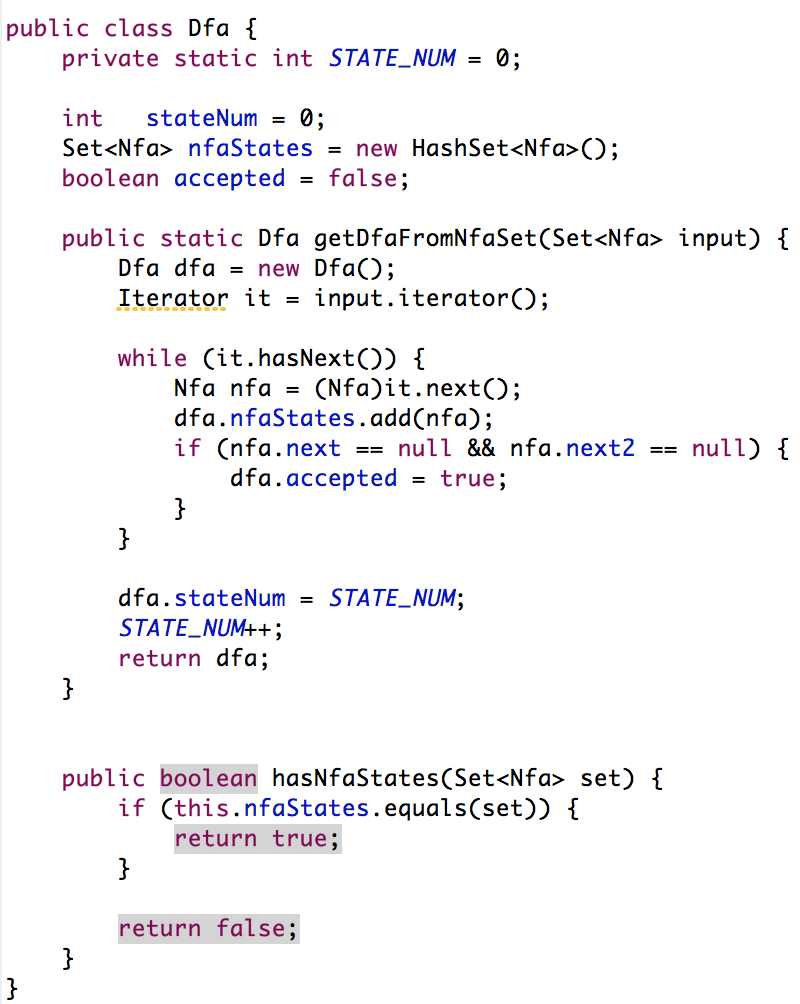
将NFA转换为DFA：代码实现

大家好，欢迎大家来到Coding迪斯尼，上一节，我们讨论了NFA转DFA的算法步骤，这一节，我们看看，如何在代码上实现我们上节讨论过的算法原理。

DFA数据结构描述：



STATE\_NUM是一个静态变量，它用来标记DFA状态节点编号，每生成一个DFA节点，该编号就自动自加。stateNum 是DFA的几点编号，节点S0那该节点对应的编号，也就是stateNum的值就是0，S1节点的stateNum就是1，以此类推。nfaStates用来记录DFA节点对应的Nfa闭包集合，以上节我们讲过的节点S0为例，(S0, { 17, 3 , 1, 4, 5, 9})，那么nfaStates就存储了编号为17,3,1,4,5,9的NFA节点对象。

getDfaFromNfaSet 是Dfa类的构造函数，我们不需要用new来生成Dfa对象，只需要把Nfa闭包集合传入该函数，那么它会返回一个Dfa节点对象。在这个接口中，它会判断传入的Nfa集合中，有没有接受节点，也就是while中的if判断语句，如果有，那么该Dfa节点就是接收状态节点。

hasNfaStates用来判断传入的Nfa集合是否与节点对应的Nfa集合相同，大家还记得上一节，每次构造转移集合的ε闭包时，都要到队列里查看有没有该闭包对应的Dfa节点存在吧，这个函数就用来做闭包将的判断。

转换算法的代码实现：

在代码中，文件DfaConstructor.java用来实现算法代码，我们分析下代码的逻辑。

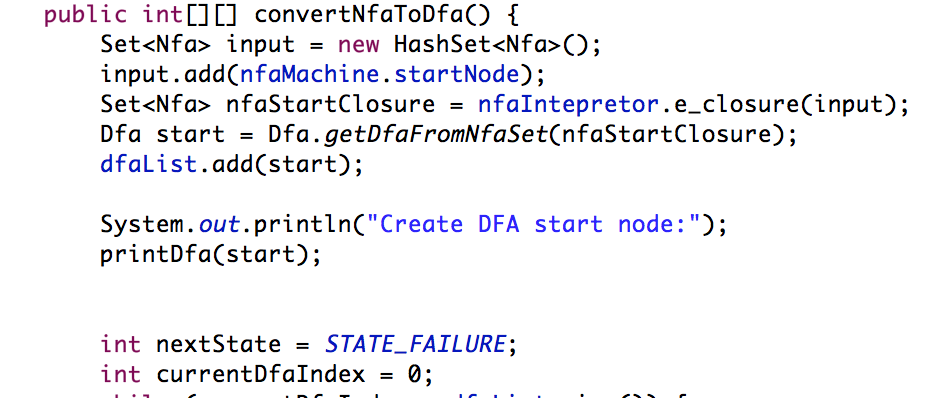


由于转换算法中，闭包运算和转移运算是关键，这两中运算的实现都在类NfaIntepretor中，因此我们需要引用它的实例。dfaList对应上节讲过的队列，它用来存储每一个生成的Dfa节点，当要生成新节点时，先从该队列中取出原有节点，看看要生成的节点是否已经存在。

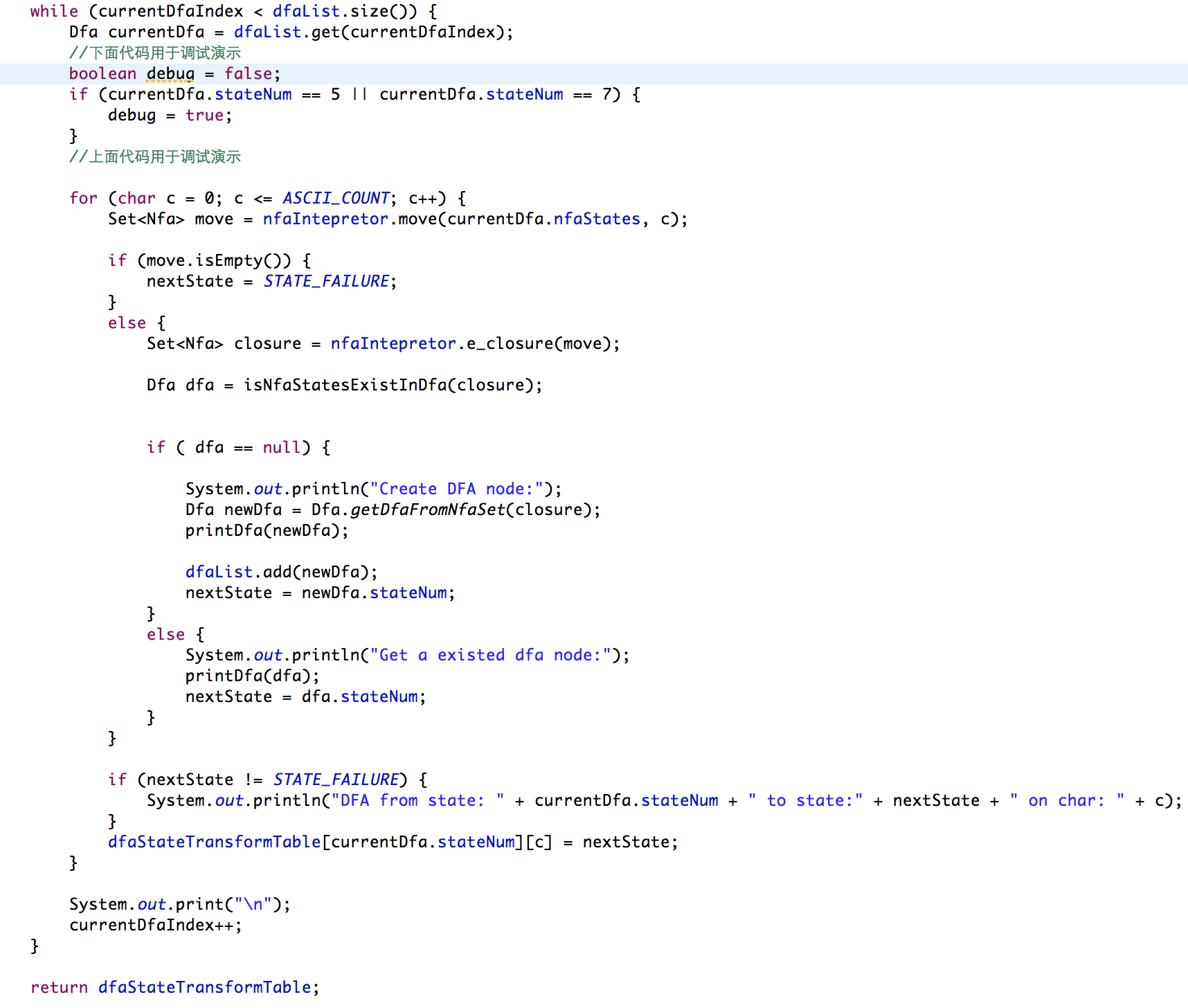
dfaStateTransformTable 用来表示Dfa状态机，大家还记得前面有一节讨论：《有限状态机驱动的整形，浮点型数值识别器》，在那里，我们使用二维数组来表示DFA有限状态自动机吧，同理，这个二维数组也是用来表示我们转换完成后的有限状态自动机的，它的行对应Dfa的节点，对应ASCII字符输入，例如字符’a’的ascii值是98，那么dfaStateTransformTable[0][98]表示节点0接收字符’a’后将要跳转的写一个状态的标号。

nfaMachine用来表示我们要转换的Nfa状态机。

接下来我们看看转换算法的实现主体，即函数convertNfaToDfa:



这一部分代码的目的对应于上节算法描述的第一步，即使用Nfa的初始状态节点，构造该节点的ε闭包，然后在用该闭包构造DFA起始节点S0.接着往下看，



在while循环中，我们要做的是从队列中取出已经构造好的Dfa节点，计算节点的Nfa闭包集合对应的转移集合。由于我们状态机可以接收ascii字符作为输入，因此，里面有个for循环来对应每一个可能的ascii字符。但是我们的状态机对数字字符和字符’.’才会有跳转，因此在for循环中，如果c变量的值不是数字字符或字符’.’， 那么nfaIntepretor.move都会返回空。如果c是数字字符或点字符，得到的转移集合不是空，那么就对转移集合做闭包运算，也就是代码走入了else部分。isNfaStateExistInDfa 它的作用就是上节算法所说的遍历队列，看看队列中是否已经有Dfa节点的nfa集合与当前生成的闭包集合一样，如果有那么该函数返回一个Dfa节点对象，如果没有，那么返回null, 返回null意味着，当前的闭包集合可以生成一个新的Dfa节点，也就是进入if(dfa == null) 的代码块。在代码块中，我们生成一个新的Dfa节点，然后把节点加入队列。最后我们在dfaStateTransformTable中设置点与点间的转移关系。

代码逻辑基本描述就结束了，下面我们通过调试运行的方式去进一步理解转换算法的设计实现。