第一章、基础

一 、基本类型

1. 数字 Double Float Long Int Short Byte ( 常量可以使用下划线)
2. 转换 toByte toShort toInt toLong toFloat toDouble toChar
3. 运算
   * 1. shl(bits) – 有符号左移 (Java 的 <<)
     2. shr(bits) – 有符号右移 (Java 的 >>)
     3. ushr(bits) – 无符号右移 (Java 的 >>>)
     4. and(bits) – 位与
     5. or(bits) – 位或
     6. xor(bits) – 位异或
     7. inv() – 位非
4. 字符 \t、 \b、\n、\r、\'、\"、\\ 、 \$ '\uFF00'
5. 数组 Array： ByteArray、 ShortArray、IntArray
6. 字符串
   * 1. 原生字符串 使用三个引号（"""）
     2. 通过 [trimMargin()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/trim-margin.html) 函数去除前导空格
     3. 字符串模版
        + 1. $ 修饰任何变量 或者 ${ }
          2. 原声字符串中使用 $ 也是有效的

二 、 包

1. 关键字 import 什么都可以导入

三 、控制流

1. If表达式 （可以用作 变量 ，代码块 等）
2. When ( 如果不提供参数，所有的分支条件都是简单的布尔表达式，而当一个分支的条件为真时则执行该分支)
3. For 循环 （主要是针对数组，集合的遍历）
4. While 、do..while 照常使用

四 、控制流

1. Break 、continue

（默认从最直接包围它的位置跳出，但是可以在函数前面定义标签 xxx@ , 跳转后@xxx 来控制跳转位置）

1. Return ( 默认从最直接包围它的函数或者[匿名函数](http://www.kotlincn.net/docs/reference/lambdas.html" \l "匿名函数)返回(lambda ),

但是可以设置标签控制返回位置，)

第二章 类和对象

一、类和继承

1. 构造函数 constructor 没有修饰时，可以省掉，其次构造函数 需要使用constructor

2. 继承类之后 ，覆盖函数需要使用**override**

**3. class** C() : A(), B { // 编译器要求覆盖 f()：

**override** **fun** f() {

**super**<A>.f() // 调用 A.f()

**super**<B>.f() // 调用 B.f() }

}

1. 属性和字段
2. Val 只要get方法
3. Var setterVisibility: String

**private** **set** // 此 setter 是私有的并且有默认实现

**private** **get**

1. 幕后字段 ：默认的字段 field 代表其自己本身
2. 编译器常量 ：const  （编译的时候就初始化了，相当于 static final XXX，主要是用于赋值操作的时候，编译期间就能够被赋值 得到其数据，这样子就快多了）
3. 延迟初始化 lateinit 修饰var (必须初始化之后才能使用) ； val 可以使用lazy来初始化

三 、 接口

1. 接口可以使用接口代理进行操作接口的数据
2. Interface 定义接口（如果有多个父亲，则使用super<X>.XX() 来控制访问哪个）
3. 可见性修饰符
4. Private internal(模块可见) protected public
5. 扩展
6. 扩展方法： X . XX() ;
7. 可空接受者：Any?.XX()
8. 扩展属性：扩展函数一样 只是不能初始化 ，要像使用get 、set一样初始化才行
9. 伴生对象：一样使用，只是需要类名作为限定符调用他们
10. 数据类

1. 使用data修饰 ，自动有toString,equals()/hashCode(),copy,[componentN()](http://www.kotlincn.net/docs/reference/multi-declarations.html)

2. allopen ，noarg插件 （去掉final ,添加无参构造）

classpath "org.jetbrains.kotlin:kotlin-noarg:$kotlin\_version"  
 classpath "org.jetbrains.kotlin:kotlin-allopen:$kotlin\_version"

apply plugin: 'kotlin-noarg'  
apply plugin: 'kotlin-allopen'

noArg{  
 annotation("com.example.yanchunlan.kotlinhello.annotation.Poko")  
}

allOpen{

annotation("com.example.yanchunlan.kotlinhello.annotation.Poko")  
}

1. 密封类
2. Sealed修饰，子类有限，主要是密封住这个类，不让其他人拓展它
3. 泛型

１.

1. 嵌套类
2. Inner 修饰类 之后就是非静态内部类
3. 匿名内部类 使用Object： 开始
4. 枚举类
5. 每一个枚举都是一个实例，每一个枚举都是初始化了一个name
6. XX.valueOf(“ ＸX　”)　可以得到一个枚举的对象
7. 对象
8. 类的祖宗 就是Object，
9. 对象表达式：Object的对象表达式就是 Object: 开始写的，z可以针对函数 ，属性等，就如匿名部类
10. 对象声明：Object就一个单列 ，做简单的单列就可以使用Object来声明
11. 伴生对象：companion object 修饰，其实就是一个独立的静态单列，也是可以继承类， 实现接口的
12. @JvmField（属性）、@JvmStatic（方法）可以用于kotlin代码在java中使用
13. 委托
14. class Derived(b: Base) : Base by b 一句话实现对类代理
15. class SeniorManager(driver: Driver, writer: Writer) : Driver by driver, Writer by writer接口代理 ，其中Driver，Writer为接口 ，可以传入其实现类，去实现代理的目的（就是说由子类控制其中的参数类型）
16. 委托属性
17. Delegate主要是处理延迟加载的一个父接口
18. 延迟属性 lazy : 默认是LazyThreadSafetyMode.PUBLICATION 同步锁模式 ，当然可以自己设置自由的模式
19. 可观察属性 Observable : [Delegates.observable()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.properties/-delegates/observable.html)  有三个参数：被赋值的属性、旧值和新值
20. 属性存储在映射中 ： 一个映射（map）里存储属性的值
21. 属性委托要求 ：需要提供getValue ,setValue()

第三章 函数和Lambda表达式

一、函数

1. infix 表达式（成员函数或[扩展函数](http://www.kotlincn.net/docs/reference/extensions.html)、且只有一个参数的时候） ：允许使用中像运算符一样显示，不一定需要使用 方法.（） 的模式

2. 设置默认值（ 参数：类型=默认值，继承的函数不能设置默认值 ）

3. 不返回任何有用的值，它的返回类型是 Unit

4. vararg 无数参数

5.用 tailrec 修饰符标记并满足所需的形式时，编译器会优化该递归，留下一个快速而高效的基于循环的版本

6.重载的函数 在java中使用可以使用@JvmOverLoads

二、 Lambda表达式

1. （）-> T 最多支持23个参数

2. T.() 代表其对象

三、 内联函数

１．　函数引用有３种方式：

１＞　：：Ｘ

２＞　类名：：Ｘ

３＞　对象：：Ｘ

1. inline：修饰符影响函数本身和传给它的 lambda 表达式：所有这些都将内联到调用处。（noinline　禁用内联）
2. crossinline 禁止使用　控制流（ｒｅｔｕｒｎ，ｂｒｅａｋ，ｃｏｎｔｉｎｕｅ）
3. reified 修饰了，就能够在函数内使用它，并且调用的时候可以使用泛型＜Ｔ＞的方式去使用它
4. ｉｎｌｉｎｅ用在属性处就是一个内联属性
5. 协程

第四章 其他

一、解构声明

1. 定义：把一个对象 解构 成很多变量

2. componentN() 函数需要用 operator 关键字标记，以允许在解构声明中使用它们

3. \_ 表示不需要某个变量

4. 如果 lambda 表达式具有 Pair 类型（或者 Map.Entry 或任何其他具有相应 componentN 函数的类型）的参数，那么可以通过将它们放在括号中来引入多个新参数来取代单个新参数

二、 集合

1. 转换 toXXX()

2. list.requireNoNulls 全部数据便利一遍

三 、 区间

1. 正区间 ： .. rangeTo
2. 倒区间 ： downTo
3. 步长 ： step 一般用于区间之后
4. 区间之外 ： until
5. 反转之后 ： reversed()

四 、类型检查与转换

1. Is !is
2. as as? 检查是否为空需要注意

五 、 This表达式

1. this@XX 指定哪个对象

六 、 相等性

1. === /!== 引用相等/不等
2. ==/!= 结构相等/不等

七 、 操作符重载

1. 重载操作符的函数 ：需要用 operator 修饰符标记（ 任何类都可以定义，只对参数个数做要求，对其类型和返回无要求）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| +a | a.unaryPlus() |  | a++ | a.inc() |  | a in b | b.contains(a) |
| -a | a.unaryMinus() |  | a-- | a.dec() |  | a !in b | !b.contains(a) |
| !a | a.not() |  |  |  |  |  |  |

|  |  |
| --- | --- |
| a + b | a.plus(b) |
| a - b | a.minus(b) |
| a \* b | a.times(b) |
| a / b | a.div(b) |
| a % b | a.rem(b)、 a.mod(b) （已弃用） |
| a..b | a.rangeTo(b) |

|  |  |
| --- | --- |
| a += b | a.plusAssign(b) |
| a -= b | a.minusAssign(b) |
| a \*= b | a.timesAssign(b) |
| a /= b | a.divAssign(b) |
| a %= b | a.modAssign(b) |

|  |  |
| --- | --- |
| a() | a.invoke() |
| a(i) | a.invoke(i) |
| a(i, j) | a.invoke(i, j) |
| a(i\_1, ……, i\_n) | a.invoke(i\_1, ……, i\_n) |

|  |  |
| --- | --- |
| a > b | a.compareTo(b) > 0 |
| a < b | a.compareTo(b) < 0 |
| a >= b | a.compareTo(b) >= 0 |
| a <= b | a.compareTo(b) <= 0 |

八、 空安全：

属性？ : 属性不能为null

String？ : 允许定义的类型为null

?: 左侧表达式非空，elvis 操作符就返回其左侧表达式，否则返回右侧表达式

X!! :为null就会抛一个异常

as? :安全的类型转换，不能转换就返回null

集合的null过滤： X.filterNotNull()

九 、 异常

1. throw 表达式的类型是特殊类型 Nothing

十 、注解

视频笔记：

伴生对象：companion 相当于Static

By动态代理：

函数嵌套：

闭包：

委托：invote()

扩展函数 有专门的类管理

问题： 1 反射 由于包名跟文件名不同 反射容易错

2 调试 可能找不到位置

3 不能返回 java可以转化成kotlin,但是转不回来，可以试试反编译回来