目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

09 当switch遇到空指针

更新时间：2019-11-06 18:58:59

> 生活的理想，就是为了理想的生活。
>
> ——张闻天

1. 前言

《手册》的第 18 页有关于 switch 的规约：

> 【强制】当 switch 括号内的变量类型为 String 并且此变量为外部参数时，必须先进行 null
> 判断。1

在《手册》中，该规约下面还给出了一段反例（此处略）。

最近很火的一篇名为《悬赏征集！5 道题征集代码界前 3% 的超级王者》2 的文章，也给出了类似的一段代码：

```
public class SwitchTest {
    public static void main(String[] args) {
        String param = null;
        switch (param) {
            case "null":
                System.out.println("null");
                break;
            default:
                System.out.println("default");
        }
    }
}
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

真实，您知道答案很容易，运行一下程序答案就出来了。

但是如果浅尝辄止，我们就丧失了一次难得的学习机会，不像是一名优秀程序猿的作风。

我们还需要思考下面几个问题：

- switch 除了 String 还支持哪种类型？
- 为什么《手册》规定字符串类型参数要先进行 null 判断？
- 为什么可能会抛出异常？
- 该如何分析这类问题呢？

本节将对上述问题进行分析。

2. 问题分析

2.1 源码大法

按照我们一贯的风格，我们应该先上“源码大法”，但是 switch 是关键字，无法进入 JDK 源码中查看学习，因此我们暂时放弃通过源码或源码注释来分析解决的手段。

2.2 官方文档

我们去官方文档 JLS3 查看 swtich 语句相关描述。

> switch 的表达式必须是 char, byte, short, int, Character, Byte, Short, Integer, String, 或者 enum 类型，否则会发生编译错误
>
> switch 语句必须满足以下条件，否则会出现编译错误：
>
> - 与 switch 语句关联的每个 case 都必须和 switch 的表达式的类型一致；
> - 如果 switch 表达式是枚举类型，case 常量也必须是枚举类型；
> - 不允许同一个 switch 的两个 case 常量的值相同；
> - 和 switch 语句关联的常量不能为 null ；
> - 一个 switch 语句最多有一个 default 标签。

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

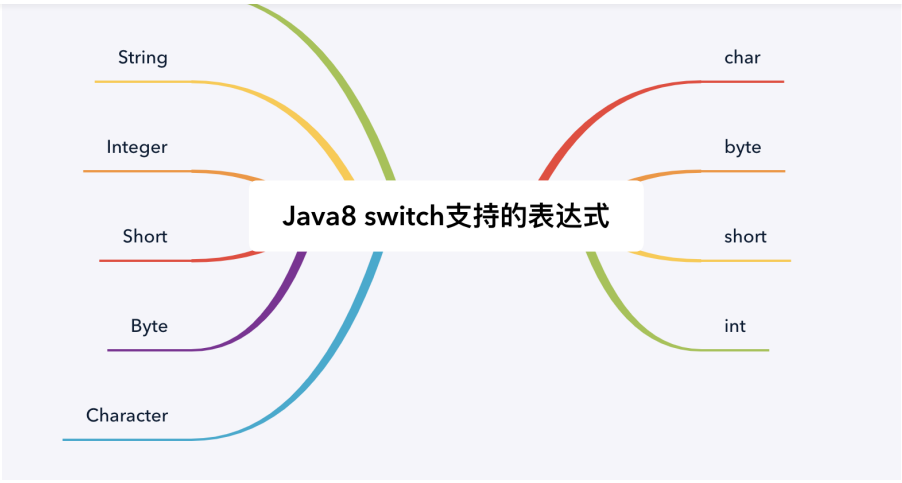
17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势



我们了解到 switch 语句支持的类型，以及会出现编译错误的原因。

我们看到关键的一句话：

> When the switch statement is executed, first the Expression is evaluated. If the Expression evaluates to null, a NullPointerException is thrown and the entire switch statement completes abruptly for that reason.
>
> switch 语句执行的时候，首先将执行 switch 的表达式。如果表达式为 null, 则会抛出 NullPointerException，整个 switch 语句的执行将被中断。

这里的表达式就是我们的参数，前言中该参数的值为 null，因此答案就显而易见了：结果会抛出异常，而且是前面章节讲到的 NullPointerException 。

另外从 JVM4 3.10 节 “Compiling Switches” ，我们学习到：

> 编译器使用 tableswitch 和 lookupswitch 指令生成 switch 语句的编译代码。tablesswtich 语句用于表示 swtich 结构的 case 语句块，它可以地从索引表中确定 case 语句块的分支偏移量。当 switch 语句的条件值不能对应索引表的任何一个 case 语句块的偏移量时就会用到 default 语句。
>
> Java 虚拟机的 tableswitch 和 lookupswitch 指令只能支持 int 类型的条件值。如果 swich 中使用其他类型的值，那么就必须转化为 int 类型。
>
> 当 switch 语句中的 case 分支条件比较稀疏时， tableswtich 指令的空间利用率较低。可以使用 lookupswitch 指令来取代。
>
> lookupswitch 指令的索引表项由 int 类型的键（来自于 case 语句后的数值）和对应目标语句的偏移量构成。当 lookcupswitch 指令执行时， switch 语句的条件值将和索引表中的键进行比对，如果某个键和条件的值相符，那么将转移到这个键对应的分支偏移量的代码行处开始执行，如果没有符合的键值，则执行 default 分支。

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

我们去 String 源码中寻找可以将字符串转 int 的函数，发现 hashCode() 可能是最佳的选择之一（后面会印证）。

因此空指针出现的根源在于：**虚拟机为了实现 switch 的语法，将参数表达式转换成 int。而这里的参数为 null， 从而造成了空指针异常。**

通过官方文档的阅读，我们对 switch 有了一个相对深入的了解。

2.3 Java 反汇编大法

如何印证官方文档的描述？如何进一步分析呢？

按照惯例我们用反汇编大法。

2.3.1 switch 举例

我们先看一个正常的示例：

```
public static void main(String[] args) {
        String param = "t";
        switch (param) {
            case "a":
                System.out.println("a");
                break;
            case "b":
                System.out.println("b");
                break;
            case "c":
                System.out.println("c");
                break;
            default:
                System.out.println("default");
        }
```

先进入到代码目录，对类文件进行编译：

```
javac SwitchTest2.java
```

然后反汇编的代码如下：

```
javap -c SwitchTest2
```

前方高能预警，先稳住，不要怕，不要方，后面会给出解释并给出简化版：

```
Compiled from "SwitchTest2.java"
public class com.imooc.basic.learn_switch.SwitchTest2 {
  public com.imooc.basic.learn_switch.SwitchTest2();
    Code:
      0: aload_0
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
      0: ldc           #2                  // String t
      2: astore_1
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

```
 6: istore_3
 7: aload_2
 8: invokevirtual #3                  // Method java/lang/String.hashCode:()I
11: tableswitch   { // 97 to 99
              97: 36
              98: 50
              99: 64
         default: 75
    }
36: aload_2
37: ldc           #4                  // String a
39: invokevirtual #5                  // Method java/lang/String.equals:(Ljava/lang/Object;)Z
42: ifeq          75
45: iconst_0
46: istore_3
47: goto          75
50: aload_2
51: ldc           #6                  // String b
53: invokevirtual #5                  // Method java/lang/String.equals:(Ljava/lang/Object;)Z
56: ifeq          75
59: iconst_1
60: istore_3
61: goto          75
64: aload_2
65: ldc           #7                  // String c
67: invokevirtual #5                  // Method java/lang/String.equals:(Ljava/lang/Object;)Z
70: ifeq          75
73: iconst_2
74: istore_3
75: iload_3
76: tableswitch   { // 0 to 2
               0: 104
               1: 115
               2: 126
         default: 137
    }
104: getstatic    #8                  // Field java/lang/System.out:Ljava/io/PrintStream;
107: ldc          #4                  // String a
109: invokevirtual #9                  // Method java/io/PrintStream.println:(Ljava/lang/String;)\
112: goto         145
115: getstatic    #8                  // Field java/lang/System.out:Ljava/io/PrintStream;
118: ldc          #6                  // String b
120: invokevirtual #9                  // Method java/io/PrintStream.println:(Ljava/lang/String;)\
123: goto         145
126: getstatic    #8                  // Field java/lang/System.out:Ljava/io/PrintStream;
129: ldc          #7                  // String c
131: invokevirtual #9                  // Method java/io/PrintStream.println:(Ljava/lang/String;)\
134: goto         145
137: getstatic    #8                  // Field java/lang/System.out:Ljava/io/PrintStream;
140: ldc          #10                 // String default
142: invokevirtual #9                  // Method java/io/PrintStream.println:(Ljava/lang/String;)\
145: return
}
```

首先介绍一个简单的背景知识：

> 字符 a 的 ASCII 码为 97, b 为 98，c 为 99（我们发现常见英文字母的哈希值为其 ASCII 码）。

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

我们讲解核心代码，先看偏移为 8 的指令，调用了参数的 hashCode() 函数来获取字符串 "t" 的哈希值。

```
tableswitch   { // 97 to 99
          97: 36
          98: 50
          99: 64
          default: 75
      }
```

接下来我们看偏移为 11 的指令处： tableswitch 是跳转引用列表， 如果值小于其中的最小值或者大于其中的最大值，跳转到 default 语句。

> 其中 97 为键，36 为对应的目标语句偏移量。

hashCode 和 tableswitch 的键相等，则跳转到对应的目标偏移量，t 的哈希值为 116，大于条件的最大值 99，因此跳转到 default 对应的语句行（即偏移量为 75 的指令处执行）。

从 36 到 74 行，根据哈希值相等跳转到判断是否相等的指令。

然后调用 java.lang.String#equals 判断 switch 的字符串是否和对应的 case 的字符串相等。

如果相等则分别根据第几个条件得到条件的索引，然后每个索引对应下一个指定的代码行数。

default 语句对应 137 行，打印 “default” 字符串，然后执行 145 行 return 命令返回。

然后再通过 tableswitch 判断执行哪一行打印语句。

因此整个流程是先计算字符串参数的哈希值，判断哈希值的范围，然后哈希值相等再判断对象是否相等，然后执行对应的代码块。

2.3.2 分析问题

经过前面的学习我们对 String 为参数的 switch 语句的执行流程有了初步认识。

我们反汇编开篇的示例，得到如下代码：

```
Compiled from "SwitchTest.java"
public class com.imooc.basic.learn_switch.SwitchTest {
  public com.imooc.basic.learn_switch.SwitchTest();
    Code:
      0: aload_0
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
      0: aconst_null
      1: astore_1
      2: aload_1
      3: astore_2
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

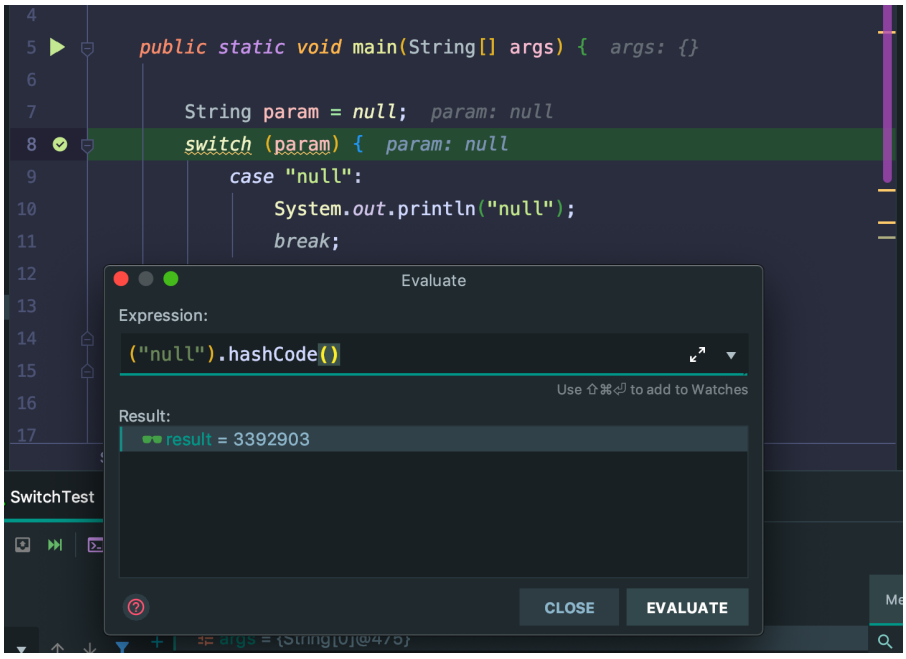
```
 7: invokevirtual #2                  // Method java/lang/String.hashCode:()I
10: lookupswitch  { // 1
          3392903: 28
          default: 39
        }
28: aload_2
29: ldc           #3                  // String null
31: invokevirtual #4                  // Method java/lang/String.equals:(Ljava/lang/Object;)Z
34: ifeq          39
37: iconst_0
38: istore_3
39: iload_3
40: lookupswitch  { // 1
                0: 60
          default: 71
        }
60: getstatic     #5                  // Field java/lang/System.out:Ljava/io/PrintStream;
63: ldc           #3                  // String null
65: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
68: goto          79
71: getstatic     #5                  // Field java/lang/System.out:Ljava/io/PrintStream;
74: ldc           #7                  // String default
76: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
79: return
}
```

猜想和验证是学习的最佳方式之一，我们通过猜想来提取知识，通过验证来核实自己的猜想是否正确。

猜想 1：根据上面的分析我们可以 “猜想”：3392903 应该是 "null" 字符串的哈希值。

我们可以打印其哈希值去印证：`System.out.println(("null").hashCode());` ，也可以通过编写单元测试来断言，还可以通过调试来执行表达式等方式查看。

在调试模式下，在变量选项卡上右键，选择 “Evaluate Expression…” ，填写想执行想计算的表达式即可：



目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

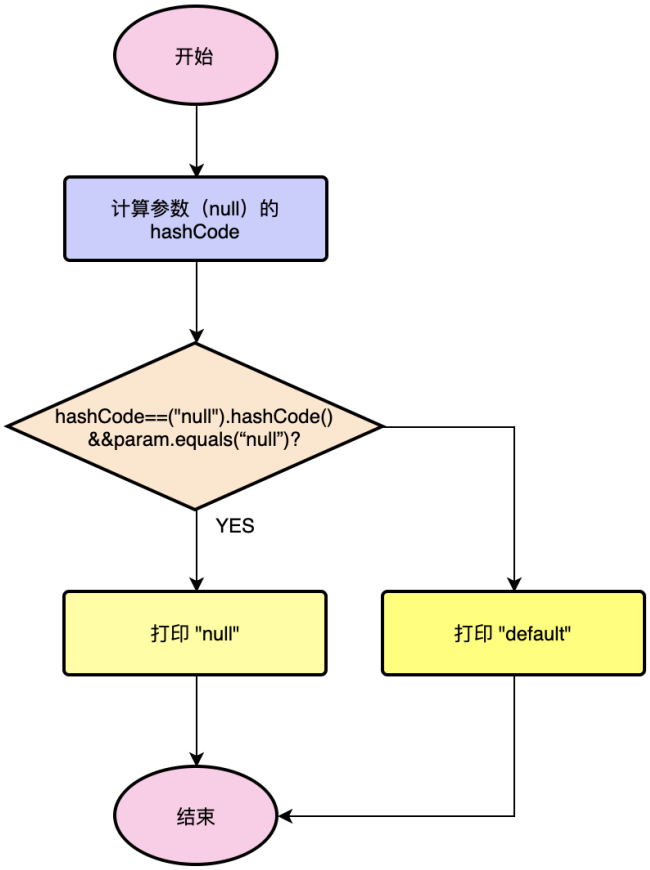
第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

```
String param = null;
int hashCode = param.hashCode();
if(hashCode == ("null").hashCode() && param.equals("null")){
   System.out.println("null");
}else{
   System.out.println("default");
}
```

对应流程图如下：



因此空指针的原因就一目了然了。

回忆一下空指针的小节讲到的：

> 空指针异常发生的原因之一：“调用 null 对象的实例方法。”。
>
> 以及 “JVM 也可能会通过 Throwable#Throwable(String, Throwable, boolean, boolean) 构造函数来构造 NullPointerException 对象。”

此处字节码执行时调用了 null 的 hashCode 方法，虚拟机可以通过上面的函数构造 NPE 并抛出。

那么将字符串通过 hashCode 函数转为整型和 case 条件对比后，为什么还需要 equals 再次判断呢？

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

hashCode 不等的对象一定不是同一个对象。

详情参见 java.lang.Object#hashCode 和 java.lang.Object#equals 的注释。

通过这一特性，可以快速判断对象是否有可能相当，避免不必要的比较。

另外我们还可以猜想如何提高比较的效率？

猜想 2： 如果编译期能够将 lookupswitch 按照 hash 值升序排序，则运行时就可讲参数的 hash 值（最小）先和第一个和除 default 外的倒数第一个 hash 值（最大）比较，不在这个范围直接走 default 语句即可，在这个范围就可以使用使用二分查找法，将时间复杂度降低到 O(logn) ，从而大大提高效率。

大家可以通过读 jvms 甚至读虚拟机代码去核实和验证上述猜想。

另外，**虽然有些哈希函数设计的比较优良，能够尽可能避免 hash 冲突，但是对象的数量是 “无限” 的，整数的范围是 “有限” 的，将无限的对象映射到有限的范围，必然会产生冲突。**

因此通过上述反汇编代码可以看出：

switch 表达式会先计算字符串的 hashCode （main 函数偏移为 7 处代码），然后根据 hashCode 是否相等快速判断是否要走到某个 case（见 lookupswith），如果不满足，直接执行到 default （main 函数偏移为 39 处代码）；如果满足，则跳转到对应 case 的代码（见 main 函数偏移为 28 之后的代码）再通过 equals 判断值是否相等，来避免 hash 冲突时 case 被误执行。

这种先判断 hash 值是否相等（有可能是同一个对象 / 两个对象有可能相等）再通过 equals 比较 “对象是否相等” 的做法，在 Java 的很多 JDK 源码中和其他框架中非常常见。

3. 总结

本节我们结合一个简单的案例 和 jvms， 学习了 switch 的基本原理，分析了示例代码产生空指针的原因。本节还介绍了一个简单的调试技巧，以及 “猜想和验证” 的学习方式，希望大家在后面的学习和工作中多加实践。

下一节我们将深入学习枚举并介绍其高级用法。

4. 课后题

下面的代码结果是啥呢？

```
public class SwitchTest {
    public static void main(String[] args) {
        String param = null;
        switch (param="null") {
            case "null":
                System.out.println("null");
                break;
            default:
                System.out.println("default");
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

大家可以通过今天学习的知识，自己去实战分析这个问题。

参考资料

1. 阿里巴巴与 Java 社区开发者.《 Java 开发手册 1.5.0》华山版. 2019.18 ↩

2. 悬赏征集！5 道题征集代码界前 3% 的超级王者 ↩

3. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley.《 Java Language Specification: Java SE 8 Edition》. 2015 ↩

4. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley. 《 Java Language Specification: Java SE 8 Edition》. 2015 ↩

← 08 空指针引发的血案　　10 枚举类的正确学习方式 →

精选留言 4

欢迎在这里发表留言，作者筛选后可公开显示

letro

有点迷，看了反编译的代码，param参数压根没有被用到，switch中的赋值语句被拎到前面了，而且还多了个临时变量var2= “null”，后续实际上是使用的var2，跟param没有关系了。求解惑

0 回复 2019-12-08

明明如月 回复 **letro**

你是一个比较爱思考的同学，这点非常棒，好奇心是学习的巨大驱动力。 之所以会有这种疑问，我认为主要是因为 对反编译的概念理解不太深刻，其次对现象和本质的认识不够深刻。 写了一个手记解答你的问题，希望对你有帮助。 https://www.imooc.com/article/297415

回复 7天前

letro 回复 **letro**

感谢解答，不过只解决了我一半的疑惑，因为在字节码中也是将null执行了两次压栈赋值，为什么会出现这种情况呢？不是增加了变量吗？还有操作指令的数量

回复 7天前

明明如月 回复 **letro**

这个在群里已经解答。 不要用源码的思维来理解字节码。 符串 null 复制一份并再次入栈 是为了store 存到局部变量表 这种重复是必然的，因为要保存到两个局部变量中，而且每次store就会出栈，所以必须dup一次，否则后面还得加载 null 字符串再store 相当于把: ldc #2 astore_1 ldc #2 astore_2 改为了 ldc #2 dup astore_1 astore_2 效果相同，没有新增变量，操作指令也没增加,而且从时间和空间局部性的角度考虑，这种效率应该更高。 就像你每次都查数据库，既然两次确定是一样的就没必要查了 直接拿上次查的clone一下就完了 就是这种感觉

回复 4天前

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 已学完

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针 最近阅读

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗?

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题?

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

后可不就是执行了case“null”了吧

0 回复 2019-12-02

铭心_

输出null
switch后面的括号里对param进行了赋值为字符串“null”，所以在case(“null”)满足条件，输出null

0 回复 2019-11-19

慕粉3543028

题目的答案 是 控制台输出 null

0 回复 2019-11-07

明明如月 回复 **慕粉3543028**
那么为什么呢？能否给出稍微详细的说明呢?
回复 2019-11-08 11:11:56

唯月_潇洒 回复 **慕粉3543028**
param赋值"null"
回复 2019-12-05 22:51:54

Seed2009 回复 **明明如月**
lookupswitch前边多了一步astore给param赋值"null"是吗？看不太明白，只能猜一下了:)
回复 4天前

点击展开后面 1 条

千学不如一看，千看不如一练