2数据

2.1 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。

整数类型

Byte: 8 位，范围从 -128 到 127。

Short: 16 位，范围从 -32,768 到 32,767。

Int: 32 位，范围从 -2^31 到 2^31 - 1。

Long: 64 位，范围从 -2^63 到 2^63 - 1。

浮点数类型

Float: 32 位，单精度，带有 6-7 位有效数字。

Double: 64 位，双精度，带有 15-16 位有效数字。

字符类型

Char: 16 位的 Unicode 字符。

布尔类型

Boolean: 有两个值：true 和 false。

字符串类型

String: 一系列字符的序列。

数组类型

Kotlin 提供了数组类型来存储同种类型的元素，例如：

IntArray: 存储 Int 类型的数组。

DoubleArray: 存储 Double 类型的数组。

Array<T>: 泛型数组，可以存储任意类型。

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

Kotlin支持数字之间使用无效果的下划线，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

2.2比较两个数字

在 Kotlin 中，比较两个数字可以使用标准的比较运算符，包括 ==、!=、<、>、<= 和 >=。这些运算符可以比较基本数据类型，如 Int、Double、Float 等。

fun main() {

val a: Int = 5

val b: Int = 10

val c: Double = 5.0

// 相等和不相等比较

println("a == b: ${a == b}") // 输出 false

println("a != b: ${a != b}") // 输出 true

// 大小比较

println("a < b: ${a < b}") // 输出 true

println("a > b: ${a > b}") // 输出 false

println("a <= b: ${a <= b}") // 输出 true

println("a >= b: ${a >= b}") // 输出 false

// 不同类型的比较

println("a == c: ${a == c}") // 输出 true，Kotlin 自动将 Int 转换为 Double 进行比较

println("a < c: ${a < c}") // 输出 false

println("a > c: ${a > c}") // 输出 false

}

类型转换：Kotlin 在比较不同数据类型时会自动进行类型转换。例如，上面的例子中，a 是 Int 类型，而 c 是 Double 类型，Kotlin 会自动将 a 转换为 Double 类型进行比较。

相等性：Kotlin 中 == 运算符用于结构相等性比较，即值的比较，而 === 运算符用于引用相等性比较，即对象是否是同一个实例。在比较基本数据类型时，通常使用 == 运算符。

fun main() {

val x: Int = 1000

val y: Int = 1000

println("x == y: ${x == y}") // 输出 true，值相等

println("x === y: ${x === y}") // 输出 false，不同的实例

val z: Int = x

println("x === z: ${x === z}") // 输出 true，引用相同

}

2.3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

2.4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

2.5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’) {

// ……

}

}

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

}

2.6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

2.7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，我们分别用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) })//3个数 下标i对应的值为下标的2倍

//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 中数组是不协变的（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]

println(x[0]) // 输出结果：5

2.8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(">")。

2.9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

}