但是 Kotlin 的类型推导机制并不总是可以正常工作的，比如说如果我们对一个变量延迟赋值的话，Kotlin 就无法自动推导它的类型了。这时候就需要显式地声明变量类型才行，Kotlin 提供了对这一功能的支持，语法如下所示：

```
val a: Int = 10
```

可以看到，我们显式地声明了变量 a 为 Int 类型，此时 Kotlin 就不会再尝试进行类型推导了。如果现在你尝试将一个字符串赋值给 a，那么编译器就会抛出类型不匹配的异常。

如果你学过 Java 并且足够细心的话，你可能发现了 Kotlin 中 Int 的首字母是大写的，而 Java 中 int 的首字母是小写的。不要小看这一个字母大小写的差距，这表示 Kotlin 完全抛弃了 Java 中的基本数据类型，全部使用了对象数据类型。在 Java 中 int 是关键字，而在 Kotlin 中 Int 变成了一个类，它拥有自己的方法和继承结构。表 2.1 中列出了 Java 中的每一个基本数据类型在 Kotlin 中对应的对象数据类型。

表 2.1　Java 和 Kotlin 数据类型对照表

Java 基本数据类型	Kotlin 对象数据类型	数据类型说明
int	Int	整型
long	Long	长整型
short	Short	短整型
float	Float	单精度浮点型
double	Double	双精度浮点型
boolean	Boolean	布尔型
char	Char	字符型
byte	Byte	字节型

接下来我们尝试对变量 a 进行一些数学运算，比如说让 a 变大 10 倍，可能你会很自然地写出如下代码：

```
fun main() {
    val a: Int = 10
    a = a * 10
    println("a = " + a)
}
```

很遗憾，如果你这样写的话，编译器一定会提示一个错误：Val cannot be reassigned。这是在告诉我们，使用 val 关键字声明的变量无法被重新赋值。出现这个问题的原因是我们在一开始定义 a 的时候将它赋值成了 10，然后又在下一行让它变大 10 倍，这个时候就是对 a 进行重新赋值了，因而编译器也就报错了。

解决这个问题的办法也很简单，前面已经提到了，val 关键字用来声明一个不可变的变量，而 var 关键字用来声明一个可变的变量，所以这里只需要把 val 改成 var 即可，如下所示：