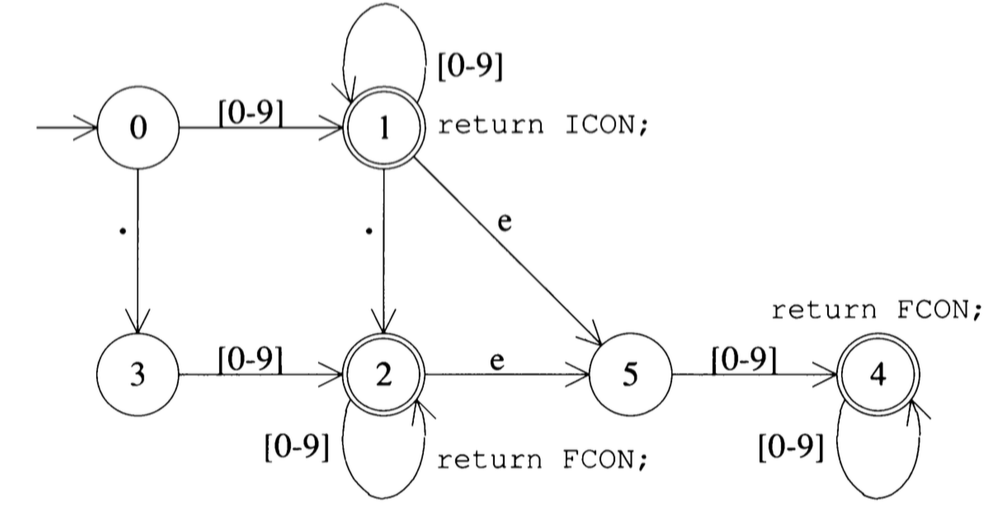
Thompson构造:正则表达式的词法解析

大家好，欢迎大家来到coding迪斯尼，继上一节我们开发了闭包替换功能后，这一节，我们继续推进Thompson 构造算法的开发。我们的目标是，给定一组正则表达式后，把他转换为NFA有限状态自动机。无论是正则表达式，还是最终的有限状态自动机，他们的本质都是对输入文本进行判定。例如正则表达式：

{D}+

({D}+|{D}\*\.{D}+|{D}+\.{D}\*)(e{D}+)

用来判别输入是否是整形或浮点型字符串。其对应的有限状态自动机：

也是用来判定整形和浮点型字符串。他们形式不同，但表达的内容相同。大家再联想到开始我们做的一个简易编译器。将算术表达式转换为计算机汇编伪代码。同理，算术表达式与伪代码的表现形式不同，但本质都是表述如何对数字进行运算。这种将信息的一种表现形式，转换为另一种表现形式，转换后虽然形式不同，但信息内容相同的行为，不就是编译吗。由此可见，将正则表达式转换为有限状态自动机的过程，其实就是编译过程。我们要实现的这个程序，其实就是一个编译器。

前面我们提到，编译器的架构时，大家可以了解到编译器的运行过程。大体上就是通过词法解析器，将要编译的内容读入，词法解析将读入的内容分解成有特定意义的子部分，也就是打标签。语法解析器通过词法解析器解读输入内容。当遇到特定标签的时候，采取特定的解读操作。由于我们目前正在开发的也是一个编译器，因此，同样需要走类似的流程。而这一节我们的重点，就是开发正则表达式的词法解析器。

正则表达式的特点：

我们在前面的章节中，曾对正则表达式做过解析，在这里，我们复习一下。正则表达式其实是由一组由普通字符和特殊字符组合而成的一种表达形式。特殊的字符有特殊的含义，在正则表达式中，特殊字符有：

+ ? { } [ ] ( ) . ^ $ “ \

在解读正则表达式时，遇到普通字符，我们知道，普通字符匹配它对应的ascii字符，例如字符 a 匹配 它对应的ascii字符 ‘a’. 但普通字符和特殊字符结合在一起时，我们需要进行不同的解读，例如 [a-z] 匹配所有小写字母。普通字母跟转义符结合时，也需要进行不同的解读，例如：

\b 表示backspace， 相当于键盘的delete

\n 表示换行

\r 表示回车

\s 表示空格

\e 表示键盘的 esc 键

\ddd d表示数字，\ddd 表示三位八进制数

\xddd 表示三位十六进制数

\^C 一个反斜杠，一个上尖括号，C代码任意一个字母，他表示键盘键Ctrl 加对应字母的组合

\c 一个转义符加任何一个字符，表示匹配这个字符本身。例如 \. 就匹配字符 .

如果没有斜杠，. 在正则表达式中是一个通配符。 \\* 表示匹配字符 ‘\*’, 如果没有斜杠， \* 在正则表达式中表示闭包操作。也就是特殊字符前面是转义符的话，它就不再具有特殊含义。

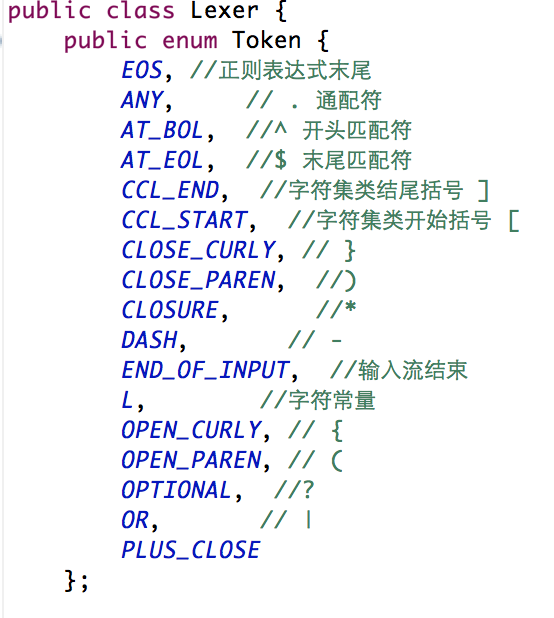
有一些特殊符号，例如^ 表示开头匹配， ^[a-z] 表示匹配任何以小写字符开头的字符串，[a-z]$ 匹配任何以小写字母结尾的字符串。{ 表示宏定义的开始， }表示宏定义的结束。

在双引号” ” 中的任何特殊字符都不再具备特殊含义，例如 正则表达式 “+\*?” 就只匹配三个字符 + \* ?.

词法解析器的实现：(挑出eclipse)

基于以上结论，我们可以开始设计词法解析器。对正则表示是，词法解析比较简单，我们只要一次读入一个字符，并返回该字符对应的标签就可以了。在代码中，

我们给每个特殊字符都赋予特殊的标签，而对于普通字符，我们统一给一个标签叫 L , 表示 Literal 即字符常量，在代码中，用enum 类定义了特殊字符的标签值：



由于我们要处理的字符，基本上只有ascii 码字符，因此字符总共才128个。我们用一个长度为128字节的数组来存放每一个字符所对应的标签，在代码里这个数组叫tokenMap, 词法解析器读入一个字符，获取该字符对应的ascii 码值，根据ascii 码值在tokenMap中去获取对应标签值， 例如， 如果输入的是符号 .

, “.” 对应的ascii 码值是62， 于是tokenMap[62]得到的值就是enum Token 中的ANY.

我们接下来看tokenMap的初始化代码：



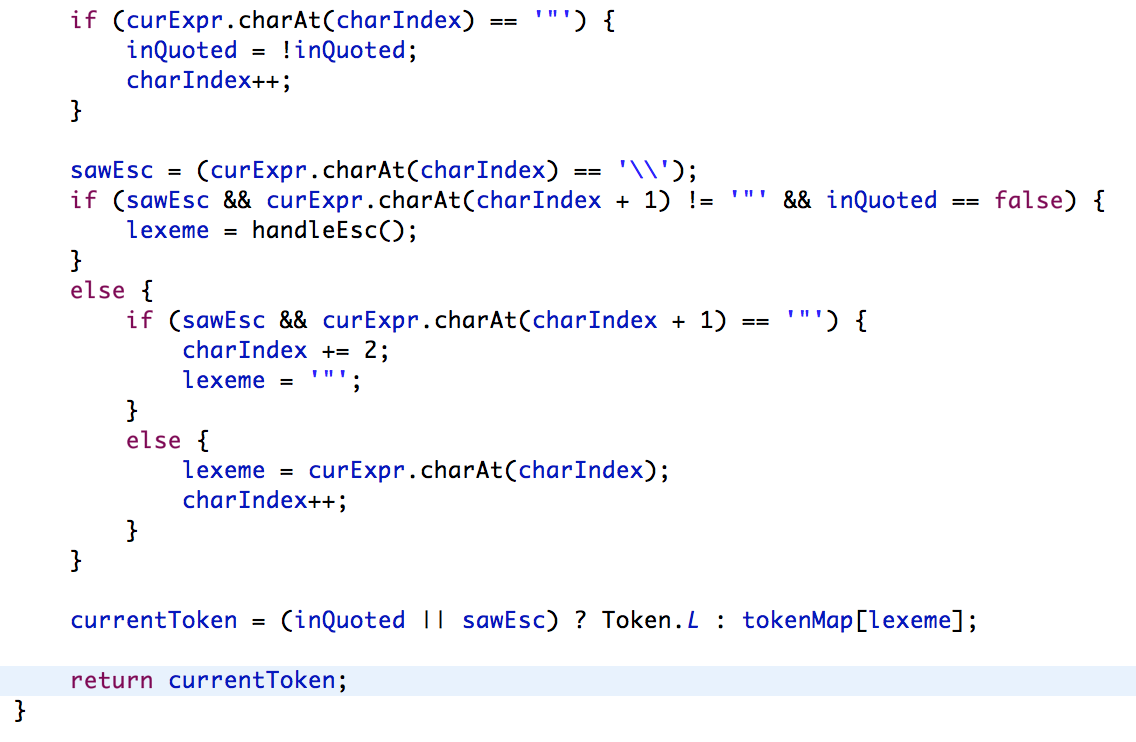
解析器在构造时，先给tokenMap的每一个元素赋初值Token.L, 然后再对特殊字符设置对应的标签。

Lexer 类的advance 接口用来读入字符，然后返回字符对应的标签：



在advance函数中，我们先从exprHandler中获取已经处理好的正则表达式字符串。然后从获取的字符串中依次取出字符，进行逐个解析，charIndex用来表示当前解析字符所在的正则表达式字符串中的下标。EOS表示当前正则表达式的字符串已经分析完毕，当下次在进入advance 时， 一旦发现当前标签是EOS, 则从exprHandler 中读入新的正则表达式字符串。

再往下看：



当我们读到的字符是双引号时，我们需要做标记，因为在双引号中的所有字符，不管是普通字符，还是特殊字符，我们都把他们当做普通字符处理。如果读到转义符，那就要进行相应处理。对转义符进行处理的函数是handleEsc, 在advance函数的最后，我们看看当前解析的字符是否处于双引号中或字符被转义了，前面提到过转义符后面跟着的任何字符或处于双引号中的字符都会被当做普通字符处理，如果不是这种情况，则在tokenMap中查找字符对应的标签。

我们再看看对转义符的专门处理函数handleEsc, 如果正则表达式[\b]被解析时，字符 \ 和 字符 b 会连在一起解读，解读他们的函数就是handleEsc:

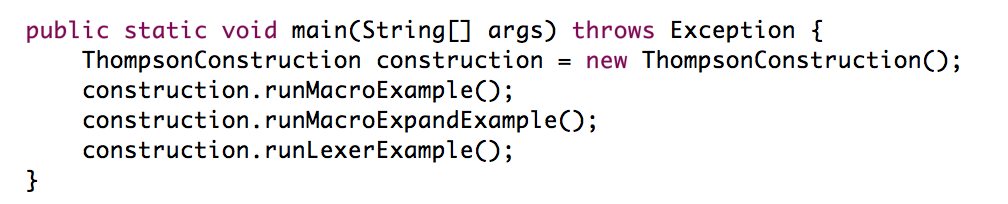


大家可以看到 \ 和 b 会被连在一起解读成 ‘\b’, ‘\b’ 在ASCII表中表示为回删符，也就是键盘上的Delete键。继续往下看：



如果表达式中遇到字符串 \^A, 前面我们提到过，这表示在键盘上的Ctrl+A, 在代码注释中，我给出了如何对类似的字符进行转换。接着的代码处理的是将\xDDD解读成三位十六进制数。在后面的调试演示中，我们会详细的理解这段代码的运作原理。

在主函数main 中，执行runLexerExample, 就可以调试词法解析器的功能了：



在runLexerExample中，词法解析器从控制台中读入正则表达式，然后逐个解析表达式的字符，并把字符对应的标签含义显示到控制台中，例如输入的正则表达式宏替换后是[0-9]+, 那么runLexerExample会输出结果如下：



在下一节，我们将进行代码的调试演示。让大家对词法解析器的实现原理有进一步的理解。

演示例子：

[0-9]+

[\b\r\n]

[\x1F]

[“+\*?”]