字符串转换整数 (atoi)

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：atoi 函数、DFA、正则、力扣题：8

题目描述

实现一个 atoi 函数，使其能将字符串转换成整数。

该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。

当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。

该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0。

说明：

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 $[−2^{31}, 2^{31} − 1]$ 。如果数值超过这个范围，请返回 INT_MAX ($2^{31} − 1$) 或 INT_MIN ($−2^{31}$) 。

示例 1

```
输入: "42"
输出: 42
```

示例 2

```
输入: "   -42"
输出: -42
解释: 第一个非空白字符为 '-', 它是一个负号。
      我们尽可能将负号与后面所有连续出现的数字组合起来，最后得到 -42 。
```

示例 3

```
输入: "4193 with words"
输出: 4193
解释: 转换截止于数字 '3' ，因为它的下一个字符不为数字。
```

示例 4

```
输入: "words and 987"
输出: 0
解释: 第一个非空字符是 'w', 但它不是数字或正、负号。
     因此无法执行有效的转换。
```

示例 5

```
输入: "-91283472332"
输出: -2147483648
解释: 数字 "-91283472332" 超过 32 位有符号整数范围。
     因此返回 INT_MIN (−231) 。
```

1、正常解法

1.1 编码

```java
public int myAtoi(String str) {
    str = str.trim();  // 删除字符串头尾空格
    if (str.length() == 0) return 0;
    int flag = 1;  // 符号位标识
    int rev = 0;  // 数值（无符号）
    int edge = Integer.MAX_VALUE / 10;  // 判断数值是否超过范围的边界线，这样写可以节省时间
    if (str.charAt(0) == '-') {
        flag = -1;
        str = str.substring(1, str.length());  // 跳过符号位，可不写第二参数
    } else if (str.charAt(0) == '+') {
        str = str.substring(1, str.length());  // 跳过符号位，可不写第二参数
    } else if (!(str.charAt(0) >= '0' && str.charAt(0) <= '9')) {  // 如果开始非空字符不为符号或数字，则直接返回 0
        return 0;
    }
    for (char s : str.toCharArray()) {
        if (s >= '0' && s <= '9') {
            int n = s - '0';  // 计算字符代表值
            if (rev >= edge) {  // 超过边界情况较少，故该判断写于外侧
                if (flag == 1) {
                    if (rev > edge || n > 7) return Integer.MAX_VALUE;
                } else {
                    if (rev > edge || n > 8) return Integer.MIN_VALUE;
                }
            }
            rev = rev * 10 + n;
        } else {
```

```
                break;
            }
        }
        return rev * flag;
    }
```

1.2 复杂度分析

- 时间复杂度：$ O(n) $ 。
- 空间复杂度：$ O(n) $ 。

2、通过有限状态机(DFA)解析此题

2.1 确定有限状态机(DFA)是什么?

对于该题，就是逐字符读取题目给出的字符串，然后在当前状态下，读到该字符，应该转变为什么样子的状态，然后再通过该状态去进行相应的操作；
对于“有限”两字的理解，我的理解是，当前状态只会不断的朝着结束状态靠近，不会返回到上一级的状态，就是只会朝着可执行状态数量越来越少的方向靠近。

2.2 确定有限状态机(DFA)的作用

通过DFA，可以减少代码有过多的 if - else 条件判断语句（貌似这道题通过DFA并没有减少很多的 if - else语句，相反代码行数还挺多，应该是这道题的判断条件并不多，如果判断条件更加的多，更加的繁杂，通过DFA还是有相应的优化的）；
使用DFA还可以增加代码的维护性，使代码更加容易修改和维护

2.3 题目解析

对于该题，有四种状态，分别是：
start ：读到空格时的状态
singed： 读到 - / + 号时的状态
in_number：读到数字时的状态
end：读到其他的字符时的状态

初始状态为start，作为该DFA的一个入口，可以列出下面这个状态表

	‘ ’（空格）	**+ / -**	**number(数字)**	**other（其他）**
start	start	singed	number	other
singed	end	end	in_number	end
in_number	end	end	in_number	end
end	end	end	end	end

解释：（只举了其中一种可能的例子）

刚开始是start状态，如果刚开始读到的字符是空格，就对应的start的状态，不停的循环，直到读到非空格为止；

如果start的状态下读到了 + / - 号，那么就进入下一个状态singed状态，此状态下将读到的符号通过一个变量给存储下来。

根据题目的要求，符号的后面只能接数字，如果是非数字，那么这个字符串就无法转化为整数，直接返回0；所以，singed状态下如果读到了 空格、 符号、 其他字符，就直接跳转到结束状态，如果读到的是数字，那么就跳转到in_number状态，而in_number状态下和singed状态是一样的，只要读到非数字，就直接转到结束状态。

2.4 代码实现

```
class Solution {

    public int myAtoi(String str) {
        Automaton auto = new Automaton();
        char[] ch = str.toCharArray();

        for(char c : ch){
            //如果auto.calculate(c)是false,则终止循环
            if(!auto.calculate(c)) break;
        }
        return auto.sum;
    }
}
class Automaton{
    //设置确定有限状态机的状态
    final String START = "start";//开始状态
    final String SIGNED = "signed";//符号位状态
    final String IN_NUMBER = "in_number";//数字位状态
    final String END = "end";//结束状态

    String state = START;//设置初始状态，start
    int sum = 0;//记录字符转换为数字的结果
    int sign = 1;//记录符号

    //通过HashMap构建一个DFA表
    Map<String, String[]> table;
    public Automaton(){
        table = new HashMap<>();
        table.put(START, new String[]{START, SIGNED, IN_NUMBER, END});
        table.put(SIGNED, new String[]{END, END, IN_NUMBER, END});
        table.put(IN_NUMBER, new String[]{END, END, IN_NUMBER, END});
        table.put(END, new String[]{END, END, END, END});
    }

    //用来获取该字符所对应的DFA表里的列
    public int getState(char c){
        if(c == ' ') return 0;
        if(c == '+' || c == '-') return 1;
        if(c >= '0' && c <= '9') return 2;
        return 3;
    }

    public boolean calculate(char c){
        //table.get(state)获取到DFA表里的行
```

```
            //getState(c)获取该字符在DFA表里的列
            //两者组合，获取到了一个确定的状态
            state = table.get(state)[getState(c)];

            if(END.equals(state)){//结束状态，返回false，传达终止信号
                return false;
            }else if(SIGNED.equals(state)){//符号位状态，记录当前符号
                sign = c == '-' ? -1 : 1;
            }else if(IN_NUMBER.equals(state)){//数字位状态
                int num = c - '0';//将char 转换为 int
                //此处题目要求，只能用int来存储数据类型，所以通过该方法判断sum是否越界
                //如果没越界
                if(sum >= (Integer.MIN_VALUE + num) / 10 && sum <=
(Integer.MAX_VALUE - num) / 10){
                    sum = sum * 10 + (sign * num);
                }else{//如果越界了
                    sum = sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;
                    return false;
                }
            }
            return true;//传达继续判断信号
        }
    }
```

3、总结

通过DFA解决该问题，在不新增状态的条件下，修改某些要求，大部分情况我们只需要修改上述代码构建的table表，而不需要去修改其他地方的代码