

# HenCoder Plus 讲义

---

## Kotlin 基础

---

### 函数声明

---

- 声明函数要用 `fun` 关键字，就像声明类要用 `class` 关键字一样
- 「函数参数」的「参数类型」是在「参数名」的右边
- 函数的「返回值」在「函数参数」右边使用 `:` 分隔，没有返回值时可以省略

声明没有返回值的函数：

```
fun main() {  
    //..  
}
```

声明有返回值的参数：

```
fun sum(x: Int, y: Int): Int {  
    return x + y  
}
```

### 变量声明

---

- 声明变量需要通过关键字，`var` 声明可读可写变量，`val` 声明只读变量
- 「类型」在「变量名」的右边，用 `:` 分割，同时如果满足「类型推断」，类型可以省略
- 创建对象直接调用构造器，不需要 `new` 关键字

声明可读可写变量：

```
var age: Int = 18
```

声明只读变量：

```
val name: String = "Hello, Kotlin!"
```

声明对象：

```
val user: User = User()
```

## 类型推断

在变量声明的基础上，如果表达式右边的类型是可以推断出来，那么类型可以省略：

```
var age = 18
val name = "Hello, Kotlin!"
val user = User()
```

## 继承类/实现接口

继承类和实现接口都是用的 `:`，如果类中没有构造器 (constructor)，需要在父类类名后面加上 `()`：

```
class MainActivity : BaseActivity(), View.OnClickListener
```

## 空安全设计

Kotlin 中的类型分为「可空类型」和「不可空类型」：

- 不可空类型  對應 Java 中的 `@NonNull` 註解。

```
val editText : EditText
```

- 可空类型  對應 Java 中的 `@Nullable` 註解。

```
val editText : EditText?
```

備註：空安全設計可以讓「空指針」的狀況能在編譯時期就被處理，而不會產生「NPE」；此外，比起 Java 的註解形式與警告，「Kotlin」的空指針設計具有強制性與更強的約束性。

## 调用符

- !! 强行调用符
- ? 安全调用符  
 相當於 `if.nn`，若該目標物件為空，則不執行。

## lateinit 关键字

- `lateinit` 只能修饰 `var` 可读可写变量(思考下为什么)
- `lateinit` 关键字声明的变量的类型必须是「不可空类型」
- `lateinit` 声明的变量不能有「初始值」
- `lateinit` 声明的变量不能是「基本数据类型」
- 在构造器中初始化的属性不需要 `lateinit` 关键字

備註：「lateinit」的目的為讓變量暫時不初始化，避免強制初始化的「null」賦值，換句話說，其存在目的就是為了避免不必要的「null」初始化，若又可以修飾「可空變量」，就非常矛盾。

## 平台类型

在类型后面面加上一个感叹号的类型是「平台类型」


Java 中可以通过注解减少这种平台类型的产生

平台類型盡量避免使用，其具「可空」特性，但卻視為「非空」來操作，與空安全的設計相違背；此外，該類型必須為 IDE 添加，無法自行聲明。

- `Nullable` 表示可空类型
- `NotNull` `NonNull` 表示不可空类型

備註：由於「Kotlin」與「Java」互通，而「Java」並沒有強制性的空型別管理，也就是說，在正常情況下，我們所調用「Java」的類型就皆屬於「可空類型」；但這樣對於開發非常麻煩，因此才會出現「平台類型」，也就是型別後面加上「!」，該類型別基本操作方式與「非空型別」相同。

## 类型判断

- `is` 判断属于某类型  對應 Java 中的 `instanceof`。
- `!is` 判断不属于某类型
- `as` 类型强转，失败时抛出类型强转失败异常
- `as?` 类型强转，但失败时不会抛出异常而是返回 `null`

## 获取 Class 对象

使用 `类名::class` 获取的是 Kotlin 的类型是 `KClass`

使用 `类名::class.java` 获取的是 Java 的类型

## setter/getter

在 Kotlin 声明属性的时候(没有使用 `private` 修饰)，会自动生成一个私有属性和一对公开的 `setter/getter` 函数。

在写 `setter/getter` 的时候使用 `field` 来代替内部的私有属性(防止递归栈溢出)。

 不能在 `setter/getter` 內使用 `this`，會造成無限循環。

为什么 `EditText.getText()` 的时候可以简化, 但是 `EditText.setText()` 的时候不能和 `TextView.setText()` 一样简化? 因为 `EditText.getText()` 获得的类型是 `Editable`, 对应的如果 `EditText.setText()` 传入的参数也是 `Editable` 就可以简化了。

```
val newEditable = Editable.Factory.getInstance().newEditable("Kotlin")
et_username.text = newEditable
```

## 构造器

使用 `constructor` 关键字声明构造器

```
class User {
    constructor()
}
```

如果我们在构造器主动调用了父类构造, 那么在继承类的时候就不能在类的后面加上小括号

```
constructor(context: Context) : this(context, null)
// 主动调用了父类的构造器
constructor(context: Context, attr: AttributeSet?) : super(context, attr)
```

## @JvmField 生成属性

通过 `@JvmField` 注解可以让编译器只生成一个 `public` 的成员属性, 不生成对应的 `setter/getter` 函数

## Any 和 Unit

- Any Kotlin 的顶层父类是 Any, 对应 Java 当中的 Object, 但是比 Object 少了 `wait()/notify()` 等函数
- Unit Kotlin 中的 Unit 对应 Java 中的 void

## 数组

使用 `arrayof()` 来创建数组, 基本数据类型使用对应的 `intArrayOf()` 等

因為一般數組存放的是包裝後類型, 因此需要封裝、拆裝; 因此基本數據類型建議使用其對應數組, 以增加效能。

備註：Kotlin 中的非空基本數據類型，對應著 Java 的基本類型，如 Int 對應 int，Float 對應 float，而可空的基本數據類型則對應 Java 的基本包裝類型，如 Int? 對應 Integer，Float? 對應 Float。

## 静态函数和属性

- 顶层函数
- object
- companion object

其中，「顶层函数」直接在文件中定义函数和属性，会直接生成静态的，在 Java 中通过「文件名Kt」来访问，同时可以通过 `@file:JvmName` 注解来修改这个「类名」。

需要注意，这种顶层函数不要声明在 `module` 内最顶层的包中，至少要在一个包中例如 `com`。不然不能方便使用。

`object` 和 `companion object` 都是生成单例对象，然后通过单例对象访问函数和属性的。

## @JvmStatic

通过这个注解将 `object` 和 `companion object` 的内部函数和属性，真正生成为静态的。

## 单例模式/匿名内部类

通过 `object` 关键字实现

```
// 单例
object Singleton {

}

// 匿名内部类
object : OnClickListener {

}
```

## 字符串

### 字符串模版

通过 `${}` 的形式来作为字符串模版

```
val number = 100
val text = "向你转账${number}元。"
// 如果只是单一的变量，可以省略掉 {}
val text2 = "向你转账$number元。"
```

## 多行字符串

```
val s = """
    我是第一行
    我是第二行
    我是第三行
    """.trimIndent()
```

## 区间

`200..299` 表示 `200 -> 299` 的区间(包括 `299`)

## when 关键字

Java 当中的 `switch` 的高级版，分支条件上可以支持表达式

## 受检异常

在 Kotlin 沒有 Checked Exception 之分，所以異常皆不強制捕獲。

Kotlin 不需要使用 `try-catch` 强制捕获异常

## 声明接口/抽象类/枚举/注解

```
// 声明抽象类
abstract class
// 声明接口
interface
// 声明注解
annotation class
// 声明枚举
enum class
```

## 编译期常量

在静态变量上加上 `const` 关键字变成编译期常量

## 标签

在 Java 中通过「类名.this 例如 `Outer.this`」获取目标类引用

在 Kotlin 中通过「`this@类名` 例如 `this@Outer`」获取目标类引用

## 遍历

记得让 IDE 来帮助生成 for 循环

```
for(item in items)
```

## 嵌套类

在 Kotlin 当中，嵌套类默认是静态内部类 (不持有外部类引用)

通过 `inner` 关键字声明为普通内部类 (内部使用外部对象时就会持有外部类引用)

## 可见性修饰符

默认的可见性修饰符是 `public`

新增的可见性修饰符 `internal` 表示当前模块可见

在 Java 中並沒有相對應的修飾。

## 注释

注释中可以在任意地方使用 `[]` 来引用目标，代替 Java 中的 `@param` `@link` 等。

## 非空断言

可空类型强制类型转换成不可空类型可以通过在变量后面加上 `!!`，来达到类型转换。

# open/final

Kotlin 中的类和函数，默认是被 `final` 修饰的 ( `abstract` 和 `override` 例外)

## 问题和建议？

课上技术相关的问题，都可以去群里和大家讨论，对于比较通用的、有价值的问题，可以去我们的知识星球提问。

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。



## 觉得好？

如果你觉得课程很棒，欢迎给我们好评呀！<https://ke.qq.com/comment/index.html?cid=381952>

一定要是你真的觉得好，再给我们好评。不要仅仅因为对扔物线的支持而好评（报名课程已经是你的最大支持了，再不够的话 B 站多来点三连我也很开心），另外我们也坚决不做好评返现等任何的交易。我们只希望，在课程对你有帮助的前提下，可以看到你温暖的评价。

## 更多内容：

- 网站：<https://hencoder.com>；<https://kaixue.io>



- 各大搜索引擎、微信公众号、微博、知乎、掘金、哔哩哔哩、YouTube、西瓜视频、抖音、快手、微视：统一账号「**扔物线**」，我会持续输出优质的技术内容，欢迎大家关注。
- 哔哩哔哩快捷传送门：<https://space.bilibili.com/27559447>

大家如果喜欢我们的课程，还请去扔物线的哔哩哔哩，帮我素质三连，感谢大家！