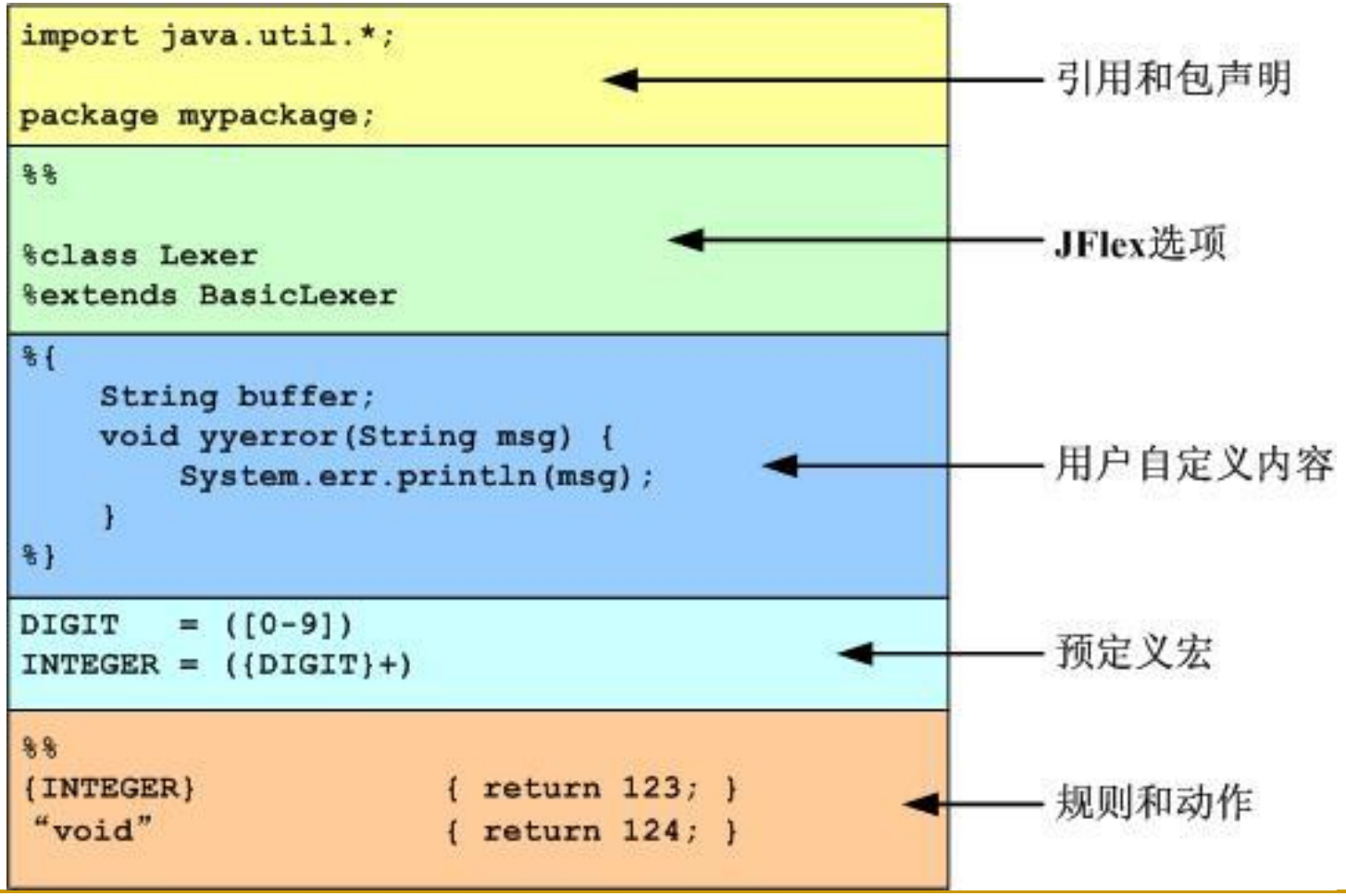
一份非常简单的JFlex介绍

- JFlex是GNU Flex的Java版本，作用是根据一系列正规表达式规则来生成词法分析器
- 输入是正规表达式和对应的匹配动作
- 输出是作为词法分析器的DFA（yylex函数）
- 每当一个正规表达式获得匹配的时候在这个正规表达式后面的匹配动作则被执行
- JFlex与GNU Flex的输入脚本格式有点不同

JFLEX INTRODUCTION

- JFlex脚本格式：



JFLEX INTRODUCTION

- “引用和包声明”部分书写各种import和package语句，这部分的内容会直接复制到生成文件的开头
- “JFlex选项”的含义请参考JFlex手册4.2节
- “用户自定义内容”部分定义词法分析器的其他成员变量和成员方法，这部分内容将直接复制到生成的文件
- “预定义宏”部分等价于GNU Flex的相同部分，格式是“宏名字 = (正规表达式)”，引用的时候用“{宏名字}”这样的形式使用

JFLEX INTRODUCTION

- “引用和包声明”、“选项、自定义内容、宏声明”、“匹配规则和动作”这三部分之间均使用两个百分号（%%）进行分隔
- 规则和动作的格式是：

  <正规表达式>　　<匹配时执行的动作>
- 当左边的正规表达式获得匹配的时候，右边的动作会被执行
- 动作是一系列Java代码，会被直接复制到词法分析器对应位置
- 建议看一下Decaf框架中通过Lexer.l生成的Lexer.java

JFLEX INTRODUCTION

- JFlex正规表达式简介：
- 由一般字符和正规表达式运算符组成
- 一般字符是ASCII字符，以下几个字符是特殊字符：” ”，”\t”，”\n”，”\r”
- 运算符：
  - . [ ] ^ – * + ? { } “ \ ( ) | / $ < >
  - 用转义符”\”引用运算符，例如”\\”

JFLEX INTRODUCTION

- 常用的正规表达式模式

x	匹配字符x
.	匹配除换行外的所有字符
[xyz]	字符集合，匹配字符‘x’、‘y’或 ‘z’
[^A-Z]	字符集合，匹配除大写字母外的所有字符
r*	正规式r出现零次或多次
r+	正规式r出现一次或多次
r?	正规式r出现零次或一次
r{2,5}	正规式r重复2至5次
r{2,}	正规式r重复2次以上（含2次）
r{4}	正规式r重复4次
{name}	辅助定义“name”的展开
“[”	字符[
\”	字符”

JFLEX INTRODUCTION

\0	空字符(ASCII 码为 0)
\123	ASCII 码为八进制数123的字符
\x2a	ASCII码为十六进制数 2a的字符
(r)	匹配正规式r
rs	正规式r后面紧跟正规式s
r|s	匹配正规式r或s
r/s	当正规式r后面紧跟s时，匹配r
^r	当正规式r位于行首时，匹配r
r$	当正规式r位于行尾时，匹配r，相当于"r/\n"
<s>r	在开始条件s下匹配正规式r
<s1,s2,s3>r	在开始条件s1,s2或s3下匹配正规式r
<*>r	在任何开始条件下都匹配正规式r
<<EOF>>	遇到文件结束符时
<s><<EOF>>	在开始条件s下遇到文件结束符时

JFLEX INTRODUCTION

[a-zA-Z0-9]

表示所有的字母和数字组成的集合；

[^ \t\n]

表示除空格、tab和换行外的所有字符组成的集合；

(\"[^"\n]*")

表示单行字符串，例如：“here is a string”；

a{1,5}

表示a重复1至5次形成的串的集合，即{a, aa, aaa, aaaa, aaaaa}

[A-Za-z][A-Za-z0-9]*

表示以字母开头，后跟若干字母和数字串

([Ee][-+][0-9]+)

表示科学计数法的指数部分

JFLEX INTRODUCTION

- 在集合中，字符之间不要留空格，否则空格“ ”将被包含在集合中
- 适当使用宏可以使正规表达式的书写清晰简洁
- 建议简单浏览一下JFlex Manual如下两章：
  - A Simple Example: How to work with JFlex
  - Lexical Specification
- 当你书写JFlex脚本遇到问题的时候请多翻阅JFlex Manual

一份非常简单的BYACC/J介绍

- BYACC/J是Berkey YACC的Java版本，作用是根据一套LALR(1)文法来生成语法分析器
- 输入是文法描述和语义动作
- 输出是作为语法分析器的DPDA（yyparse）
- 对于每条语法规则，后面的语义动作在用这条规则进行归约的时候执行，中间的语义动作会转换成一条空产生式，并在用这条空产生式进行归约的时候执行
- BYACC完全兼容AT&T YACC，比GNU Bison要小而且所生成的语法分析器更小更快

BYACC/J INTRODUCTION

BYACC输入脚本的格式：

%{

声明　　－－可选

%}

辅助定义　　－－可选

%%

语法规则　　－－必须有

%%

用户子程序　　－－可选

BYACC/J INTRODUCTION

- 所有“声明”部分的内容将会直接复制到语法分析器源文件的开头，一般是书写import和package语句
- 辅助定义区中：
- %start XXX指定非终结符XXX为文法开始符号
- %semantic XXX指定XXX为语义值类型
- %token X Y Z指定X、Y和Z为终结符
- %type <T> X声明非终结符X的语义值类型是T

BYACC/J INTRODUCTION

- 辅助定义区中：

%left X Y

%right Z W

%nonassoc U V

- 上面三行的含义是X和Y都是左结合，优先级相同；Z和W都是右结合，优先级相同；U和V均无结合性，且优先级相同；
- 上面各符号的优先级为U = V > Z = W > X = Y
- 即结合性声明写在越下面的符号优先级越高

BYACC/J INTRODUCTION

- 语法规则的格式是：
  产生式左部：产生式右部；
- 右部为空表示左部的非终结符可以匹配空串
- 左部相同的语法规则应尽量合并
- 需要递归时，请尽量使用左递归，从而尽早进行归约，以免分析栈溢出

BYACC/J INTRODUCTION

- 语义动作可以写在产生式右部的任何地方
- 语义动作用{ }括起来，其中的语句要符合Java语法
- 语义动作出现在规则的尾部时，BYACC在归约前执行它。
- 语义动作出现在规则的中间时，BYACC在识别出它前面的若干文法符号后执行它

BYACC/J INTRODUCTION

- BYACC中每个文法符号都有自己对应的语义值，具体原理请首先参考课本关于“属性文法”的相关内容
- 终结符的语义值由词法分析程序给出，并保存在yylval中
- 非终结符的语义值在语义动作中获得
- 在语义动作中可以通过$伪变量访问语义值
  - 左部非终结符的语义值为$$
  - 右部文法符号的语义值依次为$1,$2,…

BYACC/J INTRODUCTION

- 语义动作嵌在右部第n-1个和第n个文法符号之间时
  - 动作中只能引用它前面的n-1个文法符号的语义值
  - 动作的语义值为$n，动作中可以用$$为它赋值，后续动作中可以通过$n引用它的值。
  - 动作后面的文法符号的语义值依次变为$n+1, $n+2, …

BYACC/J INTRODUCTION

- 关于语义值需要注意的地方：
  - 对于空产生式，如果$$是有意义的，则必须对它赋值（在Decaf中使用$$ = new Symbol();来完成）
  - 对于这样的规则 A : X Y {…1..} Z {…2..} ;

    实际上BYACC会自动转换成这样的规则：

    A : X Y W Z {…2..} ;

    W: /* empty */ {…1..} ;

    因此在{…1..}中使用$$的话实际上是设置W的语义值而不是A的语义值，同时也说明了为什么在{…2..}中Z的语义值是$4而不是$3
  - 对于非空产生式，$$跟$1指向同一个地方，因此修改了$$以后，$1的内容已经没有意义了

BYACC/J INTRODUCTION

- 关于“归约/归约”冲突
  - 如果冲突双方都有优先级，那么选择优先级高的那条规则进行归约
  - 否则YACC默认使用写在前面的规则归约
  - 一条规则的优先级是这样决定的：
    - 如果在规则后面（归约动作之前）用%prec X来修饰，则这条规则的优先级与符号X相同
    - 否则这条规则的优先级与规则右部最右边那个终结符的优先级相同

BYACC/J INTRODUCTION

- 关于“移进/归约”冲突
  - 移进方的优先级和结合性是前瞻符号（LALR(1)的那个“1”）的优先级和结合性（可能没有定义）
  - 归约方的优先级同前所述
  - 如果冲突双方都有优先级或者结合律（“无结合性”也是一种结合律），那么优先级高的一方优先；优先级相同的时候移进方是左结合则归约，是右结合则移进，是无结合性则出错（BYACC在无结合性出错这点上有个BUG）
  - 否则YACC默认进行移进

BYACC/J INTRODUCTION

- 遇到语法错误时，YACC调用报错程序yyerror()输出错误信息
- YACC缺省的错误处理是遇到第一个语法错误就退出语法分析程序
- 利用保留的终结符error，可以在发现语法错误后继续进行语法分析

例如

```
statement   :   …
            :   error   ';'
            ;
```

BYACC/J INTRODUCTION

- 建议简单浏览一下YACC手册（即论文YACC—Yet Another Compiler Compiler）
- 由于YACC手册相对没有JFlex手册那么简单易懂，因此我们也提供了YACC的中文译文版本（来自Internet，不保证完全正确，若有不理解的细节请参考英文版本）
- 也请参考BYACC/J的主页的相关内容