

22张图带你深入剖析前缀、中缀、后缀表达式以及表达式求值

一、基本概念

在本篇文章当中主要跟大家介绍以下几点

- 前缀、中缀和后缀表达式。
- 如何将中缀表达式转化成后缀表达式。
- 如何使用后缀表达式进行求值。

表达式求值这是一个比较经典的计算机系统基础问题，但是整个过程比较抽象，本文主要通过图解的方法帮助大家理解这个问题。

表达式介绍

后缀表达式也称作**逆波兰表达式**，前缀表达式也称作**波兰表达式**，这个是因为这是由波兰数学家**杨-武卡谢维奇**提出来的，用于简化命题逻辑的一种方法。

中缀表达式

我们常见的数学表达式就是中缀表达式，比如说： $1 + 2$ ，像这种我们从小到大经常见到的表达式就叫做中缀表达式，这个表达式的特点就是将运算符（加减乘除）放在两个操作数（数字）中间。

后缀表达式

后缀表达式和中缀表达式的最大区别就是，它不是将运算符放在操作数中间，而是**将运算符放在操作数后面**，比如上面的中缀表达式 $1 + 2$ 转化成后缀表达式就为 $12+$ 。

前缀表达式

前缀表达式就是将运算符放在操作数的前面，比如上面的中缀表达式 $1 + 2$ 转化成前缀表达式之后为 $+12$ 。

二、人脑模拟中缀表达式转后缀表达式

上面的表达式还是比较简单，可能不足以帮助大家好好理解表达式之间的转化过程。

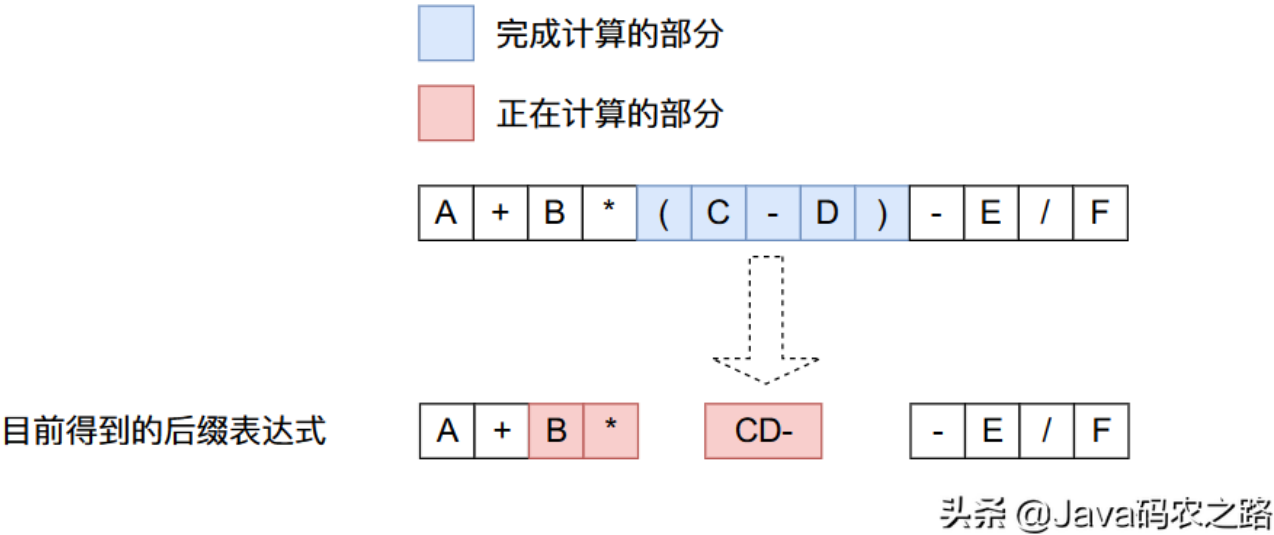
中缀表达式： $A + B * (C - D) - E / F$

现在我们来将上面的中缀表达式转化成后缀表达式，我们的第一个计算的部分如下（括号里面的优先计算）：

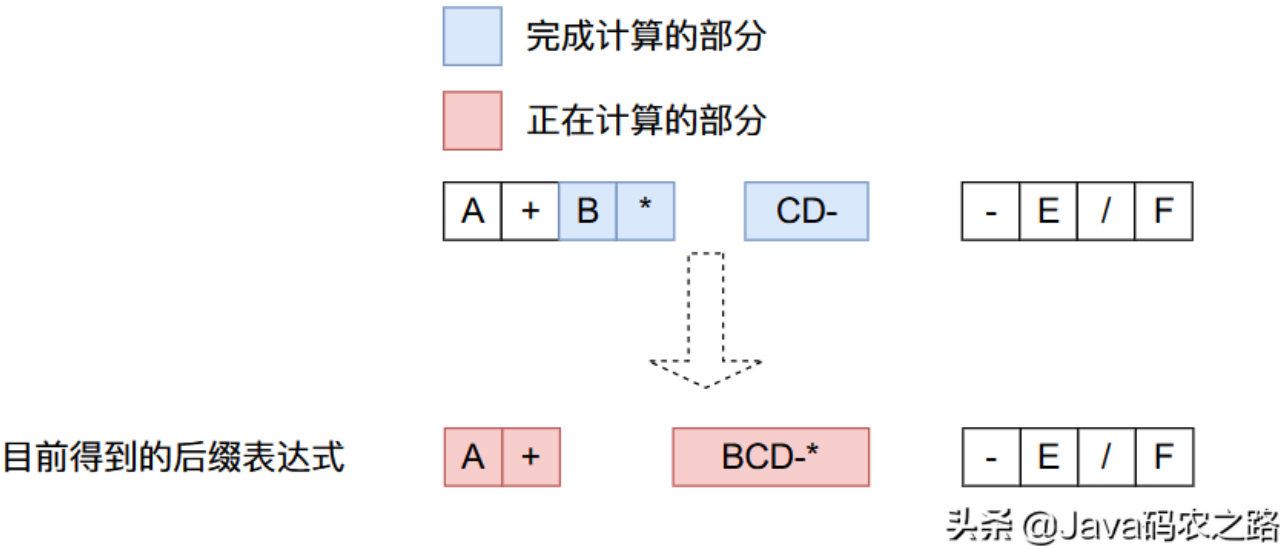
正在计算的部分

A	+	B	*	(	C	-	D	)	-	E	/	F
---	---	---	---	---	---	---	---	---	---	---	---	---

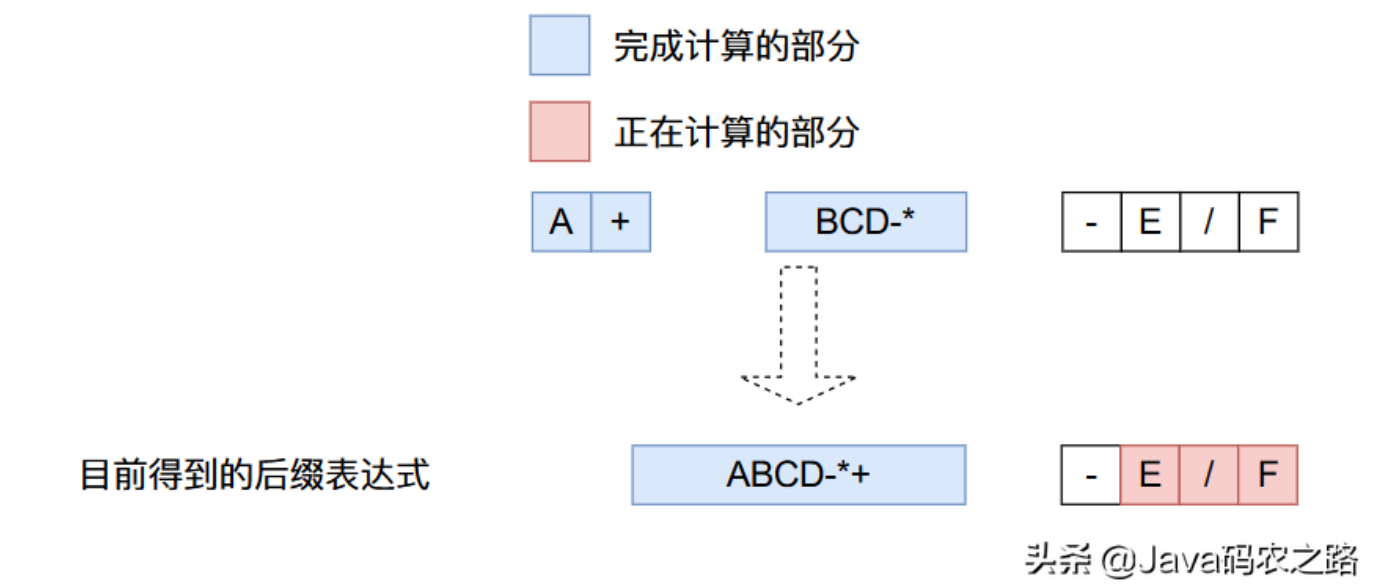
根据我们的转化原理：将**运算符**放在**操作数**后面，



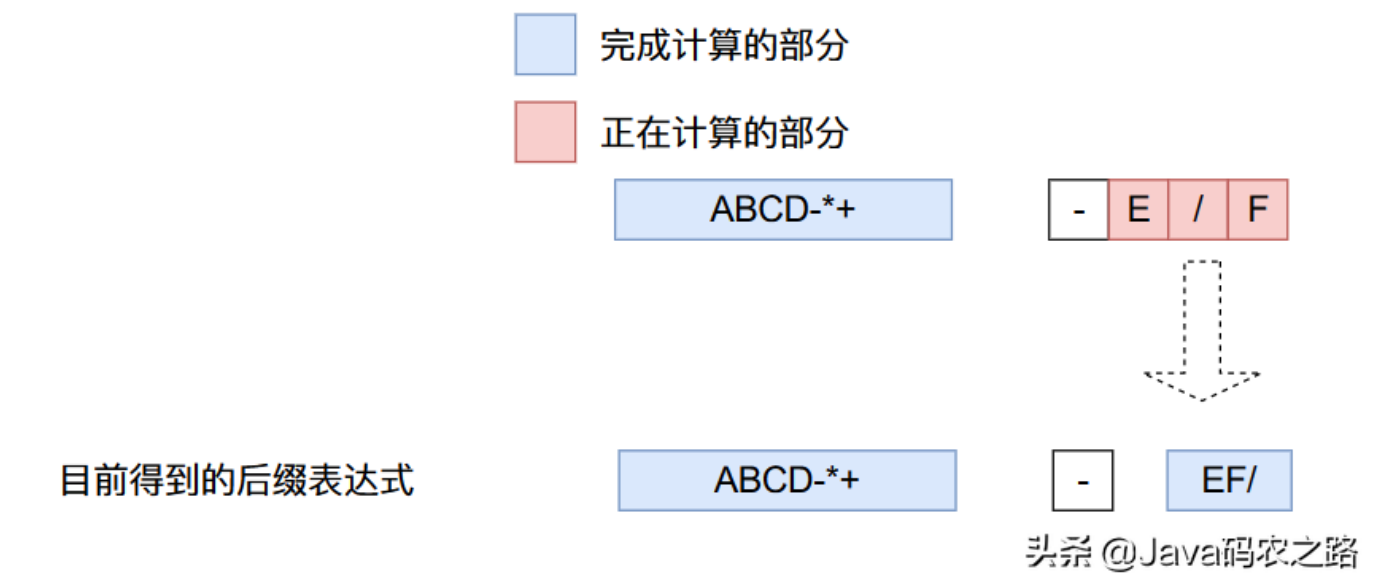
上面的得到的后缀表达式继续进行转化（现在需要计算*B*乘以后面的那个部分）：



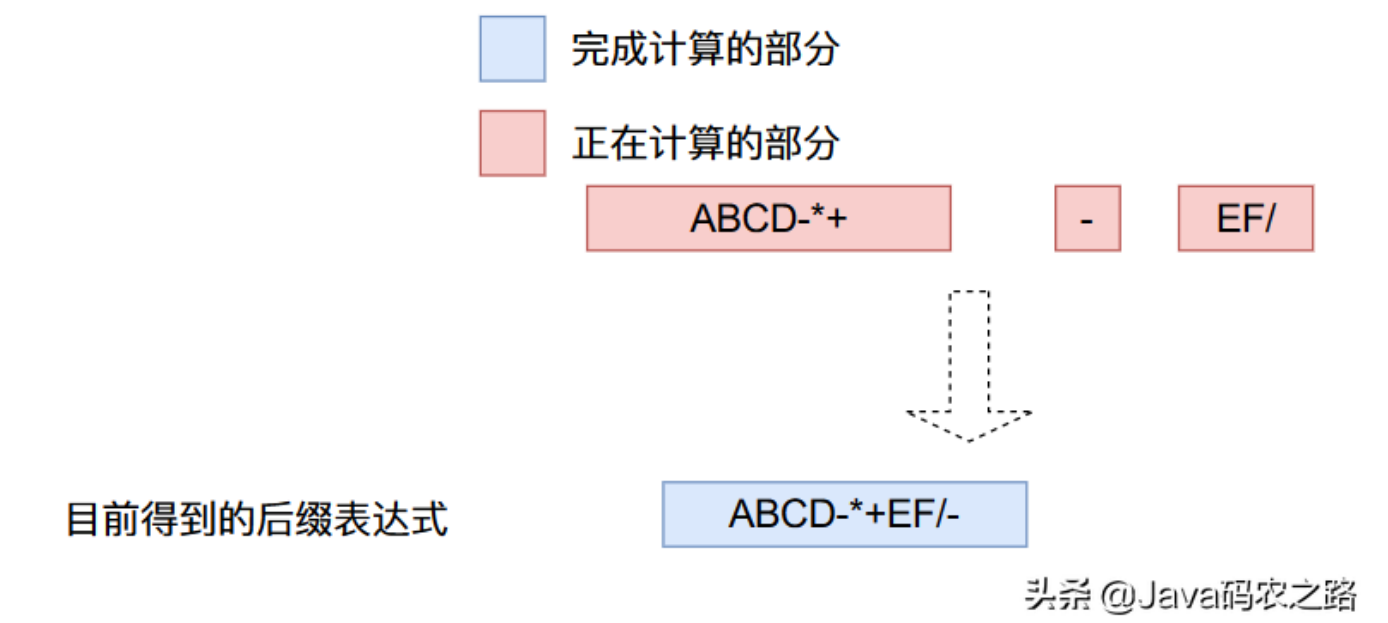
继续进行转化（从左往后进行计算）：



继续进行转化（除法的优先级比减法高）：



得到最终的结果：



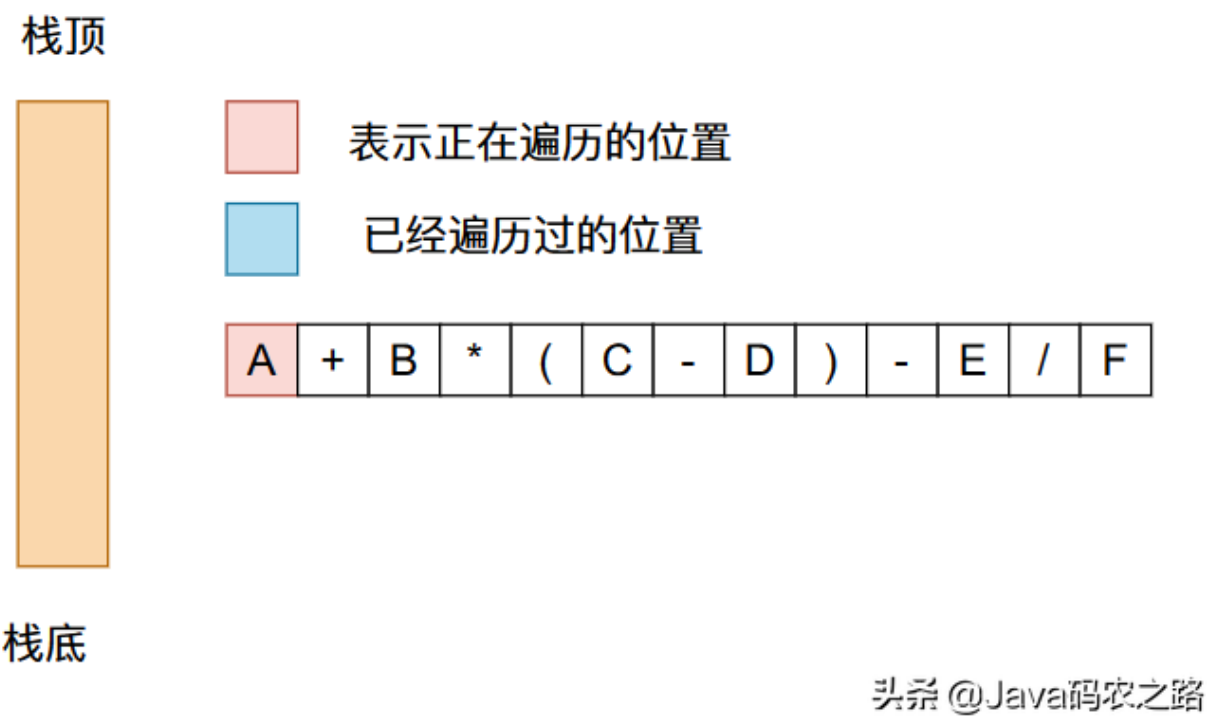
三、程序计算中缀表达式转化成后缀表达式

将中缀表达式转化成后缀表达式有以下几条规则（下面的栈是存储操作符的栈，而且只存储操作符）：

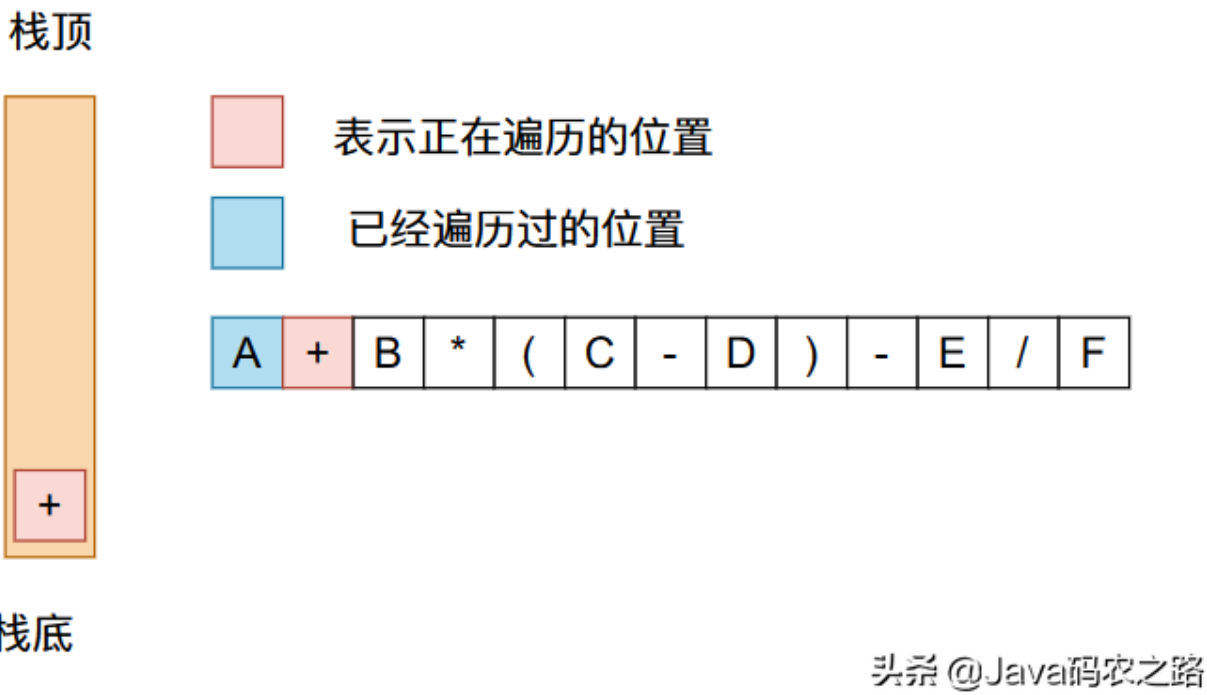
- 从左到右进行遍历。
- 遇到操作数，直接加入到后缀表达式当中。
- 遇到界限符。遇到“(“直接入栈，遇到”)“则依次弹出栈内运算符并加入后缀表达式，直到弹出“(“为止，注意:“(“不加入后缀表达式。
- 遇到运算符。依次弹出栈中优先级高于或等于当前运算符的所有运算符，并加入后缀表达式，若碰到“(“或栈空则停止。之后再把当前运算符入栈。

现在我们根据上面的规则来将上文当中的中缀表达式转化成后缀表达式。

- 遍历到A，根据第二条规则，将A加入到后缀表达式当中，当前的后缀表达式为：A。



- 现在遍历到加号，根据前面的规则需要弹出栈里面优先级大的运算符，再将加号加入到栈中，当前的后缀表达式为A。



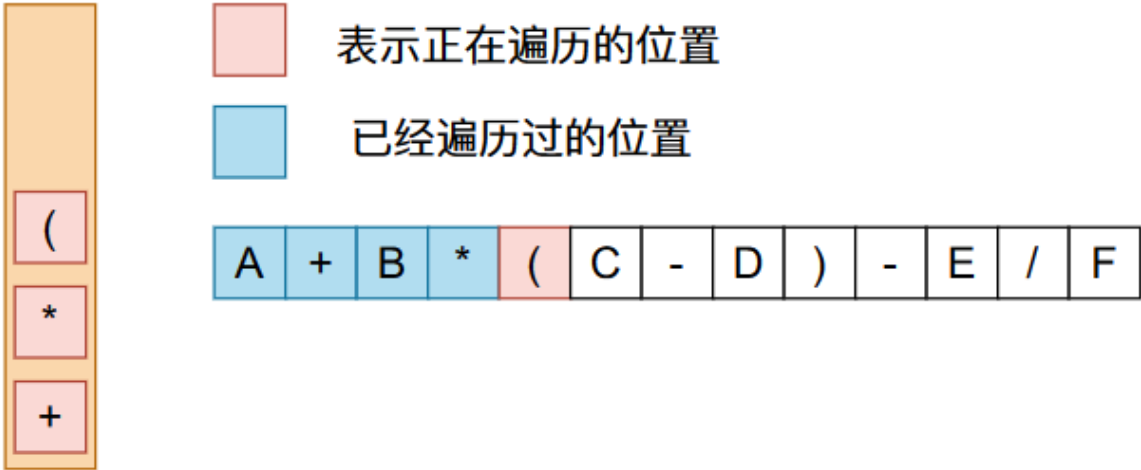
- 遍历到B，直接加入到后缀表达式当中，目前的后缀表达式为AB。



- 头条 @Java码农之路

6 / 15

栈顶

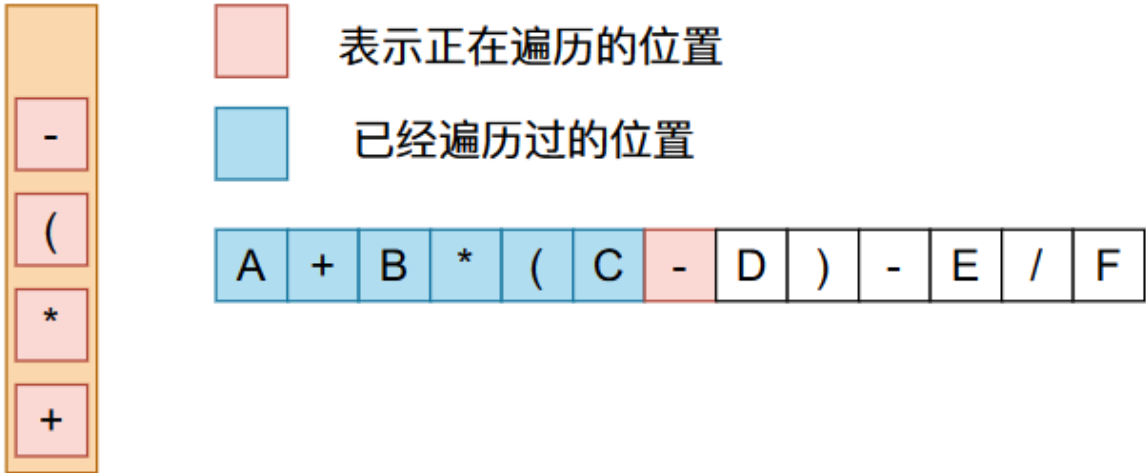


栈底

头条 @Java码农之路

- 遍历到C，则直接将其加入到后缀表达式当中，当前后缀表达式为ABC。
- 遍历到-，根据规则将其加入到栈中，当前后缀表达式为ABC。

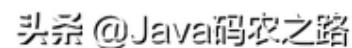
栈顶



栈底

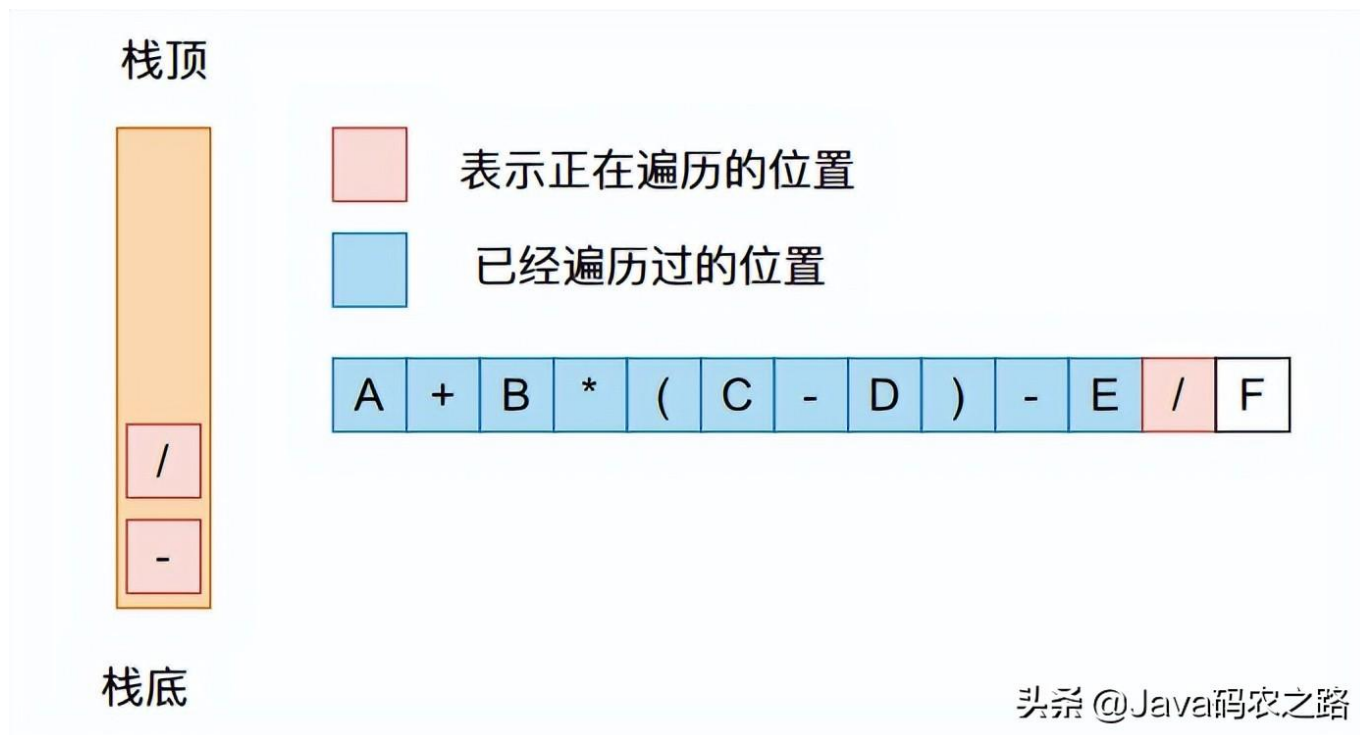
头条 @Java码农之路

- 遍历到D，则将其直接加入到后缀表达式当中，当前的后缀表达式为ABCD。
- 遍历到)，则需要将栈中的符号弹出，直到遇到(，当前的后缀表达式为ABCD-。



- 头条 @Java码农之路

- 8 / 15



- 遍历到F，直接将其加入到后缀表达式当中，则当前的后缀表达式为 $ABCD - * + EF$ 。
- 最终将栈中所有的运算符都弹出，得到的后缀表达式为 $ABCD - * + EF / -$

经过上面的过程就可以将一个中缀表达式转成后缀表达式了，大家如果想要代码实现，只需要在遍历数据的时候根据上面四个规则一个个进行判断即可，程序并不困难，就是逻辑稍微有点多，需要考虑更多的问题，在写代码的时候需要细致一点。

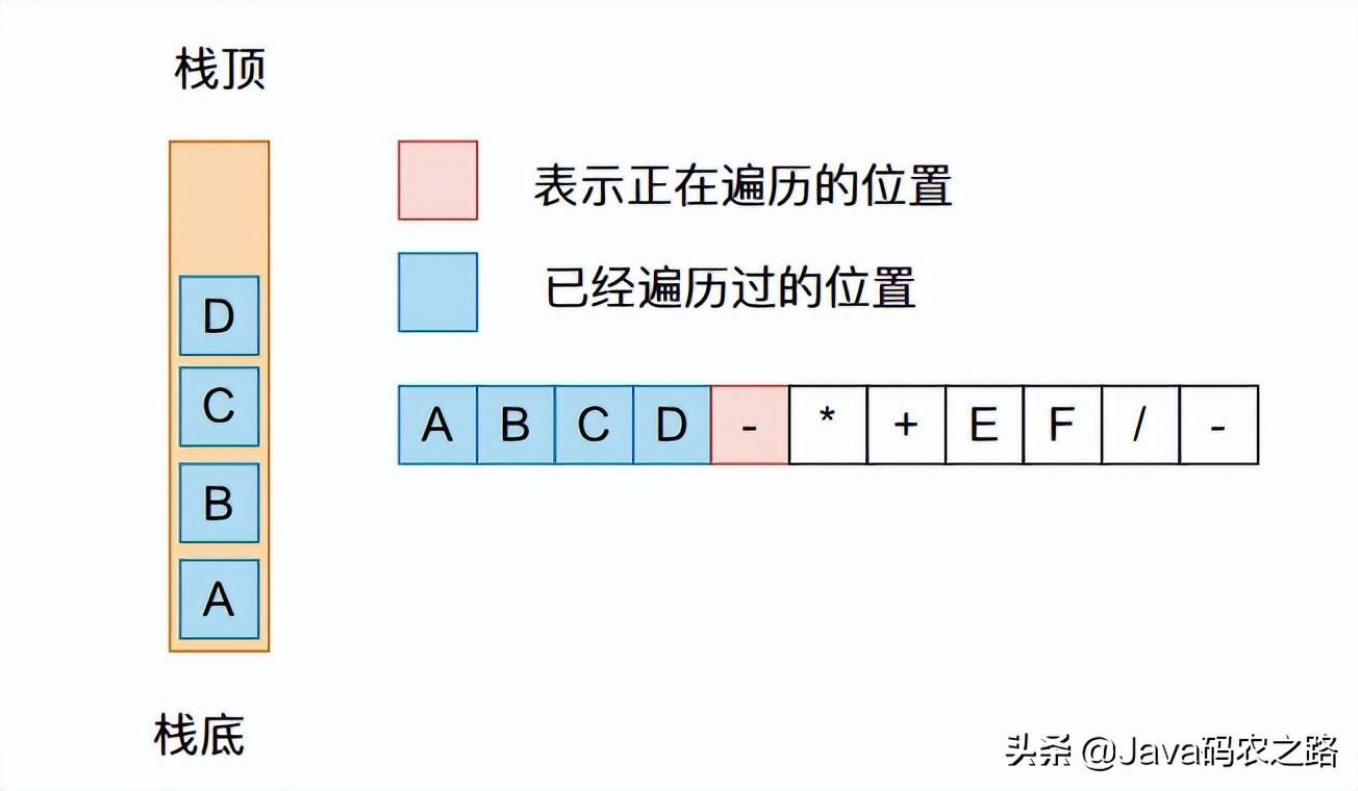
三、程序计算后缀表达式求值

在前文当中我们已经得到了表达式 $A + B * (C - D) - E / F$ 的后缀表达式为 $ABCD - * + EF / -$ 。现在我们需要将这个后缀表达式进行求值。根据后缀表达式求值主要有以下两条规则：

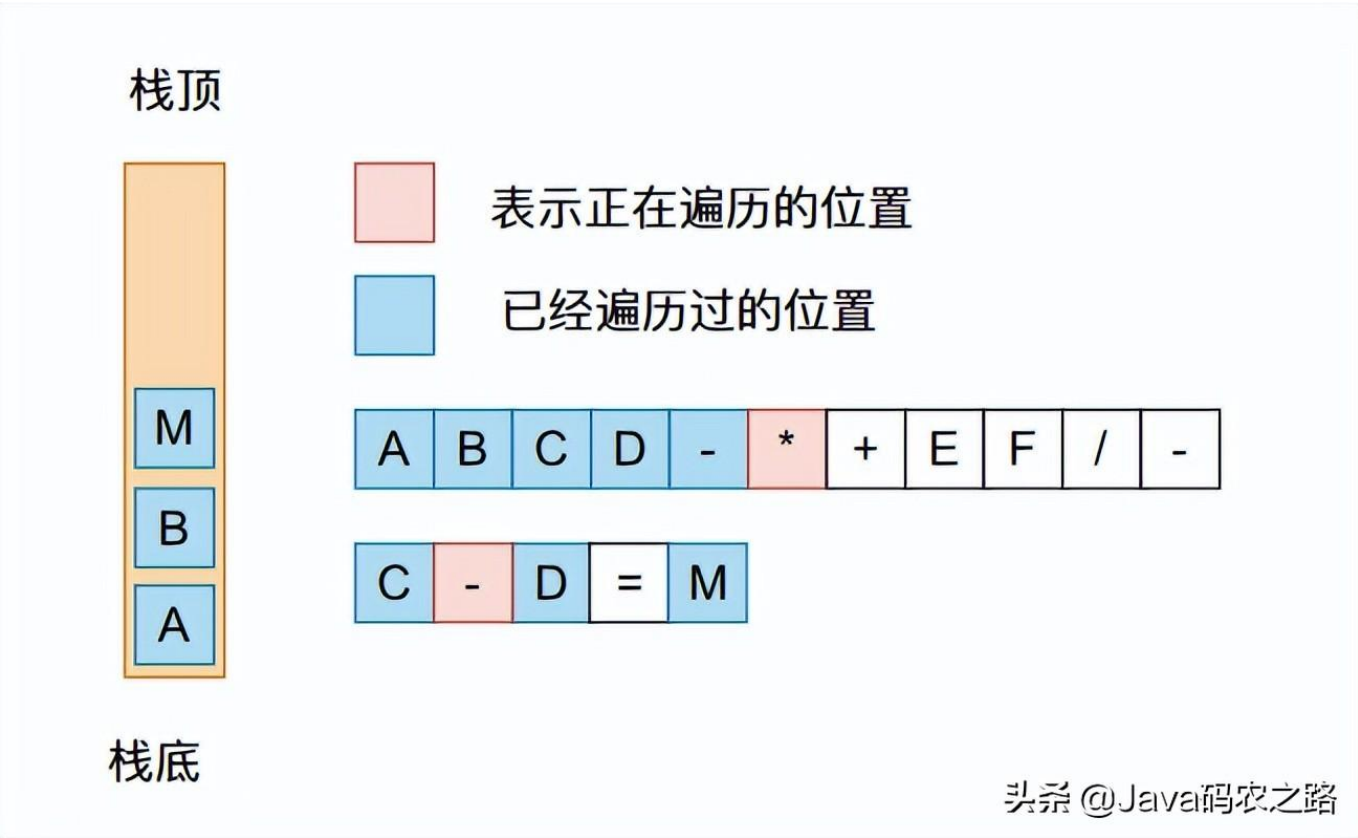
- 如果遇到数字直接将其加入到数字栈。
- 如果遇到操作符直接从操作数栈弹出两个数据进行对应的运算，再将运算结果加入到栈中。

现在我们将进行后缀表达式的求值过程:

- 首先前面四个 $ABCD$ 都是数字，根据上面提到的第一条规则，我们都需要将数字压入到栈当中，因此遍历四个数字之后，情况如下：



现在遍历到-，我们需要将D和C弹出，然后进行-操作的运算，再将结果压入栈中。



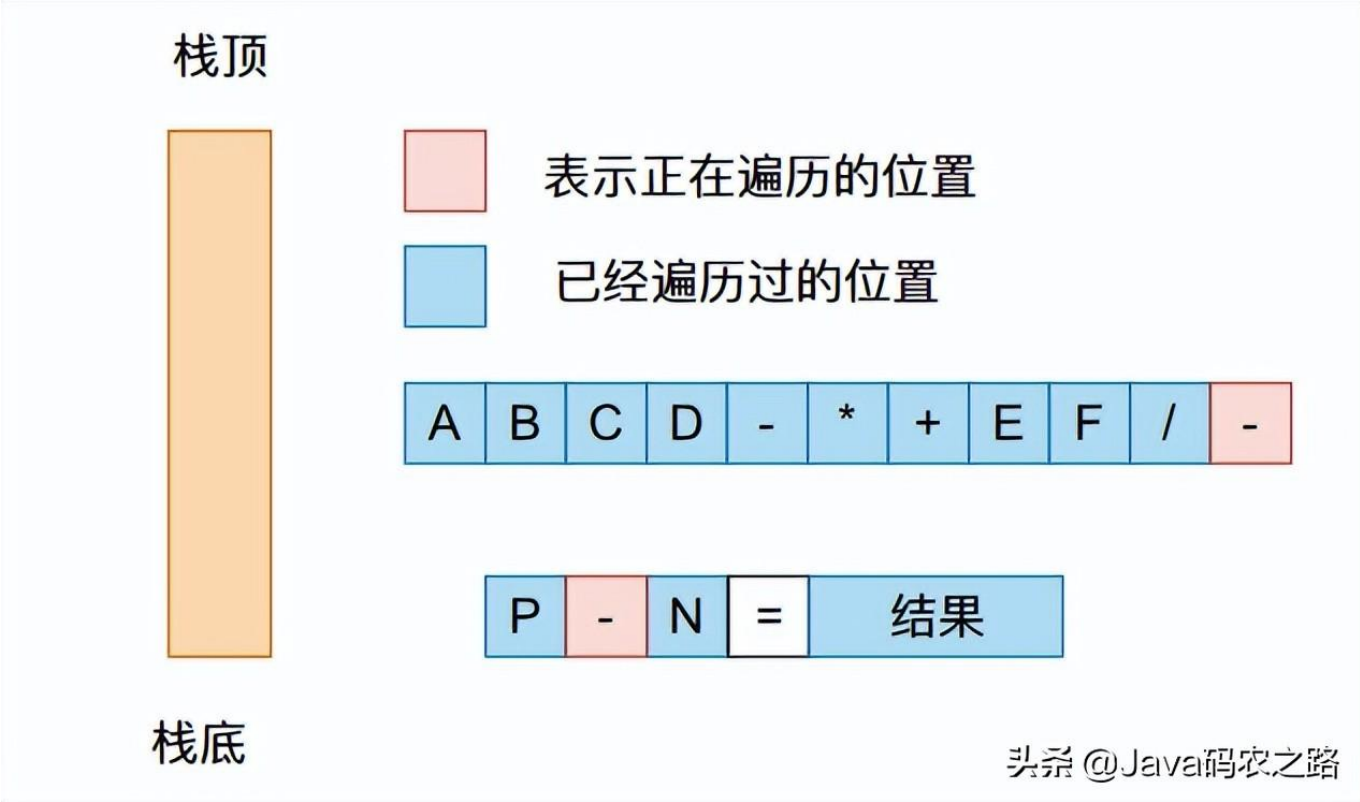
- 现在遍历到*，我们需要将C - D = M和B弹出，进行乘法运算，然后将结果压入栈中。



- 现在我们遍历到 +，需要将栈中剩余的两个数弹出，进行加法运算，再将结果压栈。



- 遍历 EF 都需要将这两个数压入到栈当中。



相信经过上面过程你对后缀表达式求值的过程已经非常清楚了。

五、计算表达式的值

```
#include <bits/stdc++.h>

using namespace std;

/*
测试用例I:
(2+2)*(1+1)

答案: 8

测试用例II: 注意: 不能在输入中带空格, 会算错!!!
2+(3*4)-((5*9-5)/8-4)

答案: 13
*/

stack<int> num; //数字栈
stack<char> op; //操作符栈

//优先级表
unordered_map<char, int> h{{'+', 1},
                           {'-', 1},
                           {'*', 2},
                           {'/', 2}};
```

```
* 功能：计算两个数的和差积商
*/
void eval() {
    int a = num.top(); //第二个操作数
    num.pop();

    int b = num.top(); //第一个操作数
    num.pop();

    char p = op.top(); //运算符
    op.pop();

    int r = 0; //结果
    //计算结果
    if (p == '+')
        r = b + a;
    else if (p == '-')
        r = b - a;
    else if (p == '*')
        r = b * a;
    else if (p == '/')
        r = b / a;
    //结果入栈
    num.push(r);
}

//读取连续的数字
void readNum(string s, int &pos, int &x) {
    while (pos < s.size() && isdigit(s[pos])) {
        x = x * 10 + s[pos] - '0';
        pos++;
    }
    //在循环里加冒了，需要返回一步，方便下一个人开始
    pos--;
}

int main() {
    //读入表达式
    string s;
    cin >> s;
    //遍历字符串的每一位
    for (int i = 0; i < s.size(); i++) {
        // 1、数字则入栈
        if (isdigit(s[i])) {
            //读出完整的数字
            int x = 0;
            readNum(s, i, x);
            num.push(x); //数字入栈
        }
        // 2、左括号无优先级，直接入栈
        else if (s[i] == '(')
            op.push(s[i]);
        // 3、右括号计算最近一对括号里面的
        else if (s[i] == ')') {

```

```
        //一直找到到左括号
        while (op.top() != '(') eval(); //将左右括号之间的计算完，维护回栈里
        //左括号出栈
        op.pop();
    } else { // 4、运算符
        //如果待入栈运算符优先级低，则先计算
        while (op.size() && h[op.top()] >= h[s[i]]) eval();
        op.push(s[i]); //操作符入栈
    }
}
// 5、//剩余的进行计算
while (op.size()) eval();
// 6、输出结果
cout << num.top() << endl;
return 0;
}
```