18 字符串转换整数 (atoi)

更新时间：2019-08-28 09:44:45

> 上天赋予的生命，就是要为人类的繁荣和平和幸福而奉献。
>
> ——松下幸之助

刷题内容

> 难度: Medium
>
> 原题链接：https://leetcode-cn.com/problems/string-to-integer-atoi/

内容描述

请你来实现一个 atoi 函数，使其能将字符串转换成整数。

首先，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 [−231, 231 − 1]。如果数值超过这个范围，qing返回 INT_MAX (231 − 1) 或 INT_MIN (−231) 。

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) 。

解题方案

思路 1：时间复杂度: O(N) 空间复杂度: O(1)

这道题还是有很多特殊情况，大家一定要提前充分考虑好再动笔，不然后面会一直在 debug。需要考虑比较多的边界条件&特殊情况：

1. 首先输入可能会有空格，所以先去掉空格‘；
2. 去掉空格后要考虑空字符串情况；
3. 字符串首位可能会有正负号，要考虑；
4. 开始转换成数字，题目说只要遇到非数字就可以break了；
5. 结果太大或者太小超过 `int` 限制就要返回特定数字 `2147483647` 或者 `-2147483648`；
6. 根据之前的正负号结果返回对应数值。

下面来看具体代码：

Python beats 90.11%

```
class Solution:
    def myAtoi(self, str):
        """
        :type str: str
        :rtype: int
        """
        str = str.strip()
        if len(str) == 0: # 空字符串
            return 0

        positive = True
        if str[0] == '+' or str[0] == '-': # 记录正负号
            if str[0] == '-':
                positive = False
            str = str[1:]
        elif str[0] < '0' or str[0] > '9': # 如果第一位不是数字也不是正负号，那么按照 example 4 返回 0
            return 0

        strNum = 0
        for char in str:
            if char >= '0' and char <= '9':
                strNum = strNum * 10 + ord(char) - ord('0')
            if char < '0' or char > '9': # 一旦不是数字了就不需要继续了
                break

        if strNum > 2 ** 31 - 1: # 数字越界情况
            if positive == False:
                return -(2 ** 31)
            else:
                return 2 ** 31 - 1

        if not positive: # 根据之前的正负号结果返回对应数值
            strNum = 0 - strNum
        return strNum
```


Java beats 100%

```java
class Solution {
    public int myAtoi(String str) {
        if (str == null || str.length() == 0) {
            return 0;
        }
        int start = 0;
        // 过滤前置空格
        while (start < str.length() && str.charAt(start) == ' ') {
            start++;
        }
        // 判断符号
        int sign = 1;
        if (start < str.length() && str.charAt(start) == '-') {
            sign = -1;
            start++;
        } else if (start < str.length() && str.charAt(start) == '+') {
            sign = 1;
            start++;
        }

        int ret = 0;
        while (start < str.length()) {
            if (str.charAt(start) <= '9' && str.charAt(start) >= '0') {
                if (sign > 0) {
                    // 判断越界
                    if (ret > 214748364 || (ret == 214748364 && str.charAt(start)> '7')) {
                        return 2147483647;
                    }
                    ret = ret * 10 + (str.charAt(start) - '0');
                } else {
                    // 判断越界
                    if (ret < -214748364 || (ret == -214748364 && str.charAt(start) > '8')) {
                        return -2147483648;
                    }
                    ret = ret * 10 - (str.charAt(start) - '0');
                }
            } else {
                break;
            }
            start++;
        }
        return ret;
    }
}
```


Go beats 91.39%

```go
func myAtoi(str string) int {
    strAfterStrip := strings.Trim(str, " ")
    if strAfterStrip == "" { // 空字符串
        return 0
    }

  sumsRune := make([]rune, 0)
  for i, _ := range strAfterStrip {
   sumsRune = append(sumsRune, rune(strAfterStrip[i]))
  }

    var positive bool = true
    if sumsRune[0] == rune('+') || sumsRune[0] == rune('-') { // 记录正负号
        if sumsRune[0] == rune('-') {
            positive = false
        }
        sumsRune = sumsRune[1:]
    } else if sumsRune[0] < '0' || sumsRune[0] > '9' { // 如果第一位不是数字也不是正负号，那么按照 example 4 返回 0
        return 0
    }

    strNum := new(big.Int)
    for _, ru := range sumsRune {
        if ru >= '0' && ru <= '9' {
            strNum = strNum.Add(strNum.Mul(strNum, big.NewInt(10)), big.NewInt(int64(ru - '0')))
        }
        if ru < '0' || ru > '9' { // 一旦不是数字了就不需要继续了
            break
        }
    }

    if strNum.Cmp(big.NewInt(1 << 31 - 1)) == 1 { // 数字越界情况
        if !positive {
            res, _ := strconv.Atoi(big.NewInt(-(1 << 31)).String())
            return res
        } else {
            res, _ := strconv.Atoi(big.NewInt((1 << 31 - 1)).String())
            return res
        }
    }

    res, _ := strconv.Atoi(strNum.String())
    if !positive { // 根据之前的正负号结果返回对应数值
        res = -res
    }
    return res
}
```


c++ beats 97.35%

```
class Solution {
public:
    int myAtoi(string str) {
        int start = 0;
        //过滤前置空格
        while (start < str.size() && str[start] == ' ') {
            start++;
        }
        //判断符号
        int sign = 1;
        if (str[start] == '-') {
            sign = -1;
            start ++;
        } else if (str[start] == '+') {
            sign = 1;
            start++;
        }

        int ret = 0;
        while (start < str.size()) {
            if (str[start] <= '9' && str[start] >= '0') {
                if (sign > 0) {
                    //判断越界
                    if (ret > 214748364 || (ret == 214748364 && str[start] > '7')) {
                        return 2147483647;
                    }
                    ret = ret * 10 + (str[start] - '0');
                } else {
                    //判断越界
                    if (ret < -214748364 || (ret == -214748364 && str[start] > '8')) {
                        return -2147483648;
                    }
                    ret = ret * 10 - (str[start] - '0');
                }
            } else {
                break;
            }
            start++;
        }
        return ret;
    }
};
```


小结

- 一定要注意大数溢出；
- 考虑到负数，0等等edge case；
- 提前做一些处理，方便后面的逻辑判断。

}

17 Z字形变换

19 三数之和