二

73 为什么加了 final 却依然无法拥有“不变性”？

本课时我们主要讲解为什么加了 final 却依然无法拥有“不变性”。

什么是不变性

要想回答上面的问题，我们首先得知道什么是不变性（Immutable）。**如果对象在被创建之后，其状态就不能修改了，那么它就具备“不变性”。**

我们举个例子，比如下面这个 Person 类：

```
public class Person {
    final int id = 1;
    final int age = 18;
}
```

如果我们创建一个 person 对象，那么里面的属性会有两个，即 id 和 age，并且由于它们都是被 final 修饰的，所以一旦这个 person 对象被创建好，那么它里面所有的属性，即 id 和 age 就都是不能变的。我们如果想改变其中属性的值就会报错，代码如下所示：

```
public class Person {
    final int id = 1;
    final int age = 18;
    public static void main(String[] args) {
        Person person = new Person();
//        person.age=5;//编译错误，无法修改 final 变量的值
    }
}
```

比如我们尝试去改变这个 person 对象，例如将 age 改成 5，则会编译通不过，所以像这样的 person 对象就具备不变性，也就意味着它的状态是不能改变的。

final 修饰对象时，只是引用不可变

这里有个非常重要的注意点，那就是当我们用 final 去修饰一个指向**对象**类型（而不是指向 8 种基本数据类型，例如 int 等）的变量时候，那么 final 起到的作用**只是保证这个变量的引用不可变，而对象本身的内容依然是可以变化的。**下面我们对此展开讲解。

在上一课时中我们讲过，被 final 修饰的变量意味着一旦被赋值就不能修改，也就是只能被赋值一次，如果我们尝试对已经被 final 修饰过的变量再次赋值的话，则会报编译错误。我们用下面的代码来说明：

```
/**
 * 描述：      final变量一旦被赋值就不能被修改
 */
public class FinalVarCantChange {
    private final int finalVar = 0;
    private final Random random = new Random();
    private final int array[] = {1,2,3};
    public static void main(String[] args) {
        FinalVarCantChange finalVarCantChange = new FinalVarCantChange();
//        finalVarCantChange.finalVar=9;     //编译错误，不允许修改final的变量(基本类型
//        finalVarCantChange.random=null;    //编译错误，不允许修改final的变量(对象)
//        finalVarCantChange.array = new int[5];//编译错误，不允许修改final的变量（数组
    }
}
```

我们首先在这里分别创建了一个 int 类型的变量、一个 Random 类型的变量，还有一个是数组，它们都是被 final 修饰的；然后尝试对它们进行修改，比如把 int 变量的值改成 9，或者把 random 变量置为 null，或者给数组重新指定一个内容，这些代码都无法通过编译。

这就证明了“被 final 修饰的变量意味着一旦被赋值就不能修改”，而这个规则对于基本类型的变量是没有歧义的，但是对于对象类型而言，final 其实只是保证这个变量的引用不可

变，而对象本身依然是可以变化的。这一点同样适用于数组，因为**在 Java 中数组也是对象**。那我们就来举个例子，看一看以下 Java 程序的输出：

```
class Test {

    public static void main(String args[]) {

       final int arr[] = {1, 2, 3, 4, 5};  //  注意，数组 arr 是 final 的

       for (int i = 0; i < arr.length; i++) {

           arr[i] = arr[i]*10;

           System.out.println(arr[i]);

       }

    }

}
```

首先来猜测一下，假设不看下面的输出结果，只看这段代码，你猜它打印出什么样的结果？这段代码中有个 Test 类，而且这个类只有一个 main 方法，方法里面有一个 final 修饰的 arr 数组。注意，数组是对象的一种，现在数组是被 final 修饰的，所以它的意思是一旦被赋值之后，变量的引用不能修改。但是我们现在想证明的是，数组对象里面的内容可以修改，所以接下来我们就用 for 循环把它里面的内容都乘以 10，最后打印出来结果如下：

```
10

20

30

40

50
```

可以看到，它打印出来的是 10 20 30 40 50，而不是最开始的 1 2 3 4 5，这就证明了，虽然数组 arr 被 final 修饰了，它的引用不能被修改，但是里面的内容依然是可以被修改的。

同样，对于非数组的对象而言也是如此，我们来看下面的例子：

```
class Test {

    int p = 20;

    public static void main(String args[]){
```

```
        final Test t = new Test();
        t.p = 30;
        System.out.println(t.p);
    }
}
```

这个 Test 类中有一个 int 类型的 p 属性，我们在 main 函数中新建了 Test 的实例 t 之后，把它用 final 修饰，然后去尝试改它里面成员变量 p 的值，并打印出结果，程序会打印出“30”。一开始 p 的值是 20，但是最后修改完毕变成了 30，说明这次修改是成功的。

以上我们就得出了一个结论，**final 修饰一个指向对象的变量的时候，对象本身的内容依然是可以变化的。**

final 和不可变的关系

这里就引申出一个问题，那就是 final 和不变性究竟是什么关系？

那我们就来具体对比一下 final 和不变性。关键字 final 可以确保变量的引用保持不变，但是不变性意味着对象一旦创建完毕就不能改变其状态，**它强调的是对象内容本身，而不是引用**，所以 final 和不变性这两者是很不一样的。

对于一个类的对象而言，你必须要保证它创建之后所有内部状态（包括它的成员变量的内部属性等）永远不变，才是具有不变性的，这就要求所有成员变量的状态都不允许发生变化。

有一种说法就认为：“要想保证对象具有不变性的最简单的办法，就是把类中所有属性都声明为 final”，这条规则是不完全正确的，它通常只适用于类的所有属性都是基本类型的情况，比如前面的例子：

```
public class Person {
    final int id = 1;
    final int age = 18;
}
```

Person 类里面有 final int id 和 final int age 两个属性，都是基本类型的，且都加了 final，所以 Person 类的对象确实是具备不变性的。

但是如果一个类里面有一个 final 修饰的成员变量，并且这个成员变量不是基本类型，而是

对象类型，那么情况就不一样了。有了前面基础之后，我们知道，对于对象类型的属性而言，我们如果给它加了 final，它内部的成员变量还是可以变化的，因为 final 只能保证其引用不变，不能保证其内容不变。所以这个时候**若一旦某个对象类型的内容发生了变化，就意味着这整个类都不具备不变性了。**

所以我们就得出了这个结论：**不变性并不意味着，简单地使用 final 修饰所有类的属性，这个类的对象就具备不变性了。**

那就会有一个很大的疑问，假设我的类里面有一个**对象类型的成员变量**，那要怎样做才能保证整个对象是不可变的呢？

我们来举个例子，即**一个包含对象类型的成员变量的类的对象，具备不可变性的例子。**

代码如下：

```
public class ImmutableDemo {

    private final Set<String> lessons = new HashSet<>();

    public ImmutableDemo() {

        lessons.add("第01讲：为何说只有 1 种实现线程的方法？");

        lessons.add("第02讲：如何正确停止线程？为什么 volatile 标记位的停止方法是错误的？

        lessons.add("第03讲：线程是如何在 6 种状态之间转换的？");

    }

    public boolean isLesson(String name) {

        return lessons.contains(name);

    }

}
```

在这个类中有一个 final 修饰的、且也是 private 修饰的的一个 Set 对象，叫作 lessons，它是个 HashSet；然后我们在构造函数中往这个 HashSet 里面加了三个值，分别是第 01、02、03 讲的题目；类中还有一个方法，即 isLesson，去判断传入的参数是不是属于本课前 3 讲的标题，isLesson 方法就是利用 lessons.contains 方法去判断的，如果包含就返回 true，否则返回 false。这个类的内容就是这些了，没有其他额外的代码了。

在这种情况下，尽管 lessons 是 Set 类型的，尽管它是一个对象，但是对于 ImmutableDemo 类的对象而言，就是具备不变性的。因为 lessons 对象是 final 且 private 的，所以引用不会变，且外部也无法访问它，而且 ImmutableDemo 类也没有任何方法可以

去修改 lessons 里包含的内容，只是在构造函数中对 lessons 添加了初始值，所以 ImmutableDemo 对象一旦创建完成，也就是一旦执行完构造方法，后面就再没有任何机会可以修改 lessons 里面的数据了。而对于 ImmutableDemo 类而言，**它就只有这么一个成员变量，而这个成员变量一旦构造完毕之后又不能变**，所以就使得这个 ImmutableDemo 类的对象是具备不变性的，这就是一个很好的“**包含对象类型的成员变量的类的对象，具备不可变性**”的例子。

总结

下面进行总结，在本课时，我们首先介绍了什么是不变性，然后介绍了用 final 修饰一个对象类型的变量的时候，只能保证它的引用不变，但是对象内容自身依然是可以变的。

之后，我们探讨了关键字 final 和不变性的关系。我们知道仅仅把所有的成员变量都用 final 修饰并不能代表类的对象就是具备不变性的。

上一页 下一页