第一个 kotlin 程序

```
fun main(args:Array<String>){
    var a=8
    var b=2
    println("${a*b}")
}
```

输出结果：16

kotlin 关键字

硬关键字：

as：用做类型转换或为 import 语句指定别名

as? ： 类型安全的类型转换

fun: 声明函数

in：在 for 循环中使用，in 还可以做双目运算符，检查一个值收费处于区间或者几个内，in 也可以在 when 表达式中使用，in 还可以修饰泛型参数，表示该泛型参数支持逆变

!in ： 双目运算符反义词

is：类型检查

null 表示空

object ： 用于声明对象表达式或定义命名对象

package：用于为当前文件指定包

return 声明函数的返回

super： 引用父类实现的方法或属性，或者子类构造器中调用父类构造器

this：代表当前类的对象或在构造器中调用当前类的其他构造器

typealias：定义类型别名

val：只读属性或变量

var：可变属性或变量

==========分隔符

软关键字 ：

by:用于接口或祖先类的实现代理给其他对象

constructor:声明构造器

delegate：用于指定该注解修饰委托属性存储其委托实例的字段

dynamic：主要用于在 kotlin 中引用一个动态类型

field：用于指定该注解修饰属性的幕后字段

file：用于指定该注解修饰该源文件本身

finally：异常处理中的 finally 块

get 用于声明属性的 getter 方法

init 声明初始化块

param 指定该注解修饰构造器参数

property 指定该注解修饰整个属性

reveiveris 指定该注解修饰扩展方法或扩展属性的接收者

set 声明属性 setter 方法

setparam 指定该注解修饰 setter 方法参数

where 泛型参数增加限制

abstract 修饰抽象类或抽象成员

annotation 修饰一个注解类

companion 声明一个伴生对象

const 声明编译时的常量

crossinline 禁止在传给内联函数的 Lambd 表达式中执行非局部返回

data 声明数据类

enum 枚举

external 声明某个方法不由 kotlin 实现

final 禁止被重写

infix 声明该函数能以双目运算符的格式执行

inline 内联函数 Lambda 表达式可在内联函数中执行

inner 声明内部类 内部类可以访问外部类的实例

internal 表示被修饰的声明只能在当前模块内可见

lateinit 修饰一个 non-null 属性，用于指定该属性可在构造器意外的地方完成初始化

noinline：禁止内联函数中个别 Lambda 表达式被内联化

open：修饰类 表示该类可派生子类，或者用于修饰成员，表示该成员可以被重写

out：修饰泛型参数，表明该泛型参数支持协变

override 声明重写父类成员

reified 修饰内联函数中的泛型形参

sealed 声明一个密封类

suspend 标识一个函数后 Lambda 表达式可作暂停

tailrec 修饰一个函数可作为尾随递归函数使用

vararg 修饰形参，表明该参数是个数可变的形参

声明变量

```
fun main(args: Array<String>) {
    var b: Int
    var name = "crazyit.org"
    b = 20
    name = "fkit-org";
    var age: Int = 25
    var sun : String ="500"
    val book = "Java-code";

    println("$b\t,$name\t,$age\t,$sun\t,$book")
}
```

参数值：20 ,fkit-org ,25 ,500 ,Java-code

? 在 kotlin 中的使用，类型扩展，让变量可以接受空字段

```
fun main(args: Array<String>) {
```

```
    var b: Int
    var name = "crazyit.org"
    b = 20
    name = "fkit-org";
    var age: Int = 25
    var sun : String ="500"
    val book = "Java-code";

    var like : Long = 29898456232

    //添加问号，可以让Byte Short Int Long 存储null 值
  var sex : Int? = null


    println("$b\t,$name\t,$age\t,$sun\t,$book")
}
```

二进制和十六进制

```
fun main(args: Array<String>) {
    var b: Int
    var name = "crazyit.org"
    b = 20
    name = "fkit-org";
    var age: Int = 25
    var sun : String ="500"
    val book = "Java-code";

    var like : Long = 29898456232

    //添加问号，可以让Byte Short Int Long 存储null 值
  var sex : Int? = null

  var ko : Int = 0x14;  //十六进制 0x 开头
  var ko2 : Int = 0b101001; //二进制 0b 开头

  println("$b\t,$name\t,$age\t,$sun\t,$book\t,$ko\t,$ko2")
}
```

输出结果:20 ,fkit-org ,25 ,500 ,Java-code ,20 ,41

为了增加代码可读性，kotlin 支持数字之间，用下划线分开

```
fun main(args: Array<String>) {
    var b: Int
    var name = "crazyit.org"
    b = 20
    name = "fkit-org";
    var age: Int = 25
    var sun : String ="500"
    val book = "Java-code";

    var like : Long = 29898456232

    //添加问号，可以让Byte Short Int Long 存储null 值
  var sex : Int? = null

    var ko : Int = 0x14;  //十六进制 0x 开头
  var ko2 : Int = 0b101001; //二进制 0b 开头

  var  ko3 : Int = 987_654_32;

    println("$b\t,$name\t,$age\t,$sun\t,$book\t,$ko\t,$ko2\t,$ko3")
}
```

并不会影响最终结果的显示

类型转换

toByte()

toShort()

toInt()

toLong()

toFloat()

toDouble()

toChar()

for 循环　从 0 加到 100

```
fun main(args: Array<String>) {
    var sum: Int = 0;
    for (i in 0..100) {
        sum += i;
    }
    println(sum)

}
```

输出结果：5050

设置变量可为空，安全调用

```
fun main(args: Array<String>) {

    var name : String? = "blues";
    println(name?.length)
    name = null
    println(name?.length)

}
```

类型转换 string 转 int ，直接调用 toInt()方法

```
fun main(args: Array<String>) {

    var a : Int = 89
    var b : Int = 213

    var price : String = "2.35"
```

```
    var doller : String = "13"

    var s1 = price.toDouble()
    var  s2 : Int = doller.toInt()

   // val c = if (a > b) true else false
    val c:Boolean = a>b
    println("$s1\t,$s2")

}
```

输出结果：2.35　　,13

in 包含，包括，类似于 contains 用法

```
fun main(args: Array<String>) {

    var str = "fkjava.org"
    println(str.contains("java"))

    println("java" in str)

    val array = arrayOf(24,45,100,-3,30)
    println(array.contains(100))
    println(100 in array)

}
```

kotlin 支持直接下标索引的方式，代替 java 的 get 方法，两种写法一样

```
fun main(args: Array<String>) {

    var str = "fkjava.org"
    println(str.get(2))
```

```
    println(str[2])


}
```

判断字符串相等

```
fun main(args: Array<String>) {

    var s1 : String = "java"
    var s2 : String = "java"

    println(s1==s2)
    println(s1.equals(s2))
}
```

比较运算符

```
fun main(args: Array<String>) {

    var s1: String = "javascript"
    var s2: String = "kotlin"

    //两个字符串并不能直接这样比较
  val s3: Boolean = s1 > s2

    //可以比较长度
  val s4: Boolean = s1.length > s2.length

    println("$s3\t,$s4\t")

}
```

输出结果：false true

区间运算符

```
fun main(args: Array<String>) {

    //闭区间运算符
  var range: IntRange = 0..5
    var sum: Int = 0
    for (i in range) {
        sum += i
    }
    println(sum)
    println("======================")


    //半开区间运算符
  val books: Array<String> = arrayOf("java", "kotlin", "flutter", "rub
y", "php")
    for (index in 0 until books.size) {
        println("${index + 1}种语言是：${books[index]}")
    }
    println("======================")


    //反向区间
  var range2: IntProgression = 6 downTo 0

    for (i in range2) {
        println(i)
    }

    println("======================")

    //设置区间的元素间隔
  var range3: IntProgression = 6 downTo 0 step 3
    for (i in range3) {
        println(i)
    }

}
```

输出结果：

15

====================

1 种语言是：java

2 种语言是：kotlin

3 种语言是：flutter

4 种语言是：ruby

5 种语言是：php

====================

6

5

4

3

2

1

0

====================

6

3

0

kotlin 的 when 可以代替 java 的 switch 语句，并且功能更加强大

kotlin 循环分为：while do-while for-in

成绩判断，if-else 结构

```
fun main(args: Array<String>) {
```

```
    val grade: Int = 101
    var result: String;

    if (grade < 60) {
        result = "不及格"
    } else if (grade < 70) {
        result = "中等"
    } else if (grade < 85) {
        result = "良好"
    } else if (grade < 101) {
        result = "优秀"
    } else {
        result = "输入成绩不正确，请核对重新输入"
    }
    println(result)

}
```

when 的应用，代替了 switch-case

```
fun main(args: Array<String>) {

    val grade: Int = -9
    var result: String? = null;

    when (grade) {
        0, 1, 2, 3, 4, 5 -> {
            result = "不及格";
        }
        6, 7, 8 -> {
            result = "良好";
        }
        9, 10 -> {
            result = "优秀";
        }
        else -> {
            result = "输入成绩错误"
        }
```

```
    }
    println(result)
}
```

区间 when 的用法

```
fun main(args: Array<String>) {

    val grade: Int = 120
    var result: String? = null;

    when (grade) {
        in 0..59 -> {
            result = "及格"
        }
        in 60..79 -> {
            result = "一般"
        }
        in 80..100 -> {
            result = "优秀"
        }
        else -> {
            result = "成绩输入错误"
        }
    }
    println(result)
}
```

when 还可以直接代替 if-else 语句使用，功能强大

```
fun main(args: Array<String>) {

    var a: Int = 16
    var b: Int = 9;

    when (a > b) {
        true -> {
            println("a 大于 b")
```

```
        }
        false -> {
            println("a 小于 b")
        }
    }

}
```

或者可以比较字符串长度

```
fun main(args: Array<String>) {

    var a: String = "javascript"
    var b: String = "googlekotlin";

    when (a.length > b.length) {
        true -> {
            println("a 大于 b")
        }
        false -> {
            println("a 小于 b")
        }
    }

}
```

比较经典的三种循环方式，while 循环，do-while 循环，for 循环，打印 0-9 的值

```
fun main(args: Array<String>) {

    var a: Int = 0;

    while (a < 10) {
        println(a)
        a++
    }

}
```

```
fun main(args: Array<String>) {

    var a: Int = 0;

    do {
        println(a)
        a++
    } while (a < 10)

}


fun main(args: Array<String>) {

    var a: Int = 0;

    for (a in 0..9){
        println(a)
    }

}
```

九九乘法表

```
fun main(args: Array<String>) {

    for (i in 0 until 10) {
        for (j in 1 until i+1) {
            println("$j*$i=${i * j}")
        }
    }
}
```

continue 忽略本次循环，执行下一次循环

```
fun main(args: Array<String>) {

    for(i in 0..3){
```

```
        println("判断处理前：$i")
        if (i==1){
            continue
        }
        println("判断处理后：$i")
    }
}
```

输出结果：

判断处理前：0

判断处理后：0

判断处理前：1

判断处理前：2

判断处理后：2

判断处理前：3

判断处理后：3

在 kotlin 中，break 和 return 都可以表示结束循环

```
fun main(args: Array<String>) {

    for(i in 0..3){
        println("判断处理前：$i")
        if (i==1){
            return
        }
        println("判断处理后：$i")
    }
}
```

输出结果：

判断处理前：0

判断处理后：0

判断处理前：1

创建数组

```
import java.util.*

fun main(args: Array<String>) {

    var array1 = arrayOf("java","dart","swift","ruby")
    var array2 = arrayOf(12,56,8,46)
    var array3 = arrayOfNulls<Double>(5)
    var array4 = arrayOfNulls<Int>(6)
    var array5 = emptyArray<String>()
    var array6 = emptyArray<Int>()
    var array7 = intArrayOf(45,89,3,6,18)
    var array8 = doubleArrayOf(2.3,5.69,6.21,4.78)

    println("${Arrays.toString(array7)}\n${Arrays.toString(array8)}")

}
```

数组的读数据和写数据

```
import java.util.*

fun main(args: Array<String>) {

    var array1 = arrayOf("java", "dart", "swift", "ruby")

    println(array1[1])

    println(array1.get(1))

    array1[1] = "php";

    array1.set(2, "javascript")
```

```
    println(Arrays.toString(array1))

}
```

输出结果：

dart

dart

[java, php, javascript, ruby]

arrayofNulls 指定数组长度和设置内容

```
import java.util.*

fun main(args: Array<String>) {

    var array1 = arrayOfNulls<String>(5)
    array1[0] = "cat";
    array1[1] = "pig";
    array1[2] = "dog";
    array1[3] = "rabbit";
    array1[4] = "snake";

    for (i in 0 until array1.size) {
        println(array1[i])
    }

}
```

输出结果：

cat

pig

dog

rabbit

snake

kotlin 数组的常用方法：

aslist ： 数组转换为 list 集合

average： 计算数组平均值

contains ： 判断数组中是否包含某元素

distinct：去除掉数组中的重复元素，并把新的数组元素赋值给 list 集合

max：最大值

min：最小值

```
import java.util.*

fun main(args: Array<String>) {

    var array1 = arrayOfNulls<String>(5)
    array1[0] = "cat";
    array1[1] = "pig";
    array1[2] = "dog";
    array1[3] = "rabbit";
    array1[4] = "pig";

    var list1 = array1.asList(); //数组转换为list 集合
    var array2 = arrayOf(7, 5, 6, 17, 8)
    println(array2.average())   //计算数组平均值

    println("==================")

    val isContain1 = "pig" in array1
    val isContain2 = 9 in array2
    val isContain3 = array1.contains("cat")  //判断数组中是否包含某元素

    val list2 = array2.distinct() //去除数组中的重复元素，并把新的数组元素
赋值给 list 集合
    val list3 = array1.distinct()

    println(list2)
    println("==================")
    println(list3)
    println("==================")
```

```
    println("$isContain1\t$isContain2\t$isContain3")
    println("==================")
    println(list1)
    println("==================")
    val list4 = array1.drop(2)  //去除掉数组前面两个元素
    println(list4);
    println("==================")
    val list5 = array1.dropLast(2); //去除掉数组后面两个元素
    println(list5)

    val index = array1.indexOf("dog")

    println("pig 索引：$index")


    println("最大值：${array2.max()}\t 最小值：${array2.min()}")


}
```

set 集合

```
fun main(args: Array<String>) {

    var set1 = setOf<String>("java", "kotlin", "go")
    println(set1)
    var mutableSet2 = mutableSetOf<String>("java", "kotlin", "php")
    println(mutableSet2)

    mutableSet2.remove("java")

    println(mutableSet2)

    var hashset3 = hashSetOf<String>("pig", "cat", "ege") //乱序  比较特殊，其他两个linkset 和treeset 是有序的
    hashset3.remove("cat")
    println(hashset3)
```

```
    var linkedHashSet4 = linkedSetOf<String>("apple", "orange", "banan
a")
    linkedHashSet4.add("pineapple")
    println(linkedHashSet4)

    var treeSet5 = sortedSetOf("table", "chair", "cabinet")
    treeSet5.add("bed")
    println(treeSet5)


}
```

输出结果：

[java, kotlin, go]

[java, kotlin, php]

[kotlin, php]

[ege, pig]

[apple, orange, banana, pineapple]

[bed, cabinet, chair, table]

set 遍历

```
fun main(args: Array<String>) {

    var set1 = setOf<String>("java", "kotlin", "go")
    println(set1)

    for (set2 in set1){
        println("索引：$set2")
    }

    var mutableSet2 = mutableSetOf<String>("java", "kotlin", "php")
    println(mutableSet2)
```

```
    mutableSet2.remove("java")

    println(mutableSet2)

    var hashset3 = hashSetOf<String>("pig", "cat", "ege") //乱序  比较特殊，其他两个linkset 和treeset 是有序的
    hashset3.remove("cat")
    println(hashset3)

    var linkedHashSet4 = linkedSetOf<String>("apple", "orange", "banana")
    linkedHashSet4.add("pineapple")
    println(linkedHashSet4)

    var treeSet5 = sortedSetOf("table", "chair", "cabinet")
    treeSet5.add("bed")
    println(treeSet5)


}
```

创建 list 集合

```
fun main(args: Array<String>) {

    var list1 = mutableListOf<String>("java", "swift", "golang", "dart") //可变的list 集合

    var list2 = listOf<String>("pig", "cat", "dog")

    var list3 = listOfNotNull("apple", "banana", "orange");
a
    list1.removeAt(0);

    println(list1)
    println(list2)
    println(list3)
}
```

输出结果：

[swift, golang, dart]

[pig, cat, dog]

[apple, banana, orange]

返回 list 集合索引

```
fun main(args: Array<String>) {

    var list1 = mutableListOf<String>("java", "swift", "golang", "dar
t") //可变的 list 集合

    var list2 = listOf<String>("pig", "cat", "dog")

    var list3 = listOfNotNull("apple", "banana", "orange");

    list1.removeAt(0);

    val a = list2.indexOf("cat")

    println("list2 索引位置：$a") //返回-1 表示集合中不存在此元素
    println(list1)
    println(list2)
    println(list3)
}
```

Map 多种创建方式和遍历，元素无序

```
fun main(args: Array<String>) {

    var map1 = mutableMapOf<String, String>("红楼梦" to "曹雪芹", "西游
记" to "吴承恩", "水浒传" to "施耐庵", "三国演义" to "罗贯中")

    map1.remove("红楼梦")

    println(map1)
```

```
    var map2 = mapOf<Int,String>(0 to "张飞",1 to "关羽",2 to "刘备")

    println("============")

    println(map2)

    var map3 = linkedMapOf<String,String>("红楼梦" to "曹雪芹", "西游记" to "吴承恩", "水浒传" to "施耐庵", "三国演义" to "罗贯中")

    map3.remove("三国演义")
    println(map3)

    println("============")

    var map4 = sortedMapOf("红楼梦" to "曹雪芹", "西游记" to "吴承恩", "水浒传" to "施耐庵", "三国演义" to "罗贯中")

    map4.remove("水浒传")

    println(map4)
    println("============")

    for (en in map1.entries) {
        println("${en.key}->${en.value}")
    }

    println("============")

    map1.forEach({ println("${it.key}->${it.value}") })
}
```

输出结果：

{西游记=吴承恩, 水浒传=施耐庵, 三国演义=罗贯中}

============

{0=张飞, 1=关羽, 2=刘备}

{红楼梦=曹雪芹, 西游记=吴承恩, 水浒传=施耐庵}

============

{三国演义=罗贯中，红楼梦=曹雪芹，西游记=吴承恩}

==========

西游记->吴承恩

水浒传->施耐庵

三国演义->罗贯中

==========

西游记->吴承恩

水浒传->施耐庵

三国演义->罗贯中

函数调用

```
fun main(args: Array<String>) {
    var a = 6
    var b = 9
    var result = max(a, b)
    println("result:${result}")
    println(sayHi("孙悟空"))
}


fun max(x: Int, y: Int): Int {
    val z = if (x > y) x else y
    return z
}

fun sayHi(name: String): String {
    println("正在执行函数 sayhi")
    return "${name},先生，您好"
}
```

运行结果：

```
result:9

正在执行函数 sayhi
```

```
孙悟空,先生，您好

函数定义
fun main(args: Array<String>) {

    foo()
    sayHi("明先生")
    showMsg("北京欢迎你",3)
}

fun foo(){
    println("定义一个函数，函数既无形参，也无返回值")
}

fun sayHi(name:String) : Unit{
    println("===程序执行 sayHi 函数===")
    println("${name},先生您好")
}

fun showMsg(msg:String,count:Int){
    for (i in 1 until count){
        println(msg)
    }
}


函数递归
fun main(args: Array<String>) {
    rec()
}

var count = 0

fun rec() {
    count++
    if (count < 6) {
        println("count:$count")
        rec()
```

```
    }
}
```

输出结果：

```
count:1

count:2

count:3

count:4

count:5
```

递归

```
fun main(args: Array<String>) {
    val number = 6
    val factorial = fact(number)
    println("Factorial of $number = $factorial")
}

tailrec fun fact(n: Int, temp: Int = 1): Int {
    return if (n == 1) {
        temp
    } else {
        fact(n - 1, temp * n)
    }
}
```

输出结果

```
Factorial of 6 = 720
```

单表达式函数

```
fun main(args: Array<String>) {

    val a = area(3.2,4.9)
```

```
    println("area 面积为:$a")

}

fun area(x:Double,y:Double) : Double = x*y
```

函数的形参，自动监听传入参数

```
fun main(args: Array<String>) {
    println(girth(3.5,4.8))
    println("========")
    println(girth(width = 3.5,height = 4.8))
    println("========")
    println(girth(height = 4.8,width = 3.5))
    println("========")
    println(girth(3.5,height = 4.8))
}

fun girth(width:Double,height:Double) : Double{
    println("width:${width}")
    println("height:${height}")
    return 2*(width + height)
}
```

输出结果：

```
width:3.5
height:4.8
16.6
========
width:3.5
height:4.8
16.6
========
```

```
width:3.5

height:4.8

16.6

========

width:3.5

height:4.8

16.6
```

函数重载

```
fun main(args: Array<String>) {
    novel()
    println(novel("西游记"))
    println(novel(699,"钢铁是怎样炼成的"))
}

fun novel() {
    println("小说大全")
}

fun novel(name: String): String {
    return name;
}

fun novel(nums: Int, name: String): String {
    return "书名：$name,页数：$nums";
}
```

类的方法调用

```
fun main(args: Array<String>) {
    var mPerson =  Person("test");
    mPerson.run()
```

```
}


class Person constructor(firstName: String) {

    fun jump(){
        println("jumoing...")
    }

    fun run(){
        this.jump()
        println("正在执行 run 方法")
    }

}
```

默认执行的构造方法

```
fun main(args: Array<String>) {
   println(ThisInConstructor().foo)

}


class ThisInConstructor {
     var foo : Int = 0

    constructor(){
        this.foo=6
    }
}
```

构造方法

```
fun main(args: Array<String>) {

    val mItem = Item("200613","西游记",23.6);
    println(mItem.toString())
    println(mItem.batCode)
```

```
    println(mItem.price)
    println(mItem.name)
}


class Item (batCode:String,name:String,price:Double){
    val batCode = batCode;
    val name = name;
    val price = price;

    override fun toString(): String {
        return "序号：$batCode,书名：$name,价格：$price";
    }


}
```

输出结果：

序号：200613,书名：西游记,价格：23.6

200613

23.6

西游记

优化版本

```
fun main(args: Array<String>) {

    val mItem = Item("200613","西游记",23.6);

    mItem.batCode = "201903"
    mItem.name = "三国演义"
    mItem.price = 99.12

    println(mItem.toString())
    println(mItem.batCode)
    println(mItem.price)
```

```
    println(mItem.name)
}


class Item (batCode:String,name:String,price:Double){
    var batCode = batCode;
    var name = name;
    var price = price;

    override fun toString(): String {
        return "序号：$batCode,书名：$name,价格：$price";
    }


}
```

可以修改属性的值，把默认的值改掉了

输出结果:

序号：201903,书名：三国演义,价格：99.12

201903

99.12

三国演义

init 初始化块

```
fun main(args: Array<String>) {

    var oc1 = ConstructorOverload()

    var oc2 = ConstructorOverload("西游记",300000)

    println("${oc1.count}${oc1.name}")
    println("${oc2.count}${oc2.name}")

}
```

```
class ConstructorOverload {
    var name:String?
    var count:Int
    init{
        println("初始化块")
    }

    constructor(){
        name = null
        count = 0
    }

    constructor(name:String,count:Int){
        this.name = name
        this.count = count

    }

}
```

初始化字段

```
fun main(args: Array<String>) {
    var us1 = User("孙悟空")
    println("${us1.name} => ${us1.age} => ${us1.info}")

    var us2 = User("镇元大仙",21)
    println("${us2.name} => ${us2.age} => ${us2.info}")

    var us3 = User("灵感大王",20,"一条金鱼")
    println("${us3.name} => ${us3.age} => ${us3.info}")
}


class User(name: String) {
```

```
    var name: String
    var age: Int
    var info: String? = null

    init {
        println("user 初始化块")
        this.name = name
        this.age = 0
    }

    constructor(name: String, age: Int) : this(name) {
        this.age = age
    }

    constructor(name: String, age: Int, info: String) : this(name, ag
e) {
        this.info = info
    }

}
```

输出字段：

user 初始化块

孙悟空 => 0 => null

user 初始化块

镇元大仙 => 21 => null

user 初始化块

灵感大王 => 20 => 一条金鱼

最简单的构造方法，传参

```
fun main(args: Array<String>) {

    var item = Item("201207", 23.2)
    println(item.price)
```

```
    println(item.code)
}


class Item(val code: String, var price: Double) {


}
```

无参构造器使用默认值

```
fun main(args: Array<String>) {

    var customer = Customer()
    println(customer.name)
    println(customer.addr)

    var customer2 = Customer("孙悟空","花果山")
    println(customer2.name)
    println(customer2.addr)
}



class Customer(val name: String = "学校", var addr: String = "观山湖
区") {


}
```

父类和子类构造器

```
fun main(args: Array<String>) {

    Sub()
    println("===========")
    Sub("鲁智深")
    println("===========")
```

```
    Sub("张飞",20)
    println("===========")
}


open class Base {

    constructor() {
        println("base 类的无参构造器")
    }

    constructor(name: String) {
        println("base 类一个参数构造器:${name}")
    }

}
class Sub :Base{

    constructor(){
        println("sub 的无参构造器")
    }

    constructor(name:String) : super(name){
        println("sub 类一个参数构造器: $name")
    }

    constructor(name: String,age:Int) : this(name){
        println("sub 类两个参数构造器:$name,$age")
    }

}
```

输出结果:

base 类的无参构造器

sub 的无参构造器

==========

base 类一个参数构造器:鲁智深

sub 类一个参数构造器：鲁智深

==========

base 类一个参数构造器:张飞

sub 类一个参数构造器：张飞

sub 类两个参数构造器:张飞,20

==========

继承，先执行父类的方法，然后执行子类

```
open class Bird {

    open fun fly(){
        println("会飞的一种鸟")
    }

}

class Seagull : Bird() {

    override fun fly() {
        super.fly()
        println("海面上飞行")
    }

}

fun main(array: Array<String>) {
    var seagull = Seagull()
    seagull.fly()
}
```

kotlin 里面的属性或者方法，想要实现重写的话，必须添加 open 字段

子类继承父类方法

```
fun main(args: Array<String>) {
    var book = Book()
```

```
    println("${book.name},${book.price},${book.validDays}")
}


open class Item {
    open protected  var price : Double = 10.9
    open val  name : String = "西游记"
    open var validDays : Int = 7
}



class Book : Item{
    override public var price: Double
    override val name = "图书"
    override var validDays: Int = 0

    constructor(){
        price = 3.8

    }
}
```

对象表达式：

```
interface Outputable {
    fun output(msg: String)
}

abstract class ObjectExprTest(var price: Double) {

    abstract val name: String
    abstract fun printInfo()

}


fun main(args: Array<String>) {

    var obl = object : Outputable {
```

```
        override fun output(msg: String) {
            for (i in 1 until 6) {
                println("<h${i}>${msg}</h${i}>")
            }
        }
    }
    ob1.output("疯狂教育软件中心")
    println("==============")
    var ob2 = object {
        init {
            println("初始化块")
        }

        var name = "kotlin"
        fun test() {
            println("test 方法")
        }

        inner class Foo

    }

    println(ob2.name)
    ob2.test()
    println("---------------")

    var ob3 = object : Outputable, ObjectExprTest(28.8) {
        override val name: String
            get() = "激光打印机"

        override fun printInfo() {
            println("高速激光打印机，支持双面打印")
        }


        override fun output(msg: String) {
            println("输出信息：$msg")
        }
```

```
    }

    println(ob3.name)
    ob3.output("kotlin 真不错")
    ob3.printInfo()

}
```

输出结果：

<h1>疯狂教育软件中心</h1>

<h2>疯狂教育软件中心</h2>

<h3>疯狂教育软件中心</h3>

<h4>疯狂教育软件中心</h4>

<h5>疯狂教育软件中心</h5>

==========

初始化块

kotlin

test 方法

激光打印机

输出信息：kotlin 真不错

高速激光打印机，支持双面打印

泛型

```
open class Apple<T>{

    open var info : T?

    constructor(){
```

```
        info = null
    }

    constructor(info:T){
        this.info = info
    }



}

fun main(args:Array<String>) {

    var a1 : Apple<String> = Apple<String>("苹果");

    println("a1:${a1.info}")

    var a2 : Apple<Int> = Apple(3)

    println("a2:${a2.info}")

    var a3 = Apple(5.67)
    println("a3:${a3.info}")

}
```