Kotlin 的程序结构

常量与变量

val 和 var 的故事

什么是常量

- val = value，值类型
- 类似 Java 的 final
- 不可能重复赋值
- 举例：
  - 运行时常量：val x = getX()
  - 编译期常量：const val x = 2

什么是变量

- var = variable
- 举例：
  - var x = “HelloWorld”//定义变量
  - x = “HiWorld”//再次赋值

类型推导

- 编译器可以推导量的类型
  - val string = “Hello”//推导出 String 类型
  - val int = 5 // Int 类型
  - var x = getString() + 5 //String 类型

函　数

得函数者得天下

什么是函数

- 以特定功能组织起来的代码块
  - fun [函数名]([参数列表]): [返回值类型]{ [函数体] }
  - fun [函数名]([参数列表]) = [表达式]
- 举例：
  - fun sayHi(name: String){ println(“Hi, $name”) }
  - fun sayHi(name: String) = println(“Hi, $name”)

匿名函数

- 无名氏一直是神一样的存在
- fun([参数列表]) ...
- 举例：
  - val sayHi = fun(name: String) = println(“Hi, $name”)

编写函数的注意事项

- 功能要单一
- 函数名要做到顾名思义
- 参数个数不要太多

Lambda 表达式

“我可是重量级的！”

什么是 Lambda 表达式

- 匿名函数
- 写法：{[参数列表] -> [函数体，最后一行是返回值]}
- 举例：
  - val sum = {a: Int, b: Int -> a + b}

Lambda 的类型表示举例

- ()->Unit
  - 无参，返回值为 Unit
- (Int) -> Int
  - 传入整型，返回一个整型
- (String, (String)->String)->Boolean
  - 传入字符串、Lambda 表达式，返回Boolean

Lambda 表达式的调用

- 用 () 进行调用
- 等价于 invoke()
- 举例：
  - val sum = {a: Int, b: Int -> a + b}
  - sum(2, 3)
  - sum.invoke(2, 3)

Lambda表达式的简化

- 函数参数调用时最后一个Lambda可以移出去
- 函数参数只有一个Lambda，调用时小括号可省略
- Lambda 只有一个参数可默认为 it
- 入参、返回值与形参一致的函数可以用函数引用的方式作为实参传入

类成员

类呀，你 Hollyhigh

什么是类成员

- 属性：或者说成员变量，类范围内的变量
- 方法：或者说成员函数，类范围内的函数

函数和方法的区别

- 函数强调功能本身，不考虑从属
- 方法的称呼通常是从类的角度出发
- 叫法不同而已，不要纠结

定义方法

- 写法与普通函数完全一致
- 举例：
  - class Hello{
  - fun sayHello(name: String) = println(“Hello, $name”)
  - }

定义属性

- 构造方法参数中 val / var 修饰的都是属性
- 类内部也可以定义属性
- 举例：
  - class Hello(val aField: Int, notAField: Int){
  - var anotherField: Float = 3f
  - }

属性访问控制

- 属性可以定义 getter/setter
- 举例：
  - val a: Int = 0
  - get() = field
  - var b: Float = 0f
  - set(value){ field = value }

属性初始化

- 属性的初始化尽量在构造方法中完成
- 无法在构造方法中初始化，尝试降级为局部变量
- var 用 lateinit 延迟初始化，val 用 lazy
- 可空类型谨慎用 null 直接初始化

运算符

+-*/%^?

基本运算符

- 任何类可以定义或者重载父类的基本运算符
- 通过运算符对应的具名函数来定义
- 对参数个数作要求，对参数和返回值类型不作要求
- 不能像 Scala 一样定义任意运算符

中缀表达式

- 只有一个参数，且用 infix 修饰的函数
- 举例：
  - class Book{ infix fun on(place: String){ ... } }
  - Book() on “My Desk”

分支表达式

重点是表达式哦

if 表达式

- if ... else
  - if(a == b) ... else if(a == c) ... else ...
- 表达式与完备性
  - val x = if(b < 0) 0 else b
  - val x = if(b < 0) 0 //错误，赋值时，分支必须完备

When 表达式

- 加强版 switch，支持任意类型
- 支持纯表达式条件分支（类似 if）
- 表达式与完备性

循环语句

看我的眼神 

for 循环

- 基本写法
  - for(element in elements) ...
- 给任意类实现Iterator 方法

While 循环

- 古董级语法
- do ... while(...) ...
- while(...) ...

跳过和终止循环

- 跳过当前循环用 continue
- 终止循环用 break
- 多层循环嵌套的终止结合标签使用
  - Outter@for(...) {
    Inner@while(i < 0) { if(...) break@Outter }
    }

异常捕获

猜猜这个是不是表达式

try...catch

- catch 分支匹配异常类型
- 表达式，可以用来赋值

finally

- finally 无论代码是否抛出异常都会执行

- 注意下面的写法：

```
return try{ x / y }catch(e: Exception){ 0 }finally{...

}
```

具名参数

报上名来！

具名参数

- 给函数的实参附上形参
- 举例：
  - fun sum(arg1: Int, arg2: Int) = arg1 + arg2
  - sum(arg1 = 2, arg2 = 3)

变长参数

任性如我

变长参数

- 某个参数可以接收多个值
- 可以不为最后一个参数
- 如果传参时有歧义，需要使用具名参数

Spread Operator

- 只支持展开 Array
- 只用于变长参数列表的实参
- 不能重载
- 不算一般意义上的运算符

默认参数

听我的，别自己瞎整了

默认参数

- 为函数参数指定默认值
- 可以为任意位置的参数指定默认值
- 传参时，如果有歧义，需要使用具名参数

一个命令行计算器

1 + 1 = 

导出可执行程序

这个好玩儿

导出可执行程序

- 配置 gradle 插件：application
- 指定 mainClass
- 运行gradle 任务：installApp