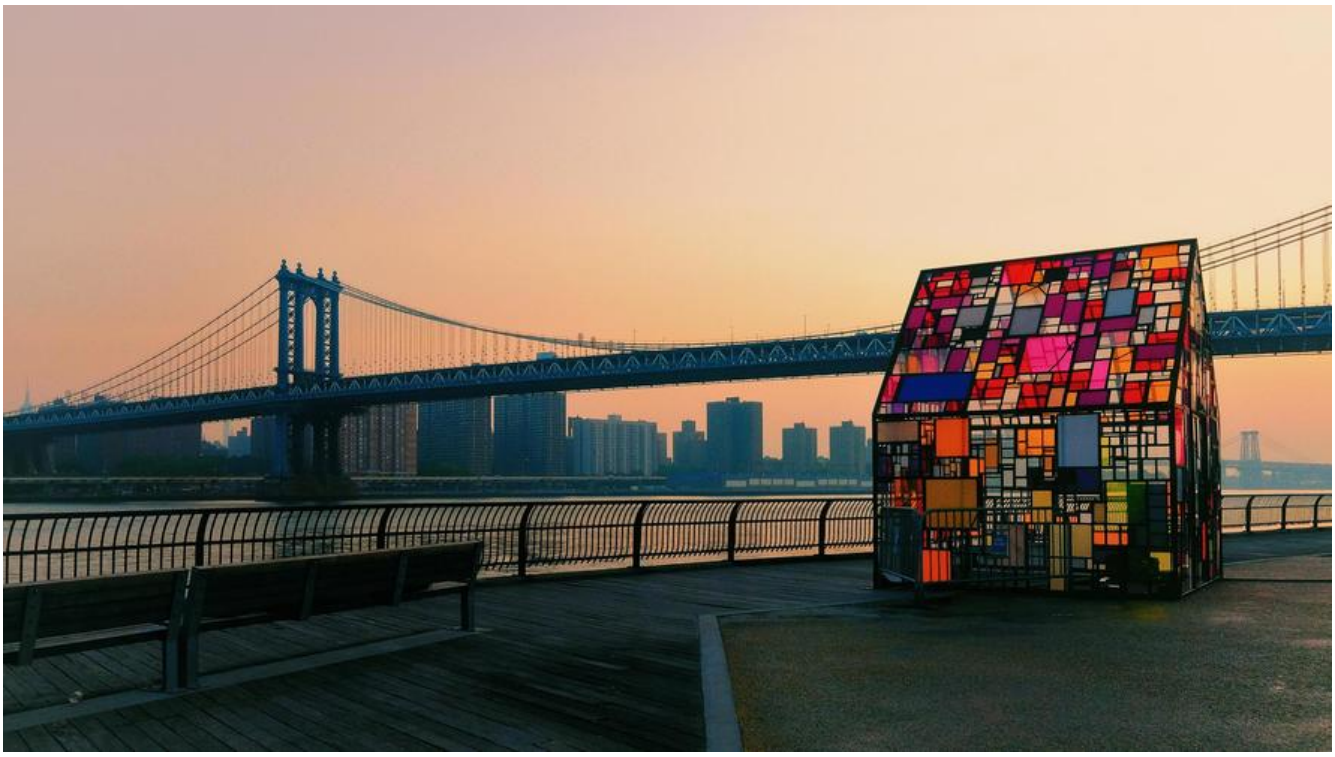
链滴

改善 Java 程序的 N 个建议（一）

作者：YYJeffrey

原文链接：https://ld246.com/article/1574951081137

来源网站：链滴



最近细读了秦小波老师的《编写高质量代码改善Jaav程序的151个建议》，要说是151个建议，其实合适的说是避免Java的一些冷门的坑，下面整理了N个比较有趣的建议重新学习了一遍。

建议3：三元操作符的类型务必一致

三元操作符运算也称为三目运算，其表达式形如："条件表达式？表达式1：表达式2"，在大部分语言都有这样的三元操作符，其目的就是为了简化if-else，当条件表达式为真时执行表达式1，否则执行达式2。

来分析一下下面这段代码：

```
public class Proposal_3 {
    public static void main(String[] args) {
        int i = 6;
        String str1 = String.valueOf(i < 10 ? 9 : 10);
        String str2 = String.valueOf(i < 10 ? 9 : 10.0);
        System.out.println("两者是否相等：" + str1.equals(str2));
    }
}
```

首先这段程序定义了i为6，那么它一定小于10，那么按道理，三目运算无论执行哪一个都会返回表达一，也就是9，之后再转为String，那str1和str2做equals之后肯定为true。真的是这样吗？通过运行现，得到结果：“两者是否相等：false”。

```
两者是否相等：false
9
9.0
```

出现了意想不到的结果，这也是Java中的一个容易踩坑的地方。事实上，三元操作符在表达式1和表式2的类型不一致时会发生转换，其转换规则如下：

- 若两个操作数不可转换，则不转换，返回Object

● 若两个操作数是明确类型的表达式（比如变量），则按照正常的二进制数字来转换，int类型转换为long类型，long类型转换为float类型

● 若两个操作数有一个是数字S，另一个是表达式，其类型为T，那么，若数字S在T的范围内，则转换T类型；若S超出了T类型的范围，则T转换为S类型

● 若操作数都是直接量数字，则返回值类型为范围较大者

根据这个几个规则就不难看出，第二个三元表达式的9是int，而10.0是float，则int会转换成float，后得到9.0，所以在最后比较时9和9.0两个字符串不相等。

建议20：不要只替换一个类

在开发中不乏会有很多常量接口（常量类），定义了开发时涉及的常量以简化代码，比如Struts2中的StrutsConstants就是一个常量类，定义了与Struts框架中与配置有关的常量，看一下下面这个常量累。

```
public class Constant {
    // 一年最多有12个月
    public final static int MAX_MONTH = 12;
}

public class Proposal_20 {
    public static void main(String[] args) {
        System.out.println("一年最多有" + Constant.MAX_MONTH + "个月");
    }
}
```

这是一个很简单的常量类，定义了一年最多有多少月份，引用这个常量类输出结果也很简单，就不再述，那么这个时候我们回到原始时代，开始用记事本写代码，用终端调试代码：

```
➜  Desktop javac Constant.java
➜  Desktop javac Proposal_20.java
➜  Desktop java Proposal_20
一年最多有12个月
```

Emmm，javac编译，java执行看到了结果，不错和预期一样，这个时候我们开启了流浪地球计划，类移民火星了，那我得把这个月份改改，火星一年有2061个月（未考证），然后修改代码如下：

```
public class Constant {
    // 一年最多有2061个月
    public final static int MAX_MONTH = 2061;
}
```

我们再来编译运行一次：

```
➜  Desktop javac Constant.java
➜  Desktop java Proposal_20
一年最多有12个月
```

What？还是12个月？移民失败了？好吧，其实这跟火星移民没有关系，原因就处在final关键字里，final修饰的基本类型和String类型编译器会认为是稳定态（Immutable Status），所以在编译之时，会接把值编译到字节码中，在上述代码中12这个数值是一个常量，而不是地址引用，所以无论后面怎么改，只要不重新编译主方法的类，其输出照旧。

不要小看这个坑位，在WEB开发中，有得时候只是对部分功能对部分类做了修改，此时，开发者要是

懒，直接采用替代部分编译好对class发布，这个时候就kennel会出现上述问题。

建议21：用偶判断，不用奇判断

判断一个数是偶数还是奇数，这个想必初学Java时都应该知道，只要判断其值是否能被2整除就行，能被2整除那么它就是奇数，能被2整除那它就是偶数，那话不多说，直接码代码：

```
public class Proposal_21 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入多个数字判断奇偶：");
        while (scanner.hasNextInt()) {
            int i = scanner.nextInt();
            String str = i + " --> " + (i % 2 == 1 ? "奇数" : "偶数");
            System.out.println(str);
        }
    }
}
```

输出结果如下图所示：

看了下结果，前几个都很正常，最后一个-7，居然输出都偶数，啥情况啊，Java这么差劲吗？别瞎下论，先来了解一下Java中都取余算法（%标示符），模拟代码如下：

```
// 取余运算，dividend是被除数，divisor是除数
public static int remainder(int dividend, int divisor) {
    return dividend - dividend / divisor * divisor;
}
```

上面的代码也很好理解，看了之后就明白了，原来Java取余和正常人的思维一样，先用被除数除以除然后得到的结果乘以除数，最后用被除数减去刚才的结果，但是Java居然没有考虑被除数为负数的情，当被除数为-7，除数为2时最终得到的结果是-1，没有产生预期的结果1，所以才导致了上述情况。

那么为了避免此类坑位，我们可以采用偶判断以求解，对于这些可有可无的知识点，我们要知其然，知其所以然。