

# JAVA 语言基础上机练习

陈德创

2020 年 3 月 12 日

## 目录

1	小组名单	2
2	题目	2
3	练习	2
3.1	<i>CharConst</i> , P15	2
3.2	<i>Assign</i> , P21, P22	2
3.3	<i>P24</i>	2
3.4	<i>TestInit</i> , P39	2
3.5	<i>UnaryConversion</i> , P43	3
3.6	<i>P46</i>	3
3.7	<i>Equivalence</i> , P48	3
3.8	<i>P56</i>	3
3.9	<i>P58</i>	3
3.10	<i>P62</i>	3
4	题目分析	4
5	程序实现	4
5.1	求给定日期的星期	5
5.2	打印某月日历	6
5.3	人机交互	6
6	执行结果	7
6.1	主界面与错误提示	7
6.2	各命令演示	8
7	个人总结	8
A	各命令演示	9

## 1 小组名单

学号	姓名	工作
19030500217	陈德创	完成程序、练习例程、调试

## 2 题目

- 1) 执行 PPT “JAVA 语言基础” 中练习 (页码:15, 21, 22, 24, 39, 43, 46, 48, 56, 58, 62, 63)
- 2) 编写一个日历程序, 要求程序具有良好的人机交互性能并完成:
  - i. 根据用户输入的年份输出该年日历
  - ii. 根据用户输入的日期输出该天星期

## 3 练习

### 3.1 CharConst, P15

输出为 Q;Q;Q;Q, 分别用了字符, 转义 Unicode 码, Unicode 码, ACSII 码来表示'Q' 这一字符常量。

### 3.2 Assign, P21, P22

正常变量声明, 没什么好说的.....

不过需要注意的是在 **JAVA** 中布尔类型用 `boolean` 而不是 `bool`, 字符串为 `String`, 首字母大写。字符串的初始化方式是多样的, 但是需要注意 `String str = "Hello World";` 和 `String str = newString("Hello World");` 的区别, 前者会检索字符池, 如果存在 `Hello World`, 则会直接返回引用而不创建一个新的字符常量。

### 3.3 P24

**JAVA** 为了可以使基本类型享受类的特性, 将基本类型封装成了类的形式, `Double` 类便是 `double` 的封装。里面提供了常量 `NaN` 和 `NEGATIVE_INFINITY` 以及 `POSITIVE_INFINITY` 用于表示 *Not a Number*, 比如 `0.0/0.0`, 和负正无穷。`NaN` 不等于任何数, 也不大于或者小于任何数。判断一个结果是否为 `NaN` 需要用到方法 `isNaN()` 或者 `== Double.NaN` (我所用的 **JDK13** 是支持这样做的)。而正无穷比任何数大, 负无穷比任何数小。判断一个数是否为无穷在 `Double` 类中也有相应的方法, `isInfinity()`, 同样也可以直接利用 `==` 判断。

### 3.4 TestInit, P39

这个没什么好说的, 需要注意的是 **JAVA** 的变量在使用前必须进行初始化, 否则会报错 (也可能是 **Idea** 报的错, 但是无论如何新建一个变量顺手初始化是一个良好的习惯)。`Math.random()` 会返回一个 `0-1` 之间的 `double` 数。

### 3.5 *UnaryConversion*, P43

关于强制类型转换，低精度可以直接赋给高精度，高精度若要赋给低精度需要强制类型转换。比如 `int` 精度小的整数类型进行运算时会一致转换为 `int`，所以 `byte b = 0; b = b + 1;` 是错误的，因为在运算中 `b` 已经被转化为了 `int`，而高精度赋值给低精度需要强制类型转换。同样的，浮点运算会一直转换为 `double`，所以 `float f = 1; f = f + 1.0;` 也是错的。

所以我建议在具体写程序的时候如果不是对内存有着过分的要求，一致使用 `int` 和 `double` 以免产生不必要的错误。事实上有时候为了所谓的省内存甚至省不了内存而会拖慢程序运行速度。(我记得之前看过一篇文章讲到内存有着“对齐”，具体我也记不太清了，以后看到再说吧。)

### 3.6 *P46*

`emmmm` 和 `C` 相仿，在此不提了。

### 3.7 *Equivalence*, P48

这里主要是说 `==` 和 `equals()` 方法，前者直接比较引用是否是一个，而不关心具体的值，后者比较具体的值而并不关心是否是同一引用。这里要说的是 `Integer(intvalue)`；这个构造方法已经废弃 (**JDK13**)，现在直接用正数赋值就可以了。

### 3.8 *P56*

这里没什么说的，**JAVA** 中浮点数转换为整型采取的是舍小数。我个人认为这非常好，四舍五入可能引起不可意料的错误。**JAVA** 中布尔型和整形不能转化，`true` 就是 `true`，不是 `1` 也不失其他什么整数。这也是不错的，起码免除了一系列诸如 `if(x = 1)` 之类产生的 **bug**。

### 3.9 *P58*

**JAVA** 不允许 `x > y > 0` 类似的连续不等，这也与 `boolean` 和 `int` 不互通有关，因为慢慢算过来 `x > y` 是 `boolean` 型，不能与 `int` 比较。

### 3.10 *P62*

浮点数不要用 `==` 来作为结束条件，很好理解，因为浮点数并不总是精确的。如图：

```
double item=1, sum=0; item: 5.551115123125783E-17 sum: 3.0000000000000004
while(item!=0.0){ item: 5.551115123125783E-17
    sum+=item;
    item-=0.2;
```

如果将判断条件改为 `!(item > -0.00001 && item < 0.00001)`，即可达到其效果。

## 4 题目分析

题目要求实现一个日历程序，有两个基本功能，一个是给定年份输出日历，一个是给定日期确定星期。

对于问题 **i** 我们先考虑要求输出某一个月份的日历，我们只需确定该月 **1** 号是星期几然后依次输出到月份最后一天，每输出完星期六的日期换行即可。这里要注意平闰年 **2** 月的具体日期问题。

再考虑输出一年的日历，其实就是依次输出这一年每个月份的日历，调用十二次输出某一月份即可。

于是问题焦点在于如何确定某一天的星期，即第二个问题。（就在我写这篇报告的时候，突然在群里发现有相应的计算公式..... 我太菜了 ORZ）

对于问题 **ii** 我们这样考虑，如果知道了某一天（即基准天（*BASEDAY*））的星期，然后知道了所求天与基准天相差的天数（记为 *delta*），即可通过简单的取模运算来获得所求天的星期。公式如下： $Week(BASEDAY + delta) = (Week(BASEDAY) + delta - 1) \% 7 + 1$ 。

于是问题就转化为了如何确定两天相差的天数。在这里我们设两天 *a, b*，其中 *a* : *ya - ma - da*；*b* : *yb - mb - db*，其中 *a* 先于 *b*。确定两天之差的难点在于每个月天数不同以及平闰年天数的差别。为了增加代码复用率以及简化逻辑（我个人不喜欢 *if*，因为逻辑会很繁琐而且程序会变得很长.....），我们先确定 *ya - 01 - 01* 和 *yb - 01 - 01* 之间相差的天数，设为 *dy*，然后再确定 *ya - 01 - 01* 和 *ya - ma - da* 之间相差的天数以及 *yb - 01 - 01* 和 *yb - mb - db* 之间相差的天数，记为 *disa* 和 *disb*，那么 *a, b* 相差的天数  $delta = dy + disb - disa$ 。

这样问题就很简单了，判断两年元旦之间相差的天数直接的乘过去就可以了，然后注意一下闰年。判断一年之内两天的差就更简单了，逐月累加然后加上日期就可以了。于是我们将问题分解至了几个小问题（在不考虑公式的情况下 **555555555555 TAT**）。

**命令的输入** 解决了内部逻辑的实现，现在考虑人机交互问题。这里可以分为两个小问题，一是程序如何识别命令，二是程序如何识别日期。所以为了程序简便，我们将所有的日期格式定义为 *year [mouth [day]]*，这样我们就可以通过命令 *command year [mouth [day]]* 来告知程序我们要做什么。这样做的好处就是我们可以每次读入一整行，然后通过 *split(" ")* 轻易地划分出命令部分和日期部分。这样我们定义一个 *mainLoop* 方法，在这里面每次循环读入一行，然后利用 *split* 和 *swith* 来解析输入，完成人机交互。

**人性化设置** 为了良好的人机交互，我们需要一个良好的并且可以随时显示的帮助文档，以告知用户可以做什么。为了增加容错率，我们可以适当放宽命令格式的要求，比如当我要查询某一月份的日历的时候，那么显然 *year* 和 *mouth* 是必要的，而 *day* 参数是不必要的，于是我们在命令解析的时候只需要解析到 *mouth*，而忽略其后面的内容。

**日期的合法性** 最后我们需要考虑用户输入的日期的合法性，并对不合法的日期予以提示（如 2020.13.14）。

## 5 程序实现

程序实现的重点仍然是在计算两个日期的差，这里我们定义一个 *Date* 类，它有如下的成员变量：*int year, month, day, weekday*；分别用来表示这个日期的年月日星期。有如下的方法：

```
1 boolean isLegal();           //判断日期是否合法
2 boolean isLean();           //判断日期是否为闰年
```

```

3  int distance();           //计算日期与该年元旦的差
4  int minus(Date x);       //计算两个日期之间的差 (this - x)
5  int getWeekday();        //计算并返回日期是星期几
6  //比较两个日期的先后,先于x则返回-1,后于则返回1,同一天则返回0
7  int compare(Date x);
8  int minusYearDate date); //计算两年元旦之间的差

```

### 5.1 求给定日期的星期

`getWeekday()` 会调用 `minus()` 方法,来获得与基准天的差。这里基准天我们取 2020.01.01, *Wednesday*。进一步得到该天的星期。如下:

```

1  int getWeekday(){
2      int delta = this.minus(BASEDAY) % 7 + 7; //+7防止负数
3      this.weekday = (BASEDAY.weekday - 1 + delta) % 7 + 1;
4      return this.weekday;
5  }

```

更具体地, `minus()` 方法的实现主要是先调用 `compare()` 方法,保证两个日期的先后顺序,避免负数运算,然后调用一个静态方法 `staticintminus(Datea,Dateb)` 来进行进一步运算,其中 *a* 先于 *b*。`staticminus()` 方法会分别调用 `minusYear()` 方法和 *a, b* 的 `distance()` 方法来进一步计算差值。两个 `minus()` 方法实现如下:

```

1  static private int minus(Date a, Date b){
2      int ans = a.minusYear(b);
3      ans -= a.distance();
4      ans += b.distance();
5      return ans;
6  }
7  int minus(Date x){
8      int por = this.compare(x); //por表示两天的先后顺序
9      if (por == 0) return 0;
10     if (por < 0) return - Date.minus(this, x);
11     return Date.minus(x, this);
12 }

```

具体的 `minusYear` 实现,即先按全部为平年计算,然后计算两年之间的闰年数。如下:

```

1  int minusYear(Date date){
2      int delta = this.year - date.year, ans = delta * 365;
3      ans += (delta / 400) * 97; //计算每400年多出来的闰年数
4      delta %= 400;
5      //计算每一百年多出来的闰年数
6      ans += (delta / 100) * 24;
7      if ((this.year % 400) + delta > 400 ||
8          (this.year % 400 == 0 && delta > 0)) ans ++;
9      delta %= 100;
10     //计算每四年多出来的闰年数

```

```

11     ans += (delta / 4);
12     if ((this.year % 100) + delta > 100 ||
13         (this.year % 100 == 0 && delta > 0)) ans --;
14     delta %= 4;
15     //计算剩下的几年是否恰好有一个闰年
16     if ((this.year % 4) + delta > 4 ||
17         (this.year % 4 == 0 && delta > 0)) ans ++;
18     return ans;
19 }

```

由此，我们就计算出了给定日期的星期，进而打印一个月的日历。

## 5.2 打印某月日历

我们定义一个 *MyCalendar* 类来控制程序的行为。如下：

```

1 public class MyCalendar{
2     //基准日期，用于确定给定日期的星期
3     private static final Date BASEDAY = new Date(2020, 1, 1, 3);
4     private static final String[] WEEK = new String[]{
5         "", "Monday", "Tuesday", "Wednesday", "Thursday",
6         "Friday", "Saturday", "Sunday"};
7     //每月的天数
8     private static final int[] MOUTH = new int[]{0, 31, 28, 31,
9         30, 31, 30, 31, 31, 30, 31, 30, 31};
10    private static void queryDay(Date date); //查询某天的星期
11    private static void queryMouth(Date date); //查询某月的日历
12    private static void queryYear(Date date); //查询某年的日历
13    private static void mainLoop(); //程序主循环
14 }

```

*queryDay* 直接调用 *date.getWeek()* 然后标准化输出即可，*queryMouth* 即按照先前所说的，判定好该月一日的星期然后一次输出就好了。*queryYear* 就调用十二次 *queryMouth*。有一点需要说的是为了共享常量我直接把 *Date* 定义为了 *MyCalendar* 的内部类，当然这完全没有必要。程序所有代码 *MyCalendar.java* 文件。

## 5.3 人机交互

通过控制台输入命令，命令格式在前面已经列举了。主要框架如下：

```

1 private static void mainLoop(){
2     ...
3     while (true) {
4         //+ " . . . ." to guarantee the array's length >= 4
5         preCommand = sc.nextLine().toUpperCase() + " . . . .";
6         command = preCommand.split(" ");
7         switch (command[0]) {...}
8     }

```

9 }

## 6 执行结果

### 6.1 主界面与错误提示

运行程序首先展示的是一块帮助文档，也就是我们的主界面，用于告知用户如何控制程序，如图所示：

```

-----|  HELP DOCUMENT  |-----
You can input the command below to query the date you want.
#####(case-insensitive)#####
date    : 'year month day'.(PLEASE NOTICE WHETHER YOUR INPUT IS LEGAL)
'Q' or 'QUIT'    : to quit.
'H' or 'HELP'    : to get this document show again.
'Y' or 'YEAR' + date    : to get the calender of the year you input.
'M' or 'MOUTH' + date   : to get the calender of the mouth you input.
'D' or 'DAY' + date     : to get the week of the day you input.
-----|          END          |-----
|

```

在这里我们一共提供了五条并不区分大小写的命令，如下：

- 1) 'Q','QUIT' : 退出程序
- 2) 'H','HELP' : 显示此文档
- 3) 'Y','YEAR' : 参数为至少精确到某年的日期，打印此年日历
- 4) 'M','MOUTH' : 参数为至少精确到某月的日期，打印此月日历
- 5) 'D','DAY' : 参数为至少精确到某日的日期，打印此天星期

在这里我们和用户约定了日期的格式，也就是以空格分隔的三个正整数，自左向右以此为年月日，如果不按照此日期格式输入或者输入的日期格式有误，程序将会给出不同的提示信息。下面几张图片给出了错误提示范例：

图 1： 命令错误

```

help ← 命令错误，应为'help'      提示命令未找到
-----|  COMMAND NOT FOUND!  |-----
You can input 'H' or 'HELP' to get help.

```

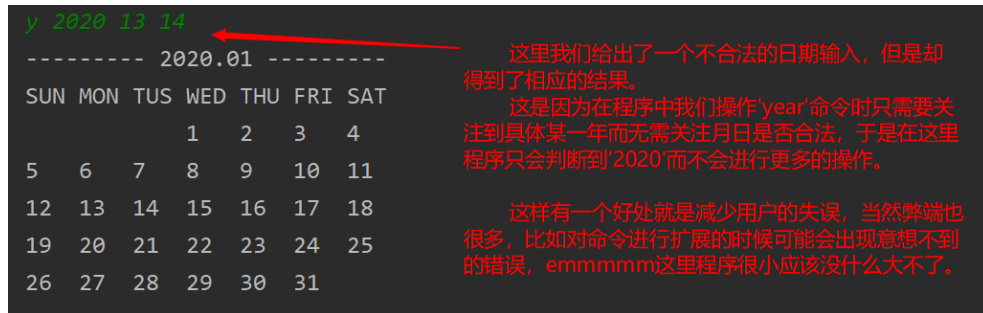
图 2： 日期格式错误

```


d 2020 13 14 日期不合法，程序给出日期格式错误
-----|  ILLEGAL INPUT DATE FORMAT!  |-----
You can input 'H' or 'HELP' to get help.

```

当然这里需要注意的是，我们程序只关注用到的部分，也就是说输出某年的日历只会关心年数是不是合法，而不关心月份和日份，如下图：



## 6.2 各命令演示

由于  图片排版出了点我无法理解的错误，所以图片就放到[附录](#)了.....

## 7 个人总结

报告写了很久终于写完了，程序五分钟，报告两小时.....

感觉写这个日历小程序的时候还是很有意思的，把一个大问题一步步分解成容易完成的小问题，然后把它们串联起来，一个程序就好了。

正如莎士比亚曾说：“简洁是智慧的灵魂，冗长是肤浅的藻饰。”我们应当在保证程序可行的前提下追寻简洁的解决方案。事实证明在动手之前进行适当的资料查询是很有用而且有好处的，如果花费十分钟提前找到公式，在实际编程中至少可以剩下半小时时间，而且逻辑清晰，安全可靠。我的程序冗长而且我只测试了百年以内的日期准确性，因为我没有在网络上找到更靠前或者更以后的日期数据。而且依靠个人测试，偶然性很大。

代码风格尤为重要。在写程序时我深深感受到了这一点，因为在初期的时候我对 `minus` 方法的前后顺序搞混了，以至于弄了很多意想不到的错误，大大减慢了开发效率。

百尺竿头，更进一步，愿进无止境。



A 各命令演示

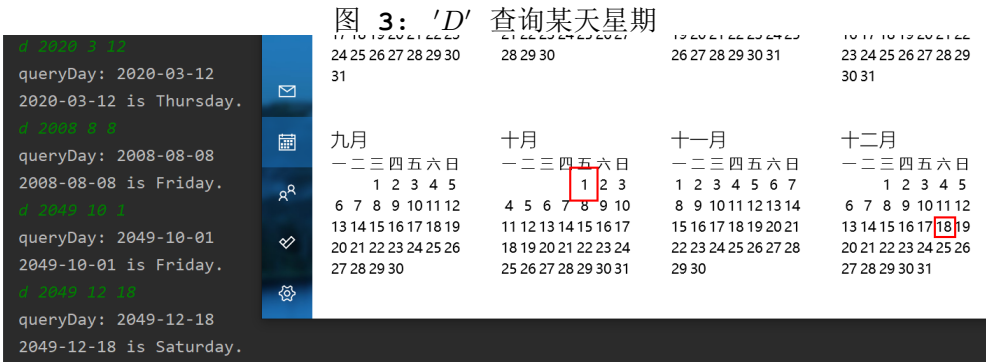




图 6: 'H'&'Q' 帮助和退出

