我们用 Kotlin 的时间已经比较长了，经历了比较多的版本，我们在 1.2 升级到 1.3 的过程中发现了一个 1.2.x 版本在处理 when 语法的一个低级 bug

```
enum class Color {
    Red,
    Blue
}
enum class FakeColor {
    Red,
    Blue
}
fun main(args: Array<String>) {
    val color = Color.Blue
    when (color) {
        FakeColor.Blue -> {
            println("Fake blue") // 1.2 版本调用这
里
        }
        FakeColor.Red -> {
            println("Fake red")
        }
        else -> {
            println("else")     // 1.3 版本调用这里
        }
    }
}
```

从代码上，我们应该可以直观地看出，肯定是进入 else 是吧，但是 1.2 的 Kotlin 居然进入了FakeColor.Blue分支
我们从字节码角度来分析一下问题的原因

0x01 使用 1.2 版本的 kotlinc 来编译上面的代码

部分字节码如下：

```
6: getstatic     #21                 // Field
Color.Blue:LColor;
9: astore_1
10: aload_1
11: getstatic     #27                  // Field
$WhenMappings.$EnumSwitchMapping$0:[I
14: swap
15: invokevirtual #31                  // Method
Color.ordinal:()I
18: iaload

19: tableswitch   { // 1 to 2
               1: 40 // 处理 Red 逻辑
               2: 53 // 处理 Blue 逻辑
         default: 66
  }
```

编译过程会生成一个新的类 $WhenMappings.class，这个类的文件我人肉翻译了一下，如下所示

```
public static class $WhenMappings {
    static int[] $EnumSwitchMapping$0;

    static {
        $EnumSwitchMapping$0 = new
int[Color.values().length];
        $EnumSwitchMapping$0[Color.Red.ordinal()]
= 1;

$EnumSwitchMapping$0[Color.Blue.ordinal()] = 2;
    }
}
```

这个类的作用就是做一下简单的映射。逐行介绍一下字节码的

- 6 ~ 18 行：获取 Color.blue 映射的 case 值 2
- 19 行：使用 tableswitch 进行分支跳转

到这里 bug 就出现了，最终执行了 println("Fake blue")
整段字节码中都没有出现 FakeColor 相关的东西！字节码翻译成 Java 代码如下

```
public void bar() {
    Color color = Color.Blue;
    int target =
$WhenMappings.$EnumSwitchMapping$0[color.ordinal(
)];
    switch (target) {
        case 1:
            System.out.println("Fake blue");
            break;
        case 2:
            System.out.println("Fake red");
            break;
        default:
            System.out.println("else");
            break;
    }
}
```

0x02 当两个枚举的名字不一样会如何

我们把 FakeColor 枚举项的名字修改一下，Color 枚举类不变

```
enum class Color {
    Red,
    Blue
}
enum class FakeColor {
    Red2,
    Blue2
}
fun main(args: Array<String>) {
    val color = Color.Blue
    when (color) {
        FakeColor.Blue2 -> {
            println("Fake blue")
        }
        FakeColor.Red2-> {
            println("Fake red")
        }
        else -> {
            println("else")
        }
    }
}
```

重新运行一下，发现这个时候，居然直接抛异常了

```
Exception in thread "main"
java.lang.NoSuchFieldError: Blue2
        at $WhenMappings.<clinit>(Unknown Source)
```

我们可以看此时 $WhenMappings.class 的字节码，翻译一下

```
public static class $WhenMappings {
    static int[] $EnumSwitchMapping$0;

    static {
        $EnumSwitchMapping$0 = new
int[Color.values().length];

$EnumSwitchMapping$0[Color.Red2.ordinal()] = 1;

$EnumSwitchMapping$0[Color.Blue2.ordinal()] = 2;
    }
}
```

因为 Color 枚举类并没有 Red2 和 Blue2 枚举项，所以这有运行时异常

0x03 使用 1.3 版本的 kotlinc 来编译上面的代码

部分字节码如下

```
6: getstatic     #21                 // Field
Color.Blue:LColor;
9: astore_1
10: aload_1
11: astore_2
12: aload_2
13: getstatic     #26                 // Field
FakeColor.Blue:LFakeColor;
16: if_acmpne     32
```

可以看到 1.3 中对 when 的选项类型不一致时，是直接使用对象相等比较的方式，而且 kotlinc 编译的时候会提示 warning

```
warning: comparison of incompatible enums 'Color'
and 'FakeColor' is always unsuccessful
        FakeColor.Blue -> {
        ^
```

翻译成 Java 代码如下

```
public void testColor() {
    Color color = Color.Blue;
    if (color.equals(FakeColor.Blue)) {
        System.out.println("Fake blue");
    } else if (color.equals(FakeColor.Red)) {
        System.out.println("Fake red");
    } else {
        System.out.println("else");
    }
}
```

这个问题曾经造成了我们线上的紧急 bug 修复，本身是我们代码写得不对。通过对字节码深入的分析，就迅速找到了问题的原因并进行了修复。还在用 Kotlin 1.2 的同学趁早升级到 1.3 及以上吧。

0x04 小结

这篇文章从项目的一个实际例子出发分析了一个编译器级别的 bug 应该如何来分析和定位。希望对你有所启发。