# Kotlin学习笔记

## Kotlin简介；

* 来自于著名的IDE IntelliJ IDEA(Android Studio基于此开发) 软件开发公司 JetBrains(位于东欧捷克)。
* 起源来自JetBrains的圣彼得堡团队，名称取自圣彼得堡附近的一个小岛(Kotlin Island)一种基于JVM的静态类型编程语言言。

## 变量，常量语法；

var关键字用来形容可变的变量，val关键字用来形容只可读的常量，

语法格式：1. 显示表示方式 ：关键字+变量名称 + ：+数据类型；(没有初始化的都要用 显示方式)；

1. 直接初始化的变量： 关键字 + 变量名 = 初始化数值 kotlin会根据初始化的数据自动辨别数据类型；

## 函数；

函数使用关键字function 缩写成fun；

1、创建无返回值函数：fun initView（v：View）{}

2、创建有返回值函数：fun sum （a:Int，b:Int）:Int{return a+b }

3、函数只有一个表达式或一个自推导型的返回值：

fun sum(a: Int, b: Int): Int = a + b

4、返回值是没有意义的：fun insignificance （str:String）:Unit { }

也可以省略Unit写成 fun insignificance （str:String）{ }

1. 中缀符号

## 控制流；

### if表达式

1.在kotlin中if是表达式，他可以返回一个值，是除了三元表达式之外唯一的表达式;

var max : Int

if（a > b）

max = a

带有else时

if(a > b)

max = a

else

max = b

用表达式的方式展示就是：var max = if(a > b) a else b 与三元表达式一样

2. if可以作为一个块结构，并且最后一句表达式为块的值；

var num = 20  
var result: Int = if (num > 20) {  
 num++  
 num  
} else if (num == 20) {  
 num--  
 num  
} else {  
 num = 30  
 num  
}  
*println*("if作为一个块结构 num== " + num)  
*println*("if作为一个块结构 result== " + result)

最终打印出来的结果：if作为一个块结构 num == 19 if作为一个块结构 result == 19

### for语句

Kotlin废除了Java中的for(初始值;条件；增减步长)这个规则。但是Kotlin中对于for循环语句新增了其他的规则，来满足刚提到的规则。

for循环提供迭代器用来遍历任何东西

for循环数组被编译为一个基于索引的循环，它不会创建一个迭代器对象

1. 递增： 关键字 until 范围：until[n,m) => 即大于等于n,小于m

for (i in 0 *until* 10) {  
 *println*("i=> " + i+"\t")  
}

打印结果：i => 0 i => 1 i => 2 i => 3 i => 4......i=>9

1. 递减：关键字downTo 范围：downTo[n,m] => 即小于等于n,大于等于m ,n > m

for (i in 20 *downTo* 15) {  
 *print*("i=> " + i+"\t")  
}

打印结果：i=> 20 i=> 19 i=> 18 i=> 17 i=> 16 i=> 15

1. 符号 . . 范围：.. [n,m]=> 即大于等于n，小于等于m （范围上与until有区别）

for (i in 10 .. 15){  
 *print*("i=> " + i+"\t")  
}

最终打印结果：i=> 10 i=> 11 i=> 12 i=> 13 i=> 14 i=> 15（前包含，后包含）

1. 设置步长：关键字step

for (i in 0 *until* 10 *step* 2){  
 *print*("i=>"+i)  
}

最终打印结果：i=>0 i=>2 i=>4 i=>6 i=>8

### 迭代

1. 遍历字符串：

var str = "TangAnna"

for (s in str) {  
 *print*("s=>" + s+"\t")  
}

最终打印结果：s=>T s=>a s=>n s=>g s=>A s=>n s=>n s=>a

1. 数组遍历：参考Kotlin学习笔记之数据类型中的数组模块

### when表达式

when表达式取代了switch，但是比switch更加的强大

when会对所有的分支进行检查直到有一个条件满足。 when 可以用做表达式或声明。如果用作表达式的话，那么满足条件的分支就是总表达式。如果用做声明，不满足条件的分支的的的值会被忽略，只保留满足条件的分支，而且返回的值也是最后一个表达式的值。

1. 传统的使用方式：

var x = 20  
when (x) {  
 2 -> *print*("x=>$x")  
 10 -> *print*("x=>$x")  
 20 -> *print*("x=>$x")  
 else -> *print*("没有符合条件的语句")  
}

1. 如果有分支可以用同样的方式处理的话， 分支条件可以连在一起处理：

var x = 10  
when (x) {  
 1, 3 -> *print*("x=> 2 或者 3")  
 10 -> *print*("x=>$x")  
 else -> *print*("没有符合条件的语句")  
}

1. 可以使用任意表达式作为分支的条件

var x = 5  
when(x){  
 in 10..20 -> *print*("x=> $x")  
 6 -> *print*("x=> 6")  
 else -> *print*("没有符合的条件")  
}

1. 作为表达式：

var x = 10  
var result = when (x) {  
 in 10..20 -> true  
 else -> false  
}

### while循环

while语句在循环刚开始时，会计算一次布尔表达式的值，如果满足条件，进入循环，如果不满足条件，将跳出循环。而在while语句的下一次迭代开始前会计算再一次布尔值，如此重复。