Java 异常与日志：表达式求值

2025 年 4 月 9 日

一、设计思路：

1. 对输入的处理思路：

我将输入的字符串全部放到一个 ArrayList 数组中，并且写了一个函数 handle 来处理这些字符串

handle 函数的逻辑：

创建一个泛型类型为 String，Number 的 Hashmap 存储变量和值的对应关系，每个字符串以空格，分号，等号符分割后，先查看第一个字符串是什么，若是 int 或者 float 则说明这是一个变量声明语句。如果是形如 int i=1；这样的语句，会被分割成三个字符串，而形如 int i；这样的语句会被分割成两个字符串，所以只需要判断分割后的字符串数组长度就可知道该语句是否为变量赋值

若该语句为变量赋值，则向 Hashmap 中添加对应的键值对，若未为变量赋值，则使用 Calculater 类中定义的字符串常量 ERRORCONSTANT 为变量赋初值，关于为什么不使用 null 赋给未定义的变量的原因会在下面解释

若第一个字符串不为类型修饰符，则为类似 i=3；这样的语句，如果在 Hashmap 中无法查出对应的键，则会抛出 VarUndefineException（变量未定义异常），若可以查出对应的键，则会更新其中的值（更新值的时候需要按照其对应的类型更新，用 instanceof 可以判断出定义变量的时候是什么类别，这就是不用 null 值赋给未赋值变量的原因，null 无法记录对应的类型信息），下面是该函数的完整代码

```
1 public static HashMap<String,Number> handle(ArrayList<String>
    language) throws VarUndefineException
2 {
3         HashMap<String,Number> varToNums = new HashMap<>();
```

```
        for(String str:language)
        {
                String[] words = str.split("[ =;]");
                if(words[0].equals("int"))
                {
                        if(words.length==3)
                        varToNums.put(words[1],Integer.parseInt
                            (words[2]));
                        else
                        varToNums.put(words[1],Integer.parseInt
                            (Calculater.ERRORCONSTANT));
                }
                else if(words[0].equals("float")) {
                        if(words.length==3)
                        varToNums.put(words[1], Float.
                            parseFloat(words[2]));
                        else varToNums.put(words[1],Float.
                            parseFloat(Calculater.ERRORCONSTANT
                            ));
                }
                else
                {
                        if(varToNums.containsKey(words[0]))
                        {
                                if(varToNums.get(words[0])
                                    instanceof Float)
                                varToNums.replace(words[0],
                                    Float.parseFloat(words[1]))
                                    ;
                                else
                                varToNums.replace(words[0],
                                    Integer.parseInt(words[1]))
                                    ;
                        }
                        else
                        throw new VarUndefineException();
                }
        }
        return varToNums;
```

```
33 }
```

2. 对表达式的处理策略

将表达式读入后，先使用正则表达式将表达式里所有的运算符号和括号提取出来放到一个数组中，然后将这些符号全部去除，将剩下的东西放在一个字符串数组中，需要注意的是，表达式中并不只有变量，还会有一些常量

如果某个字符串中含有小数点，则该值为一个 Float 型数据，会直接转化成 Float 放在一个 Object 数组 Nums 中（这样可以把所有数字都存在一个数组里，方便在表达式求值的时候进行入栈出栈操作，运算的时候判断两个操作数是否均为 Integer 实例，若是则按整数运算处理，若不是则按小数运算处理），如果该值没有小数点，则会尝试将该字符串转化为 Integer，若出现数字格式异常则说明该字符串为变量，则会去上述的 Hashmap 中寻找对应的值，若找不到对应的值则会抛出 VarUndefineException（变量未定义异常），若找到值但值为 VarUndefineException 则会抛出 VarUnassignedException（变量未赋值异常），处理结束后进行表达式求值流程

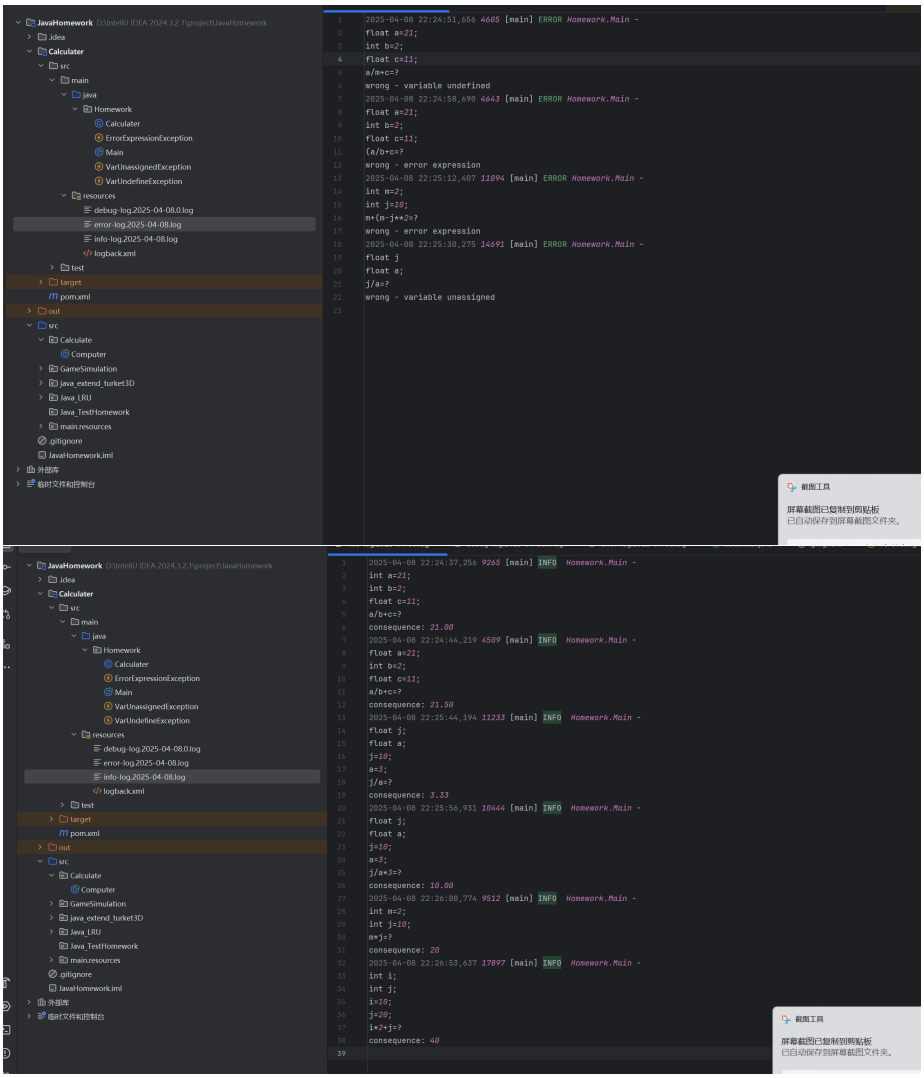
3. 表达式求值中的异常处理策略

表达式求值的具体过程在数据结构的实验报告中已经详细说明，这里不多加赘述，只说明表达式求值过程中的异常处理策略，用 try catch 语句将整个函数的主体部分包裹，如果出现 IndexOutOfBoundsException（数组越界异常），则说明表达式异常，因为该异常的触发情况多为在运算符出栈的时候操作数栈内并没有足以支持运算的操作数，而正常的表达式是不会出现此类情况的，故捕获到数组越界异常后向上抛出 ErrorExpressionException（错误表达式异常），运行到最后的时候操作数栈内应该剩余一个元素，该元素即为最后结果，但如果运行到最后发现栈内元素不为 1，则依然会抛出错误表达式异常，因为正常的表达式不会出现这种情况

4. 日志记录

更改 log.dir 值为./Calculater/src/main/resources 使日志记录在 resources 目录下，对每种过滤等级均设置自己的 appender，配置单独的过滤器，保证 info 文件中记录所有 info 记录，error 文件中记录所有 error 记录，在主函数中所有的 catch 语句中会执行进行 error 日志处理的代码，正常运行到最后会进行 info 日志的处理，源代码中会给出配置的完整 xml 文件

二. 日志记录展示：



三. 心得与收获：

1. 使用异常处理机制可以很方便地完成对各种异常的处理，让程序不会遇到异常就强制退出，增强了代码的健壮性，对各种异常类信息的设置也可以让我们很容易地定位到出错的代码段

2. 使用日志系统可以很方便地记录程序的运行情况，遇到的错误等信息，可以在测试和运维阶段提供良好的参考

3. 基于多态（将 Integer 和 Float 均存在 Object 的数组中）可以较为方便地完成将所有数字都保存在同一个数组中，保证原来在表达式中的次序的作用

4.java 工具包中的数组，链表，哈希表等数据结构是非常好的容器，熟练掌握这些容器的使用方法可以大大提高我们的效率，本例中使用 linkedlist 模拟栈，用 hashmap 处理变量和值的对应关系等均印证了这点

5. 将各个功能特定的函数封装好可以完成代码复用的工作，在设计单个类的时候尽可能提高这个类的聚合性，使单个类中的各个函数共同完成单一功能，单个类应只具有单一功能，不应把各种功能全部塞到一个类内

四. 源代码

```
1  package Homework;
2
3  import java.util.*;
4
5  public class Calculater {
6      public static final String ERRORCONSTANT="88888888";
7      private static final Float ERRORFLOAT=Float.parseFloat(
           ERRORCONSTANT);
8      public ArrayList<Character> findOperator(String input)
9      {
10         ArrayList<Character> operators = new ArrayList<>();
11         String[] output=input.split("[^-+*/%()]");
12         Arrays.stream(output).forEach(x-> {
13             if(!x.isEmpty())
14             {
15                 for (int i = 0; i < x.length(); i++) {
16                     operators.add(x.charAt(i));
17                 }
18             }
19         });
20         return operators;
21     }
22     public ArrayList<Object> findNum(String input, HashMap<
           String,Number> Nums) throws VarUndefineException,
           VarUnassignedException {
23         ArrayList<Object> nums = new ArrayList<>();
24         String[] output=input.split("[-+*/%()=?]");
25         for(String x:output)
26         {
27             if(x.isEmpty()) continue;
28             if(x.contains(".")) {nums.add(Float.parseFloat(x));
```

```
                continue;}
            try{
                nums.add(Integer.parseInt(x));
            }
            catch(NumberFormatException e){
            if(Nums.containsKey(x))
            {
                if(ERRORCONSTANT.equals(Nums.get(x).toString())
                    ||ERRORFLOAT.toString().equals(Nums.get(x).
                    toString())) throw new
                    VarUnassignedException();
                nums.add(Nums.get(x));
            }
            else throw new VarUndefineException();}
        }
        return nums;
    }
    public void calculate(LinkedList<Object> nums, Character
        operator) throws ErrorExpressionException
    {
        if(nums.size()<2) throw new ErrorExpressionException();
        Object num1 = nums.removeFirst();
        Object num2 = nums.removeFirst();
        if(num1 instanceof Integer && num2 instanceof Integer)
        {
            nums.addFirst(calculateNum((Integer) num1,(Integer)
                 num2,operator));
        }
        else
        {
            nums.addFirst(calculateNum(Float.parseFloat(num1.
                toString()),Float.parseFloat(num2.toString()),
                operator));
        }
    }
    public Integer calculateNum(Integer num2,Integer num1,
        Character operator) throws ErrorExpressionException
    {
        switch(operator){
```

```
59              case '+':{return num1+num2;}
60              case '-':{return num1-num2;}
61              case '*':{return num1*num2;}
62              case '/':{return num1/num2;}
63              case '%':{return num1%num2;}
64              default: {throw new ErrorExpressionException();}
65          }
66      }
67      public Float calculateNum(Float num2,Float num1,Character
            operator) throws ErrorExpressionException
68      {
69          switch(operator){
70              case '+':{return num1+num2;}
71              case '-':{return num1-num2;}
72              case '*':{return num1*num2;}
73              case '/':{return num1/num2;}
74              case '%':{return num1%num2;}
75              default: {throw new ErrorExpressionException();}
76          }
77
78      }
79      public Number calculate(String expression,HashMap<String,
            Number> varToNumber) throws ErrorExpressionException,
            VarUndefineException, VarUnassignedException {
80          ArrayList<Object>Nums=findNum(expression,varToNumber);
81          ArrayList<Character> operators = findOperator(
                expression);
82          LinkedList<Object> nums = new LinkedList<>();
83          LinkedList<Character> Operators=new LinkedList<>();
84          int j=0;
85          nums.addFirst(Nums.get(j++));
86          try{
87              for (int i = 0; i < operators.size(); i++) {
88                  if(operators.get(i)=='(') {Operators.addFirst('
                        (');}
89                  else if(operators.get(i)==')') {
90                      if(!Operators.isEmpty())
91                      {
92                          while(Operators.getFirst()!='('){
```

```
93                          calculate(nums,Operators.
                                removeFirst());
94                      }
95                  }
96                  Operators.removeFirst();
97              }
98              else if(operators.get(i)=='+' ||operators.get(i
                    )=='-')
99              {
100                 if(!Operators.isEmpty()){
101                     while(!Operators.isEmpty()&&Operators.
                            getFirst()!='('){
102                         calculate(nums,Operators.
                                removeFirst());
103                     }
104                 }
105                 nums.addFirst(Nums.get(j++));
106                 Operators.addFirst(operators.get(i));
107             }
108             else if(operators.get(i)=='*' ||operators.get(i
                    )=='/'||operators.get(i)=='%')
109             {
110                 if(!Operators.isEmpty()){
111                     while(!Operators.isEmpty()&&Operators.
                            getFirst()!='('&&Operators.getFirst
                            ()!='+'&&Operators.getFirst()!='-')
                            {
112                         calculate(nums,Operators.
                                removeFirst());
113                     }
114                 }
115                 nums.addFirst(Nums.get(j++));
116                 Operators.addFirst(operators.get(i));
117             }
118         }
119     }catch(IndexOutOfBoundsException e){
120         throw new ErrorExpressionException();
121     }
122     try{
```

```
            while(!Operators.isEmpty())
            {
                calculate(nums,Operators.removeFirst());
            }
        }catch(Exception e){
            throw new ErrorExpressionException();
        }
        if(nums.size()!=1) throw new ErrorExpressionException()
            ;
        return (Number)nums.get(0);
    }
}
```

```
package Homework;

public class ErrorExpressionException extends Exception {
    public ErrorExpressionException() {
        super("wrong - error expression");
    }
}
```

```
package Homework;

import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

@Slf4j
public class Main {

    public static HashMap<String,Number> handle(ArrayList<
        String> language) throws VarUndefineException
    {
        HashMap<String,Number> varToNums = new HashMap<>();
        for(String str:language)
        {
            String[] words = str.split("[ =;]");
```

```
18              if(words[0].equals("int"))
19              {
20                  if(words.length==3)
21                      varToNums.put(words[1],Integer.parseInt(
                            words[2]));
22                  else
23                      varToNums.put(words[1],Integer.parseInt(
                            Calculater.ERRORCONSTANT));
24              }
25              else if(words[0].equals("float")) {
26                  if(words.length==3)
27                      varToNums.put(words[1], Float.parseFloat(
                            words[2]));
28                  else varToNums.put(words[1],Float.parseFloat(
                        Calculater.ERRORCONSTANT));
29              }
30              else
31              {
32                  if(varToNums.containsKey(words[0]))
33                  {
34                      if(varToNums.get(words[0]) instanceof Float
                            )
35                          varToNums.replace(words[0],Float.
                                parseFloat(words[1]));
36                      else
37                          varToNums.replace(words[0],Integer.
                                parseInt(words[1]));
38                  }
39                  else
40                      throw new VarUndefineException();
41              }
42          }
43          return varToNums;
44      }
45      public static void main(String[] args) {
46          ArrayList<String> input = new ArrayList<>();
47          Scanner scanner = new Scanner(System.in);
48          String str;
49          StringBuilder logInput= new StringBuilder();
```

```
50          str = scanner.nextLine();
51          try{
52              while(!str.isEmpty())
53              {
54                  logInput.append("\n");
55                  logInput.append(str);
56                  input.add(str);
57                  str = scanner.nextLine();
58              }
59          }
60          catch(Exception e){}
61          //log.info(logInput.toString());
62          String expression=input.removeLast();
63          Calculater calculater = new Calculater();
64          HashMap<String,Number> varToNumber;
65          try{
66              varToNumber=handle(input);
67          } catch (VarUndefineException e) {
68              log.error(logInput.toString()+"\n"+e.getMessage());
69              return;
70          }
71          try{
72              Number consequence=calculater.calculate(expression,
                    varToNumber);
73              if(consequence instanceof Integer) log.info(
                    logInput.toString()+"\n"+"consequence:␣"+
                    consequence);
74              else log.info(String.format(logInput.toString()+"\n
                    "+"consequence:␣%.2f",consequence));
75          }
76          catch (ErrorExpressionException e) {
77              log.error(logInput.toString()+"\n"+e.getMessage());
78          } catch (VarUndefineException e) {
79              log.error(logInput.toString()+"\n"+e.getMessage());
80          } catch (VarUnassignedException e) {
81              log.error(logInput.toString()+"\n"+e.getMessage());
82          }
83      }
84  }
```

```java
package Homework;

public class VarUnassignedException extends Exception {
    public VarUnassignedException() {
        super("wrong - variable unassigned");
    }
}
```

```java
package Homework;

public class VarUndefineException extends Exception{
    VarUndefineException(){
        super("wrong - variable undefined");
    }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="60 seconds">

    <!-- 定义变量 -->
    <property name="log.maxSize" value="10MB"/>
    <property name="log.totalSizeCap" value="10GB"/>
    <property name="log.dir" value="./Calculater/src/main/
        resources"/>
    <property name="log.maxHistory" value="30"/>
    <property name="append" value="true" /> <!-- 确保追加模式启
        用 -->

    <appender name="ERROR" class="ch.qos.logback.core.rolling.
        RollingFileAppender">
        <!-- 使用 ThresholdFilter 允许 ERROR 及以上级别 -->
        <filter class="ch.qos.logback.classic.filter.
            ThresholdFilter">
            <level>ERROR</level>
        </filter>
        <append>${append}</append>
        <rollingPolicy class="ch.qos.logback.core.rolling.
            TimeBasedRollingPolicy">
```

```
18            <fileNamePattern>${log.dir}/error-log.%d{yyyy-MM-dd
                  }.log</fileNamePattern>
19            <maxFileSize>${log.maxSize}</maxFileSize>
20            <maxHistory>${log.maxHistory}</maxHistory>
21            <totalSizeCap>${log.totalSizeCap}</totalSizeCap>
22        </rollingPolicy>
23        <encoder>
24            <pattern>%d %-4relative [%thread] %-5level %logger
                  - %msg%n</pattern>
25        </encoder>
26    </appender>
27
28
29    <appender name="INFO" class="ch.qos.logback.core.rolling.
          RollingFileAppender">
30        <filter class="ch.qos.logback.classic.filter.
              LevelFilter">
31            <level>INFO</level>
32            <onMatch>ACCEPT</onMatch>
33            <onMismatch>DENY</onMismatch>
34        </filter>
35        <append>${append}</append>
36        <rollingPolicy class="ch.qos.logback.core.rolling.
              TimeBasedRollingPolicy">
37            <fileNamePattern>${log.dir}/info-log.%d{yyyy-MM-dd
                  }.log</fileNamePattern>
38            <maxFileSize>${log.maxSize}</maxFileSize>
39            <maxHistory>${log.maxHistory}</maxHistory>
40            <totalSizeCap>${log.totalSizeCap}</totalSizeCap>
41        </rollingPolicy>
42        <encoder>
43            <pattern>%d %-4relative [%thread] %-5level %logger
                  - %msg%n</pattern>
44        </encoder>
45    </appender>
46
47    <appender name="DEBUG" class="ch.qos.logback.core.rolling.
          RollingFileAppender">
48        <append>true</append>
```

```
        <filter class="ch.qos.logback.classic.filter.
            LevelFilter">
            <level>DEBUG</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <rollingPolicy class="ch.qos.logback.core.rolling.
            SizeAndTimeBasedRollingPolicy">
            <fileNamePattern>${log.dir}/debug-log.%d{yyyy-MM-dd
                }.%i.log</fileNamePattern>
            <maxFileSize>${log.maxSize}</maxFileSize>
            <maxHistory>${log.maxHistory}</maxHistory>
            <totalSizeCap>${log.totalSizeCap}</totalSizeCap>
        </rollingPolicy>
        <encoder>
            <pattern>%d %-4relative [%thread] %-5level %logger
                - %msg%n</pattern>
        </encoder>
    </appender>


    <root level="INFO" additivity="false">
        <!-- 已完全移除 STDOUT 引用 -->
        <appender-ref ref="DEBUG"/>
        <appender-ref ref="INFO"/>
        <appender-ref ref="ERROR"/>
    </root>

</configuration>
```