

Java-03

一、Task1.变量和数据类型

1. 八种基本数据类型：

byte short int long char float double boolean

特别说明： string 属于 引用类型 而非 基本类型

2.

byte	1	-128~127
数据类型	字节数	范围
short	2	$-2^{15} \sim (2^{15}-1)$
int	4	$-2^{31} \sim (2^{31}-1)$
long	8	$-2^{63} \sim (2^{63}-1)$

◦ 显式类型转换 即 强制类型转换

通常使用这种格式：**数据类型 b = (数据类型) a**

表示将 **变量a** 强制转换为该数据类型并赋值给 **变量b**

值得注意的是 1.浮点型 转为 整型 将只保留整数部分

2.显然可能造成数据溢出

◦ 隐式类型转换 即 自动类型转换

```
byte → short → int → long → float → double
                        ↑
                        char
```

遵循以上转换，即大范围类型可以接收小范围类型

何时转换 ☹️ ? 1.赋值时（包括方法调用时赋值）

2.运算时 **值得注意的是**

byte short char 都转换为 **int** 运算

3.

```
int a=4;
char c='0';
int b=a+c;
```

如上为 **隐式类型转换 b=52**

原理： 运算时 **char** 转为 **int** 计算，'0'的ASCII码为48

则 **b=4+48=52**

4. 最开始被这个问题难倒了 ☹️

原本认为 **Integer** 作为 **包装类** 应当等同于指针，存储地址（则全F） 🤔

结果运行了一下让我目瞪口呆😱

```
false
true
false
```

于是开始比较后两句的不同：看来只能是**赋值大小**导致了这一切

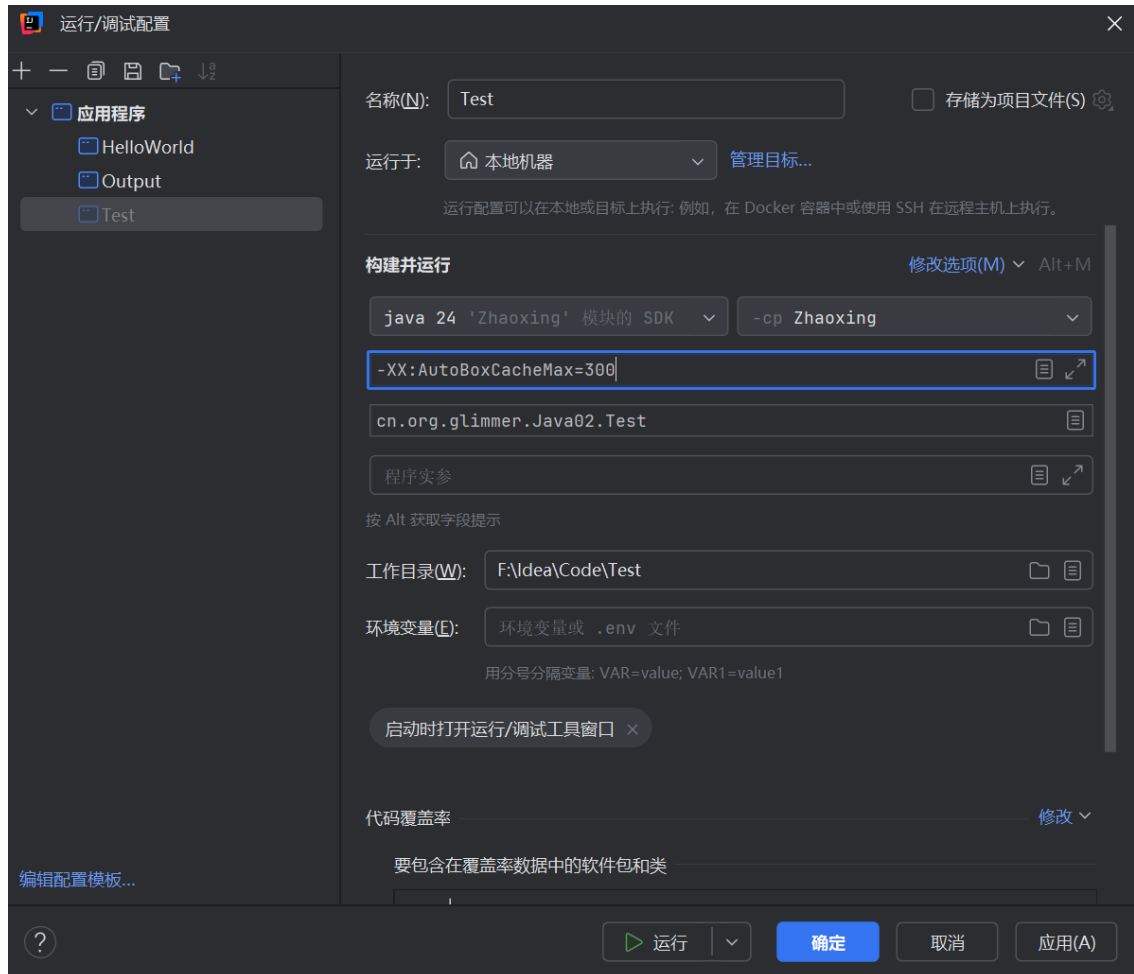
可是为什么呢？😱仔细阅读题干找到了关键信息**缓存**

啊😱确实如此。

查阅资料后我发觉 **Integer** 对于-128~127的数据使用缓存中 **同一对象**

而对于更大数据则生成了 **新的对象**，地址当然不同了

并且这个缓存数值可以通过对VM的修改来改变



如上，再尝试运行代码😱

```
false
true
true
```


可喜可贺，可喜可贺😱

二、Task2.运算符

1.

```
int a = 5 ;
int b = 7 ;
int c = (++a) + (b++)
System.out.println( c );
System.out.println(a+ " "+b);
```

结果:



```
13
6 8
```

解释: `++a` 表示在逻辑运算前使 **变量a** 自增

相应地, `b++` 表示在逻辑运算后使 **变量b** 自增

所以对于 `int c = (++a) + (b++)` 运算时 **a=5+1, b=7**

运算后 **a=6, b=7+1**

因此有如上结果

2. 数据储存

- **int** 较为简单, **正数**直接转换为二进制存储
负数将对应正数转换为二进制, 第一位(符号位)取1, 其余位取反, 再加1即可

- **float** 则相对复杂

首先需要知道, **float** 有**32**位,

它们被分成了 **1+8+23** 分别是 **符号位 指数位 尾数位**

1. **符号位** 顾名思义 **0为正数, 1为负数**
2. **指数位** 和 **尾数位** 需要先将其整数与小数部分分别转为二进制

特别说明 小数部分进行乘2取整数部分 小数部分继续操作到0

这些整数部分组合成为二进制的小数部分

再使用科学计数法得到 **指数** 和 **尾数 (即小数部分)**

指数位 为 (**指数+127**)转二进制并补位得到

其中127为 **偏移码** (为了防止负指数的比较大小小时大于正指数故将所有指数增大偏移成正数比较)

尾数位 为二进制的小数部分后面补零成23位得到

三、Task3.数据结构

真正编写时遇到了很多问题☹

最开始觉得看起来很简单, 还以为只需要冒泡排序就完成

实际操作的时候决定学习并使用一下 **HashMap**, 结果发觉 **HashMap**

并不是直接按照 **put** 的顺序输出的☹

事实上 **HashMap** 并不保证 **插入顺序**

经过我的查找, 总算基本知道了 **HashMap** 的 **插入顺序**

HashMap是通过**hash算法**把**key**映射到table数组的某个位置

首先对 **key** 运算得到 **Hashcode**, 方式做如下简要说明(**String**)

- 设 **h** 为 **HashCode**，令 **h=0**，算法为 对每一位字符做如下操作

$h = h * 31 + \text{该位ASCII码}$ 直至最后一位字符

很好 😊 我们现在得到了 **HashCode**

那么 **HashCode** 会是决定插入顺序的数值吗？🤔

很遗憾，仍然不是 ~~（悲）~~

查询资料后，我们注意到了关键源码

```
newTab[e.hash & (newCap - 1)] = e;
```

其中，**e.hash** 为 **HashCode**值

newCap 为 **HashMap** 扩容后大小

令人高兴的是，**e** 就是最终比较所用的数值

让我们来实操一下 ✨ 执行以下代码看看顺序 😊

```
package cn.org.glimmer.Java02;
import java.util.HashMap;

public class Test {
    public static void main(String[] args) {
        HashMap<String,Integer> map = new HashMap<>();
        for (int i = 0;i<15;i++){
            map.put(String.valueOf(i),i);
        }
        System.out.println(map);
    }
}
```

```
F:\Idea\Java\jdk-24.0.2\bin\java.exe "-javaagent:F:\Idea\IntelliJ IDEA 2025.1.1.1\lib\ide
{11=11, 12=12, 13=13, 14=14, 0=0, 1=1, 2=2, 3=3, 4=4, 5=5, 6=6, 7=7, 8=8, 9=9, 10=10}

进程已结束，退出代码为 0
```

对应 **HashCode** 分别为1568 1569 1570 1571 48 49 50 51 ... 57 1567

这显然不是直接按照 **HashCode** 排序的

那么我们再计算一下 **e** 吧，**值得说明的是** 扩容大小只能是2的次幂

因此按照我的理解，此时扩容后大小应为32

因此 **e** 分别等于0 1 2 3 16 17 18 ... 25 31

符合实际情况，看来的确如此

好了，让我们**回归正题**，距离解决问题还有天堑之隔

难题有这些：

- **HashMap** 的排序
- 初始数据处理和输入数据加入到已有的 **HashMap** 中

这困扰了我很久，我发现没有思路处理这样的问题

在网上查询学习之后，我得知了一种方法十分管用

那便是 **stream流** (写来轻松的过程其实花了我很多精力和时间)

需要注意的是 **stream** 并非数据结构，它并不保存数据

它由三个部分组成：

1. **创建流**：方式如下

- **使用集合创建** 直接使用 **Map或者List** 来创建，例如：

```
Map<String,Integer> map = new HashMap<>;
Map<String,Integer> Steam = map.entrySet.stream()
```

```
List list = Arrays.asList(1,2,3,4,5,6,7,8,9,10);
Stream<Integer> stream = list.stream();
```

- **使用数组创建**

```
String[] arr = new String[]{"a", "b", "c"};
Stream<String> stream = Arrays.stream(arr);
```

2. **中间操作**：

- **排序 sorted()** 自然排序
sorted (Comparator) 或者sorted(Map) 调用其他接口排序
- **筛选切片**
filter 过滤 **limit(n)** 获取n个元素
skip(n) 跳过n个元素
- **映射 map** 用一个函数映射生成一个新的集合并存在该函数中

3. **结束操作**：很多操作，只列举收集

- **collect** 调用 **Collector** 收集
- **toList等** 转换成 **List, String, Array**

大概了解 **stream** 之后便可以开始操作了，详细代码如下

([student.java](#))

已上传仓库 

```
import java.util.*;
import java.util.stream.Collectors;

public class student {
    private String name;
    private Map<String,Integer> subjects;
    //创建成员
    public void addnum(String subject,int num){
        subjects.put
        (subject,subjects.getOrDefault(subject,0)+num);
    }
}
```

```

//构建加值方法，使输入直接加到value当中
public static void main(String[] args) {
    student xiaoming = new student();
    student xiaohong = new student();
    xiaoming.subjects = new HashMap<>();
    xiaohong.subjects = new HashMap<>();
    xiaoming.name = ("小明");
    xiaohong.name = ("小红");
    String initialdata =
"math:5,English:10,Chinese:10,math:20,English:10,chemistry:30,math:10,math:20";
    Arrays.stream(initialdata.split(","))
        //创建stream并切除“，”
        .map(a -> a.split(":"))
        //建立映射使用a接收切除“：”后划分出的集合
        .forEach(a -> xiaoming.addnum(a[0], Integer.parseInt(a[1])));
    //对元素分别使用加值方法，一个key一个value
    int ans = 1; //声明循环判断变量
    do {
        System.out.println("请输入学生 科目 错题数目: ");
        Scanner input = new Scanner(System.in);
        String str = input.nextLine();
        List<String> initialadd = Arrays.stream(str.split(" ")).toList();
        //收集输入并切分装入List
        String stu = initialadd.get(0);
        //第一个元素是学生
        if (stu.equals(xiaoming.name)) {

            if (initialadd.size() == 3)
                //判断输入是否总共三个值，防止报错
                {
                    xiaoming.addnum(initialadd.get(1),
Integer.parseInt(initialadd.get(2)));
                }
            Map<String,Integer> sorteddata =
xiaoming.subjects.entrySet()
                .stream()
                //键值对做成stream
                .sorted(Map.Entry.
<String,Integer>comparingByValue().reversed())
                //利用Map.Entry中的排序
                //reversed是倒序
                .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
//分别用getKey和getValue得到两个map，后续必须合成并接收
                    (e1, e2) -> e1, LinkedHashMap::new));
            //把刚才的两个map合成一个map并保存在一个LinkedHashMap中

            System.out.println("小明的错题数目从高到低依次为: ");
            System.out.println(sorteddata);
            //输出排序数组
            System.out.println("按0继续输入: ");
            ans = input.nextInt();
        }
        else if (stu.equals(xiaohong.name)) {
            if (initialadd.size() == 3) {
                xiaohong.addnum(initialadd.get(1),
Integer.parseInt(initialadd.get(2)));
                Map<String,Integer> sorteddata =
xiaohong.subjects.entrySet()

```

```

        .stream()
        .sorted(Map.Entry.
<String,Integer>comparingByValue().reversed())
        .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
            (e1, e2) -> e1, LinkedHashMap::new));

        //排序原理同上
        System.out.println("小红的错题数目从高到低依次为: ");
        System.out.println(sorteddata);
        //输出排序数组
        System.out.println("按0继续输入: ");
        ans = input.nextInt();
    }
    else {
        System.out.println("不存在小红的数据");
        System.out.println("按0重新输入: ");
        ans = input.nextInt();
    }

}

else{
    System.out.println("您输入的内容有误");
    System.out.println("按0重新输入: ");
    ans = input.nextInt();

}

}

while (ans == 0); //循环判断
System.out.println("退出系统");
}
}

```

大概是完成了