代码实现Thompson构造: 实现正则表达式OR连接后的nfa

大家好，欢迎大家来到coding迪斯尼。在前两节课程，我们由简单到复杂的构建了相应正则表达式的Nfa有限状态机，回忆一下，我们先构造了最简单的正则表达式nfa:

term -> character | . |[…] | [.-.] | [^…] | [^.-.]

term构造的是最简单的正则表达式的nfa. 在term的基础上，我们添加了构造闭包操作的nfa实现:

factor -> term\* | term+ | term?

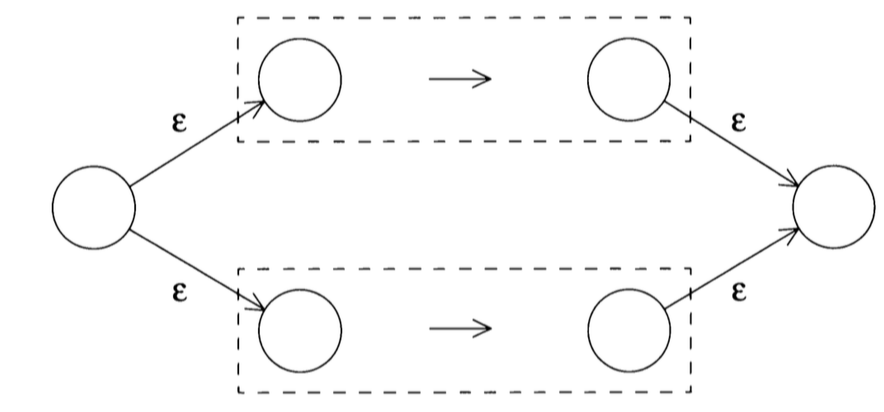
在factor基础上，我们又实现了正则表达式的连接操作:

cat\_expr -> factor factor …..

连接操作是由一个或多个factor前后连接构成的，所以上面的语法又可以写成：

cat\_expr -> factor cat\_expr

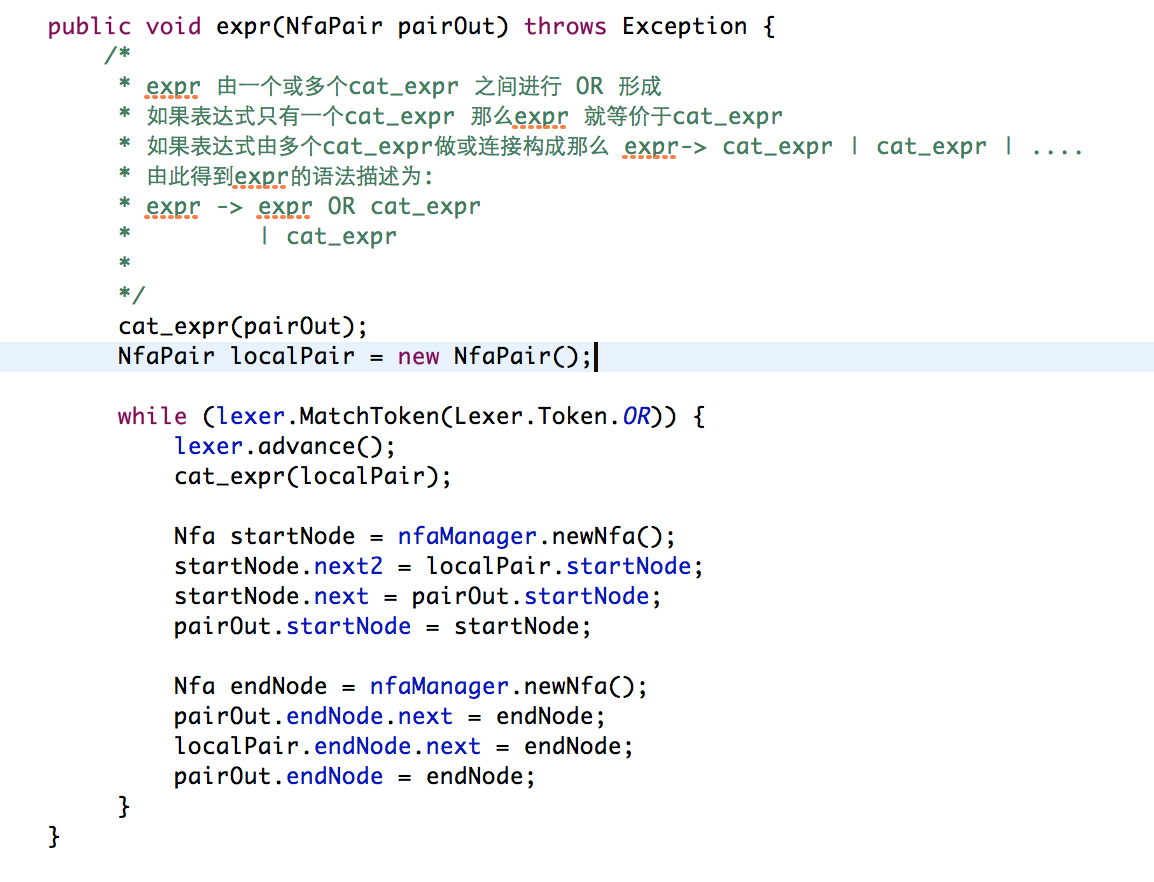
本节，在cat\_expr， 也就是正则表达式的连接操作基础上，我们构造正则表达式的OR操作。或运算，实际上是多个cat\_expr 进行 | 操作实现的。根据以前讨论过的，关于 OR 操作的Thopmson构造算法如下图：



图中，虚线框部分就是cat\_expr, 因此我们可以总结出，正则表达式进行OR运算后，它对应的Nfa构造流程如下：

1. 先构造 cat\_expr
2. 检验当前字符是否是 ‘|’
3. 如果第二部成立，构造第二个 cat\_expr
4. 分配两个nfa节点，分别为startNode 和 endNode，将分配的startNode 的next 指向第一个cat\_expr的起始节点，将startNode的next2指向第二个cat\_expr的起始节点
5. 将第一个cat\_expr 和 第二个 cat\_expr 的末尾节点的next指针都指向上面分配的endNode节点
6. 重复步骤 4

根据前面提到的步骤，我们代码实现如下：



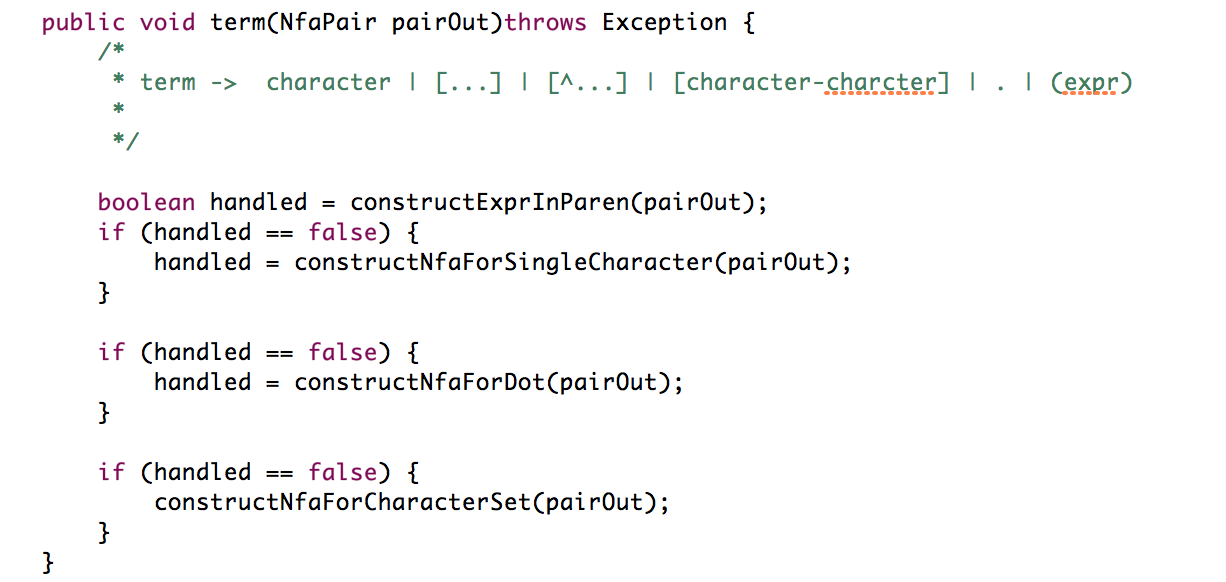
在expr 调用中，首先调用cat\_expr ，对应的是前面步骤的第一步。While判断中，lexer.MatchToken(Lexer.Token.OR) 指向的是第二部。在while循环中，cat\_epxr(localPair) 执行的是第三步，后面指针赋值对应的是构造步骤中的第4，5步，完成后重新进入while循环，对应的就是步骤6.

在后续的代码调试演示中，我会给大家展示程序运行的结果。到这里，正则表达式转换为Nfa有限状态自动机的实现，基本就结束了。但代码还有一点问题，就是没有对()圆括号进行处理，不难理解，圆括号里面的内容，其实对应的就是我们现在实现的expr, 所以，一旦我们遇到( 之后，我们就调用expr 对 (后面的内容进行解析。

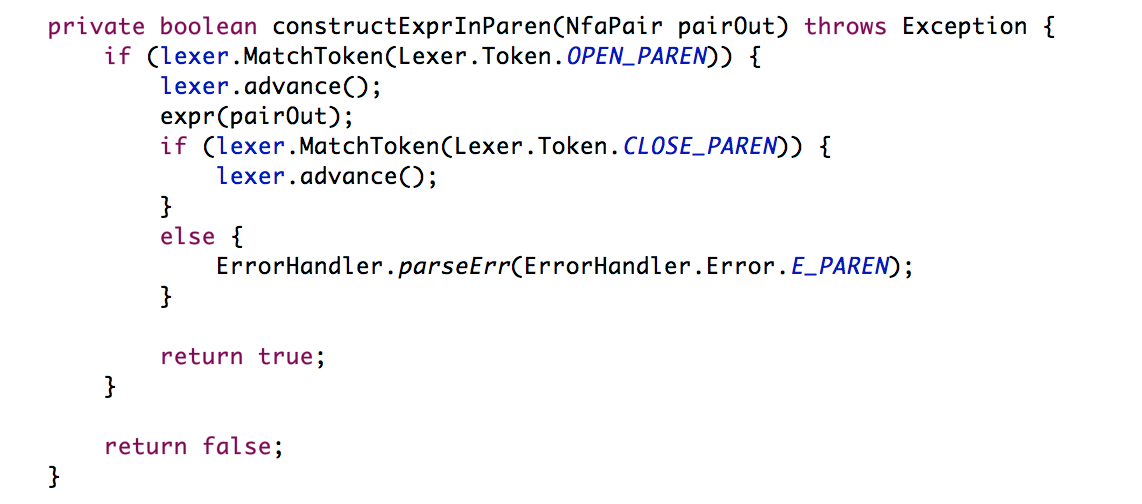
从代码上看，解析正则表达式时，代码的调用流程如下:

expr -> cat\_expr -> factor -> term

因此，我们可以在term中处理遇到圆括号的情况，于是我们对代码做如下添加：



在term 中，我们先调用constructExprInParen来处理圆括号的情况，再看看constructExprInParen的实现：



一旦遇到 符号 ( 我们立马调用expr函数来构造圆括号中的内容，然后看看正则表达式是否以)结尾，如果不是，那抛出异常。

至此，我们用Thopmson构造法将正则表达式转换为Nfa有限状态自动机的实现流程就结束了。

语法解析规则:

通过这几节的代码实现，我们的思路是自底向上，由简单到复杂，有具体到抽象的过程。term 函数处理的对象是最具体的，它解析给定的正则表达式，并转换成Nfa状态机， factor 就比term抽象一些，它在term实现的基础上，实现了闭包操作， cat\_expr就更抽象，它在factor的基础上，实现了连接操作的Nfa转换，最抽象的是expr, 它在cat\_expr的基础上，实现了表达式的或操作。这是我们从底部开始往上看所得到的形态，如果我们从上往下看，我们便会得到正则表达式到Nfa转换的语法解析规则:

expr -> expr “|” cat\_expr

| cat\_expr

cat\_expr -> factor cat\_expr

| factor

factor -> term\* | term+ | term?

term -> [string] | . | character | (expr) | [^string]

string 表示ASCII字符组成的字符串

character 表示单个ASCII 字符。

大家如果读过其他编译原理书籍时，或许会发现，他们采取的方法是第二种，从上到下的方式来介绍语法解析流程，但是从上到下这种方式，对于初学者来说，会很抽象，难以理解和入手，因此我采用的是从下到上，先让大家看看具体细节是怎样的，理解细节后，再尝试总结出抽象规则，由感性到理性，这样或许大家更容易接受。

后续的课程，我们将讨论如何将NFA 转换为 DFA.