# Kotlin

## 变量

`var 可读可写`
`val 只读`

定义编译时常量
`const val PI = 3.14151`
定义在函数之外,编译期间使用

---

## 具名参数

```
fun main(args: Array<String>) {
    println(计算圆的周长2(直径 = 2.0f))

}

fun 计算圆的周长(py: Float = 3.14f, 半径: Float): Float {
    return 2 * py * 半径;
}

fun 计算圆的周长2(py: Float = 3.14f, 直径: Float): Float {
    return py * 直径;
}
```

2. Brain/Voice/Eye Interface

`redaLine();`

3. 递归

```
fun 阶乘(itt: BigInteger): BigInteger {
    return if (itt == BigInteger.ONE) {
        BigInteger.ONE
    } else {
        itt * 阶乘(itt - BigInteger.ONE)
    }
}
```

kotlin中 `Unit`类似`java`中的`void`,但是一方法或者函数表达式都是有返回值的，这里返回一个`Unit`的单例 `Any`其实就跟`Java`里的`Object`是一样的，也就是说在`Kotlin`中`Any`取代了`Java`中的`Object`，成为了`Kotlin`中所有类的父类。 `Nothing` 是一个类，这个类构造器是私有的，也就是说我们从外面是无法构造一个 `Nothing` 对象的。前面说每一个方法都有返回值，且返回值至少也是一个 `Unit`，这是对正常方法来说

的。如果一个方法的返回类型定义为 Nothing ，那么这个方法就是无法正常返回的。可以这么理解，Koltin中一切方法都是表达式，也就是都有返回值，那么正常方法返回 Unit ，无法正常返回的方法就返回 Nothing 。

## 面向对象

```
package com.aurora.kotlin.classes

//抽象人类
abstract class Person(var name: String) {

    abstract fun eat()
}


/*最基础的人类是具有吃的功能，好点的人类是有笑的功能，真笑和虚伪的笑方法也不同*/

//abstract实现类的参数都不可以修改
class Man(name: String) : Person(name) {

    override fun eat() {

        println("${name}大口吃");
    }

    fun smail(userName: String): String {
        return userName
    }

    //多态
    fun smail(userName: String, flag: Boolean): String {
        return userName
    }
}

/*人类具有运动的属性 但是不是所有人类都具有 就有个接口 霍金有脑机接口*/
interface IMan {
  fun runs();
}

/*接口反应事物能力 抽象类反应本质 正常的人类就是如下*/

class Mans(name:String) :Person(name), IMan {

    override fun runs() {
        TODO("Not yet implemented")
    }

    override fun eat() {
        TODO("Not yet implemented")
    }
}
```

```
/*但是程序员developer只会吃,不会跑啊*/
class Developer(name:String):Person(name) {

    override fun eat() {

        TODO("Not yet implemented")
    }

}
```

## 委托和代理

> 委托是指把事情托付给别人或别的机构办理 代理是指以他人的名义，在授权范围内进行被代理人直接发生法律行为。代理的产生，可以受他人委托

```
interface IWashBowl {

    fun washing();
}

class Datouerzi : IWashBowl {
    override fun washing() {
        println("小头儿子洗一次碗1块钱")
    }
}

/*将洗完交给了大头儿子去做*/
class Xiaotoubaba : IWashBowl by Datouerzi() {
    override fun washing() {
        println("小头爸爸洗一次碗10块钱")
        Datouerzi().washing();
        println("儿子把碗洗完了")
    }
}

var baba = Xiaotoubaba();
baba.washing()
```

> *Return:* 小头爸爸洗一次碗10块钱
> 我是大头儿子，我在开心的洗碗，赚了1块钱
> 儿子把碗洗完了

```
class Xiaotoubaba : IWashBowl by Datouerzi() {
    override fun washing() {
        println("小头爸爸洗一次碗10块钱")
        Datouerzi().washing();
        println("儿子把碗洗完了")
    }
}
```

这里的by Datouer() 已经创建了个大头儿子这里需要修改,修改为Object默认创建

```
object Datouerzi : IWashBowl {
    override fun washing() {
        println("我是大头儿子，我在开心的洗碗，赚了1块钱")
    }
}
```

这里就不需要括号创建新的大头儿子了

```
class Xiaotoubaba : IWashBowl by Datouerzi {
    override fun washing() {
        println("小头爸爸洗一次碗10块钱")
        Datouerzi.washing();
        println("儿子把碗洗完了")
    }
}
```

## 印章类

```
//有限的子类个数
sealed class 儿子 {
    fun sayHello() {
        println("大家好")
    }

    class 小骡子() : 儿子()
    class 小驴子() : 儿子()
}

fun main() {

    var 小驴子:儿子 = 儿子.小驴子();
    var 小骡子:儿子 = 儿子.小骡子();

    val listOf = listOf<儿子>(小驴子, 小骡子)
}
```