushback**后，被退回的字符将不会包含在yylength 和yytext( )中。

##### int yyline

包含输入文件的当前行数（从0 开始，只有在设置了%line 指令时才被激活）。

##### int yychar

包含输入文件的当前字符数（从0 开始，只有在设置了%char 指令时才被激活）。

##### int yycolumn

包含输入文件的当前列数（从0 开始，只有在设置了%column 指令时才被激活）。

### 如何匹配输入流

当对输入流进行词法分析时，词法分析器依据最长匹配规则来选择匹配输入流的正规式，即所选择的正规式能最长地匹配当前输入流。如果同时有多个满足最长匹配的正规式，则生成的词法分析器将从中选择最先出现在词法规范描述中的那个正规式。在确定了起作用的正规式之后，将执行该正规式所关联的动作。如果没有匹配的正规式，词法分析器将终止对输入流的分析并给出错误消息。如果在词法规范描述中使用了%standalone 指令，则所生成的词法分析器会把不匹配的输入输出到java.lang.System.out，然后继续进行词法分析。