Java 有值类型吗?

有人看了我之前的文章『Swift 语言的设计错误』,问我:"你说 Java 只有引用类型(reference type),但是根据 Java 的官方文档,Java 也有值类型(value type)和引用类型的区别的。比如 int,boolean 等原始类型就是值类型。"现在我来解释一下这个问题。

Java 有值类型,原始类型 int, boolean 等是值类型,其实是长久以来的一种误解,它混淆了实现和语义的区别。不要以为 Java 的官方文档那样写就是权威定论,就可以说"王垠不懂":) 当你认为王垠不懂一个初级问题的时候,都需要三思,因为他可能是大智若愚……看了我下面的论述,也许你会发现自己应该怀疑的是,Java 的设计者到底有没有搞明白这个问题:P

胡扯结束,现在来说正事。Java,Scheme 等语言的原始类型,比如 char,int,boolean,double 等,在"实现"上确实是通过值(而不是引用,或者叫指针)直接传递的,然而这完全是一种为了效率的优化(叫做 inlining)。这种优化对于程序员应该是不可见的。Java 继承了 Scheme/Lisp 的衣钵,它们在"语义"上其实是没有值类型的。

这不是天方夜谭,为了理解这一点,你可以做一个很有意思的思维实验。现在你把 Java 里面所有的原始类型都"想象"成引用类型,也就是说,所有的 int, boolean 等原始类型的变量都不包含实际的数据,而是引用(或者叫指针),指向堆上分配的数据。然后你会发现这样"改造后"的 Java,仍然符合现有 Java 代码里能看到的一切现象。也就是说,原始类型被作为值类型还是引用类型,对于程序员完全没有区别。

举个简单的例子,如果我们把 int 的实现变成完全的引用,然后来看这段代码:

int x = 1; // x指向内存地址A, 内容是整数1

int y = x; // y指向同样的内存地址A,内容是整数1

x = 2; // x指向另一个内存地址B,内容是整数2。y仍然指向地址A,内容是1。

由于我们改造后的 Java 里面 int 全部是引用,所以第一行定义的 x 并不包含一个整数,而是一个引用,它指向堆里分配的一块内存,这个空间的内容是整数 1。在第二行,我们定 int 变量 y,当然它也是一个引用,它的值跟 x 一样,所以 y 也指向同一个地址,里面的内容是同一个整数:1。在第三行,我们对 x 这个引用赋值。你会发现一个很有意思的现象,虽然 x 指向了 2,y 却仍然指向 1。对 x 赋值并没能改变 y 指向的内容,这种情况就跟 int 是值类型的时候一样一样!所以现在虽然 int 变量全部是引用,你却不能实现共享地址的引用能做的事情:对 x 进行某种操作,导致 y 指向的内容也发生改变。

出现这个现象的原因是,虽然现在 int 成了引用类型,你却并不能对它进行引用类型所特有(而值类型没有)的操作。这样的操作包括:

- 1. deref。就像 C 语言里的 * 操作符。
- 2. 成员赋值。就像对 C struct 成员的 x. foo = 2。

在 Java 里,你没法写像 C 语言的 *x = 2 这样的代码,因为 Java 没有提供 deref 操作符 *。你也没法通过 x.foo = 2 这样的语句改变 x 所指向的内存数据(内容是1)的一部分,因为 int 是一个原始类型。最后你发现,你只能写 x = 2,也就是改变 x 这个引用本身的指向。x = 2 执行之后,原来数字 1 所在的内存空间并没有变成 2,只不过 x 指向了新的地址,那里装着数字 2 而已。指向 1 的其它引用变量比如 y,不会因为你进行了 x = 2 这个操作而看到 2,它们仍然看到原来那个1……

在这种 int 是引用的 Java 里,你对 int 变量 x 能做的事情只有两种:

- 1. 读出它的值。
- 2. 对它进行赋值,使它指向另一个地方。

这两种事情,就跟你能对值类型能做的两件事情没有区别。这就是为什么你没法通过对 x 的操作而改变 y 表示的值。所以不管 int 在实现上是传递值还是传递引用,它们在语义上都是等价的。也就是说,原始类型是值类型还是引用类型,对于程序员来说完全没有区别。你完全可以把 Java 所有的原始类型都想成引用类型,之后你能对它们做的事情,你的编程思路和方式,都不会因此有任何的改变。

从这个角度来看,Java 在语义上是没有值类型的。值类型和引用类型如果同时并存,程序员必须能够在语义上感觉到它们的不同,然而不管原始类型是值类型还是引用类型,作为程序员,你无法感觉到任何的不同。所以你完全可以认为 Java 只有引用类型,把原始类型全都当成引用类型来用,虽然它们确实是用值实现的。

一个在语义上有值类型的语言(比如 C#,Go 和 Swift)必须具有以下两种特性之一(或者两者都有),程序员才能感觉到值类型的存在:

- 1. deref 操作。这使得你可以用 *x = 2 这样的语句来改变引用指向的内容,导致共享地址的其它引用看到新的值。你没法通过 x = 2 让其他值变量得到新的值,所以你感觉到值类型的存在。
- 2. 像 struct 这样的"值组合类型"。你可以通过 x.foo = 2 这样的成员赋值改变引用数据(比如 class object)的一部分,使得共享地址的其它引用看到新的值。你没法通过成员赋值让另一个 struct 变量得到新的值,所以你感觉到值类型的存在。

实际上,所有的数据都是引用类型就是 Scheme 和 Java 最初的设计原理。原始类型用值来传递数据只是一种性能优化(叫做 inlining),它对于程序员应该是透明(看不见)的。那些在面试时喜欢问"Java 是否所有数据都是引用",然后当你回答"是"的时候纠正你说"int,boolean 是值类型"的人,都是本本主义者。

思考题

有人指出,Java 的引用类型可以是 null,而原始类型不行,所以引用类型和值类型还是有区别的。但是其实这并不能否认本文指出的观点,你可以想想这是为什么吗?