Kotlin 基础语法 Kotlin 条件控制

Kotlin 基本数据类型

Kotlin 的基本数值类型包括 Byte、Short、Int、Long、Float、Double 等。不同于Java的是，字符不属于数值类型，是一个独立的数据类型。

类型	位宽度
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

字面常量

下面是所有类型的字面常量：

- 十进制：123
- 长整型以大写的 L 结尾：123L
- 16 进制以 0x 开头：0x0F
- 2 进制以 0b 开头：0b00001011
- 注意：8进制不支持

Kotlin 同时也支持传统符号表示的浮点数值：

- Doubles 默认写法: `123.5`, `123.5e10`
- Floats 使用 f 或者 F 后缀：`123.5f`

你可以使用下划线使数字常量更易读：

```
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
val bytes = 0b11010010_01101001_10010100_10010010
```

比较两个数字

Kotlin 中没有基础数据类型，只有封装的数字类型，你每定义的一个变量，其实 Kotlin 帮你封装了一个对象，这样可以保证不会出现空指针。数字类型也一样，所有在比较两个数字的时候，就有比较数据大小和比较两个对象是否相同的区别了。

在 Kotlin 中，三个等号 === 表示比较对象地址，两个 == 表示比较两个值大小。

```
fun main(args: Array<String>) {
    val a: Int = 10000
    println(a === a) // true，值相等，对象地址相等

    //经过了装箱，创建了两个不同的对象
    val boxedA: Int? = a
    val anotherBoxedA: Int? = a

    //虽然经过了装箱，但是值是相等的，都是10000
    println(boxedA === anotherBoxedA) //  false，值相等，对象地址不一样
    println(boxedA == anotherBoxedA) // true，值相等
}
```

类型转换

由于不同的表示方式，较小类型并不是较大类型的子类型，较小的类型不能隐式转换为较大的类型。 这意味着在不进行显式转换的情况下我们不能把 Byte 型值赋给一个 Int 变量。

```
val b: Byte = 1 // OK, 字面值是静态检测的
val i: Int = b // 错误
```

我们可以代用其toInt()方法。

```
val b: Byte = 1 // OK, 字面值是静态检测的
val i: Int = b.toInt() // OK
```

每种数据类型都有下面的这些方法，可以转化为其它的类型：

```
toByte(): Byte
toShort(): Short
toInt(): Int
toLong(): Long
toFloat(): Float
toDouble(): Double
toChar(): Char
```

有些情况下也是可以使用自动类型转化的，前提是可以根据上下文环境推断出正确的数据类型而且数学操作符会做相应的重载。例如下面是正确的：

```
val l = 1L + 3 // Long + Int => Long
```

位操作符

对于Int和Long类型，还有一系列的位操作符可以使用，分别是：

```
shl(bits) – 左移位 (Java's <<)
shr(bits) – 右移位 (Java's >>)
ushr(bits) – 无符号右移位 (Java's >>>)
and(bits) – 与
or(bits) – 或
xor(bits) – 异或
inv() – 反向
```

字符

和 Java 不一样，Kotlin 中的 Char 不能直接和数字操作，Char 必需是单引号 `'` 包含起来的。比如普通字符 '0'，'a'。

```
fun check(c: Char) {
    if (c == 1) { // 错误：类型不兼容
        // ……
    }
}
```

字符字面值用单引号括起来: '1'。 特殊字符可以用反斜杠转义。 支持这几个转义序列：\t、 \b、\n、\r、\'、\"、\\ 和 \$。 编码其他字符要用 Unicode 转义序列语法：'\uFF00'。

我们可以显式把字符转换为 Int 数字：

```
fun decimalDigitValue(c: Char): Int {
    if (c !in '0'..'9')
        throw IllegalArgumentException("Out of range")
    return c.toInt() - '0'.toInt() // 显式转换为数字
}
```

当需要可空引用时，像数字、字符会被装箱。装箱操作不会保留同一性。

布尔

布尔用 Boolean 类型表示，它有两个值：true 和 false。

若需要可空引用布尔会被装箱。

内置的布尔运算有：

```
|| – 短路逻辑或
&& – 短路逻辑与
! - 逻辑非
```

数组

数组用类 Array 实现，并且还有一个 size 属性及 get 和 set 方法，由于使用 [] 重载了 get 和 set 方法，所以我们可以通过下标很方便的获取或者设置数组对应位置的值。

数组的创建两种方式：一种是使用函数arrayOf()；另外一种是使用工厂函数。如下所示，我们分别是两种方式创建了两个数组：

```
fun main(args: Array<String>) {
    //[1,2,3]
    val a = arrayOf(1, 2, 3)
    //[0,2,4]
    val b = Array(3, { i -> (i * 2) })

    //读取数组内容
    println(a[0])    // 输出结果：1
    println(b[1])    // 输出结果：2
}
```

如上所述，[] 运算符代表调用成员函数 get() 和 set()。

注意: 与 Java 不同的是，Kotlin 中数组是不型变的（invariant）。

除了类Array，还有ByteArray, ShortArray, IntArray，用来表示各个类型的数组，省去了装箱操作，因此效率更高，其用法同Array一样：

```
val x: IntArray = intArrayOf(1, 2, 3)
x[0] = x[1] + x[2]
```

字符串

和 Java 一样，String 是不可变的。方括号 [] 语法可以很方便的获取字符串中的某个字符，也可以通过 for 循环来遍历：

```
for (c in str) {
    println(c)
}
```

Kotlin 支持三个引号 """ 扩起来的字符串，支持多行字符串，比如：

```
fun main(args: Array<String>) {
    val text = """
    多行字符串
    多行字符串
    """
    println(text)   // 输出有一些前置空格
}
```

String 可以通过 trimMargin() 方法来删除多余的空白。

```
fun main(args: Array<String>) {
    val text = """
    |多行字符串
    |菜鸟教程
    |多行字符串
    |Runoob
    """.trimMargin()
    println(text)    // 前置空格删除了
}
```

默认 | 用作边界前缀，但你可以选择其他字符并作为参数传入，比如 trimMargin(">")。

字符串模板

字符串可以包含模板表达式 ，即一些小段代码，会求值并把结果合并到字符串中。 模板表达式以美元符（$）开头，由一个简单的名字构成:

```
fun main(args: Array<String>) {
    val i = 10
    val s = "i = $i" // 求值结果为 "i = 10"
    println(s)
}
```

或者用花括号扩起来的任意表达式:

```
fun main(args: Array<String>) {
    val s = "runoob"
    val str = "$s.length is ${s.length}" // 求值结果为 "runoob.length is 6"
    println(str)
}
```

原生字符串和转义字符串内部都支持模板。 如果你需要在原生字符串中表示字面值 $ 字符（它不支持反斜杠转义），你可以用下列语法：

```
fun main(args: Array<String>) {
    val price = """
    ${'$'}9.99
    """
    println(price)  // 求值结果为 $9.99
}
```

Kotlin 基础语法 Kotlin 条件控制