一些在 Kotlin 中广泛使用的语法习惯，如果你有更喜欢的语法习惯或者风格，建一个 pull request 贡献给我们吧！

创建 DTOs（POJOs/POCOs）

data class Customer(val name: String, val email: String)

会为 Customer 类提供以下功能：

所有属性的 getters （对于 var 定义的还有 setters）

equals()

hashCode()

toString()

copy()

所有属性的 component1()、 component2()……等等（参见[数据类](https://www.kotlincn.net/docs/reference/data-classes.html)）

函数的默认参数

fun foo(a: Int = 0, b: String = "") { …… }

过滤 list

val positives = list.filter { x -> x > 0 }

或者可以更短:

val positives = list.filter { it > 0 }

String 内插

println("Name $name")

类型判断

when (x) {

is Foo //-> ……

is Bar //-> ……

else //-> ……

}

遍历 map/pair型list

for ((k, v) in map) {

println("$k -> $v")

}

k、v 可以改成任意名字。

使用区间

for (i in 1..100) { …… } // 闭区间：包含 100

for (i in 1 until 100) { …… } // 半开区间：不包含 100

for (x in 2..10 step 2) { …… }

for (x in 10 downTo 1) { …… }

if (x in 1..10) { …… }

只读 list

val list = listOf("a", "b", "c")

只读 map

val map = mapOf("a" to 1, "b" to 2, "c" to 3)

访问 map

println(map["key"])

map["key"] = value

延迟属性

val p: String by lazy {

// 计算该字符串

}

扩展函数

fun String.spaceToCamelCase() { …… }

"Convert this to camelcase".spaceToCamelCase()

创建单例

object Resource {

val name = "Name"

}

If not null 缩写

val files = File("Test").listFiles()

println(files?.size)

If not null and else 缩写

val files = File("Test").listFiles()

println(files?.size ?: "empty")

if null 执行一个语句

val values = ……

val email = values["email"] ?: throw IllegalStateException("Email is missing!")

if not null 执行代码

val value = ……

value?.let {

…… // 代码会执行到此处, 假如data不为null

}

映射可空值（如果非空的话）

val value = ……

val mapped = value?.let { transformValue(it) } ?: defaultValueIfValueIsNull

返回 when 表达式

fun transform(color: String): Int {

return when (color) {

"Red" -> 0

"Green" -> 1

"Blue" -> 2

else -> throw IllegalArgumentException("Invalid color param value")

}

}

“try/catch”表达式

fun test() {

val result = try {

count()

} catch (e: ArithmeticException) {

throw IllegalStateException(e)

}

// 使用 result

}

“if”表达式

fun foo(param: Int) {

val result = if (param == 1) {

"one"

} else if (param == 2) {

"two"

} else {

"three"

}

}

返回类型为 Unit 的方法的 Builder 风格用法

fun arrayOfMinusOnes(size: Int): IntArray {

return IntArray(size).apply { fill(-1) }

}

单表达式函数

fun theAnswer() = 42

等价于

fun theAnswer(): Int {

return 42

}

单表达式函数与其它惯用法一起使用能简化代码，例如和 when 表达式一起使用：

fun transform(color: String): Int = when (color) {

"Red" -> 0

"Green" -> 1

"Blue" -> 2

else -> throw IllegalArgumentException("Invalid color param value")

}

对一个对象实例调用多个方法 （with）

class Turtle {

fun penDown()

fun penUp()

fun turn(degrees: Double)

fun forward(pixels: Double)

}

val myTurtle = Turtle()

with(myTurtle) { // 画一个 100 像素的正方形

penDown()

for(i in 1..4) {

forward(100.0)

turn(90.0)

}

penUp()

}

Java 7 的 try with resources

val stream = Files.newInputStream(Paths.get("/some/file.txt"))

stream.buffered().reader().use { reader ->

println(reader.readText())

}

对于需要泛型信息的泛型函数的适宜形式

// public final class Gson {

// ……

// public <T> T fromJson(JsonElement json, Class<T> classOfT) throws JsonSyntaxException {

// ……

inline fun <reified T: Any> Gson.fromJson(json: JsonElement): T = this.fromJson(json, T::class.java)

使用可空布尔

val b: Boolean? = ……

if (b == true) {

……

} else {

// `b` 是 false 或者 null

}

df