**基础语法**

//age为空返回-1

val ages2 = age?.toInt() ?: -1

为空的对象可以用这种方法给定默认值

重要的是你的脑子是否在思考

Is 表示是否是谋一个类的子类

!is 表示不是某一个类的子类



**Kotlin 基本数据类型**

.*trimIndent*() 删除字符串前部分统一的空格

.trimMargin()删除字符串|前的空格，可传入字符替换|

val s3 = """  
 衬衫的价额价格是${'$'}9.15  
   
""".*trimIndent*()

字符串有$应该这样写

arrayOf(1, 2, 3) 这种方法创建的数组会自动装箱创建如Int? 可为空的对象数组

IntArray等是底层使用int实现的不涉及自动装箱

Byte、Short、Int、Long、Float、Double 基本数据类型包括这几种



**逻辑控制**

if (x in 1..8) { println("x 在区间内") } 可以判断是否在某个区间

when (x) { 0, 1 -> print("x == 0 or x == 1") else -> print("otherwise") } 通过这种方式处理多个case 相同的逻辑

fun hasPrefix(x: Any) = when(x) { is String -> x.startsWith("prefix") else -> false } 方法写一行的情况下可以通过这种方式



loop@ for (i in 1..100) {

for (j in 1..100) {

if (……) break@loop

}

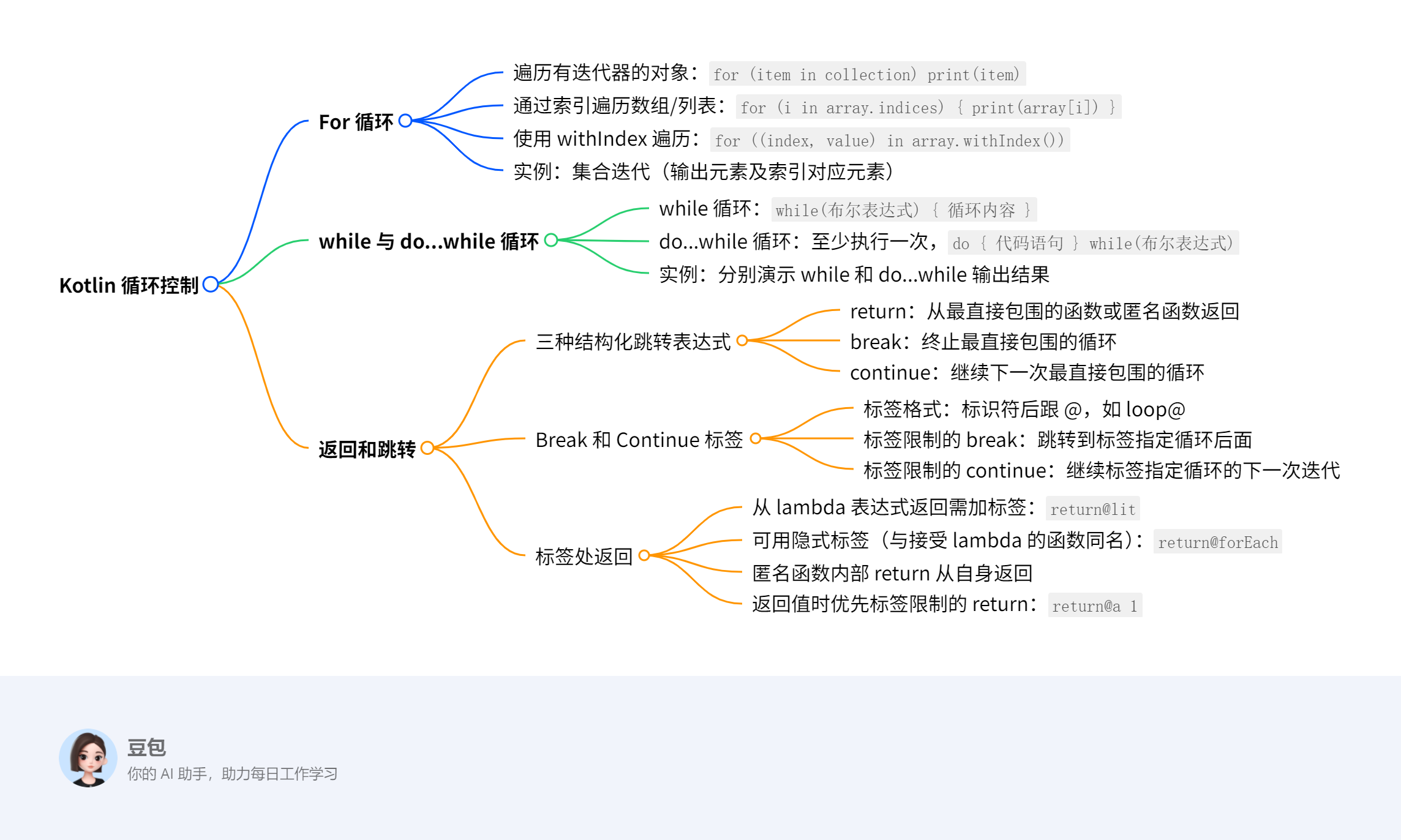
}

通过标签返回到指定位置，kotlin 遍历内部变量名可和外部相同，但是内部只能访问到内部外部访问到外部，所以双层循环遍历时还是要使用不同不同变量名 如i、j

fun **foo**() { ints.forEach lit@ { **if** (it == 0) **return**@lit print(it) } }

通过标签的方式用于返回lanbld

了解函数字面量 、匿名函数、局部函数、对象表达式



**类**

尽量通过lastinit 关键字解决成员变量初始化问题

关于实例对象”

var lastName: String = "zhang"

get() = field.toUpperCase() // 将变量赋值后转换为大写

set

var no: Int = 100

get() = field // 后端变量

set(value) {

if (value < 10) { // 如果传入的值小于 10 返回该值

field = value

} else {

field = -1 // 如果传入的值大于等于 10 返回 -1

}

}

var heiht: Float = 145.4f

private set

init {

println("FirstName is $firstName")

}

初始化相关代码放在初始化代码块中

主构造器参数可以再初始化使用，也可以在构造参数使用

Kotlin 中类默认是final 的如果需要继承该类可以在类前加open 需要继承的方法也加open

当抽象类实现该类时， 可通过abstract 继续抽象或者实现该类

类中嵌套类，两个类都是独立的类内部类不能访问外部类属性

类中嵌套的了加 inner 内部类将会持有外部类的引用可以访问外部属性和方法，但是类生命周期会绑定，可能存在内存泄露的问题



后面可以做一个学生管理系统来练习

**泛型**

基本定义

pen class Box<T> constructor(t: T) {  
 val value = t  
}  
  
fun <T> box(t: T): T {  
 println(t)  
 return t  
}

单约束和多约束

fun <T: Comparable<T>> listFun(t: List<T>): List<T> {  
 return t  
}  
  
fun <T> myFun(t: T) where T : Any, T : Comparable<T> {  
 println(t)  
}

协变和逆变

class MyType<out T : CharSequence>(t: T) {  
  
 private val value = t  
  
 fun getValue(): T {  
 return value  
 }  
  
 fun setValue(v: @UnsafeVariance T) {  
  
 }  
}  
  
class MyType2<in T : CharSequence>(t: T) {  
  
 fun setValue(t: T) {  
  
 }  
  
}

声明处形变

class W3School2<in A>(private var a: A) {  
  
 fun foo(a: A) {  
 println(this.a)  
 println(a)  
 }  
}  
  
class W3School<out A>(private val a: A) {  
 fun foo(): A {  
 return this.a  
 }  
}

class IntToStringFunction: Function<Int,String> {  
  
 override fun invoke(a: Int): String = a.toString()  
  
}  
  
interface Function<in A, out B> {  
 fun invoke(a: A): B  
}

*//协变和逆变*val myType = MyType("123")  
println("value: " + myType.getValue())  
val myType2 = MyType2("234")  
myType2.setValue("2222")  
*//声明处形变*var w3School = W3School<Any>(123)  
val w3School2 = W3School("456")  
w3School = w3School2*//协变*println(w3School.foo())  
  
val w3School3 = W3School2<Any>("abc")  
var w3School4 = W3School2<String>("def")  
w3School4 = w3School3*//逆变 子类接收父类*println("逆变")  
w3School4.foo("ccc")

星号投射

class IntToStringFunction: Function<Int,String> {  
  
 override fun invoke(a: Int): String = a.toString()  
  
}  
  
interface Function<in A, out B> {  
 fun invoke(a: A): B  
}

val func: Function<Int, \*> = IntToStringFunction()  
val result2: Any? = func.invoke(1)

