目录

[1、Init{} 1](#_Toc140568003)

[2、一、双冒号：： 1](#_Toc140568004)

[！！ 2](#_Toc140568005)

[返回值：Unit、Nothing、any 2](#_Toc140568006)

[LaunchActivity::class.java 3](#_Toc140568007)

[Kotlin 作用域函数let run apply also with 3](#_Toc140568008)

[由参数函数的返回值决定 4](#_Toc140568009)

["${a\_et.text}"串模版 4](#_Toc140568010)

[Kotlin 类和构造参数的创建 4](#_Toc140568011)

[Kotlin集合类型 5](#_Toc140568012)

[Kotlin object和companion object 区别 6](#_Toc140568013)

[？ ：符号 6](#_Toc140568014)

[@JvmOverloads 6](#_Toc140568015)

[const val 和val的区别 6](#_Toc140568016)

[Suspend fun 7](#_Toc140568017)

[泛型实化 7](#_Toc140568018)

# 1、Init{}

是主构造函数的函数体

# 2、一、双冒号：：

1 [Kotlin](https://so.csdn.net/so/search?q=Kotlin&spm=1001.2101.3001.7020) 中 双冒号操作符 表示把一个方法当做一个参数，传递到另一个方法中进行使用，通俗的来讲就是引用一个方法。

fun main(args: Array<String>) {

println(lock("param1", "param2", ::getResult))

}

/\*\*

\* @param str1 参数1

\* @param str2 参数2

\*/

fun getResult(str1: String, str2: String): String = "result is {$str1 , $str2}"

/\*\*

\* @param p1 参数1

\* @param p2 参数2

\* @param method 方法名称

\*/

fun lock(p1: String, p2: String, method: (str1: String, str2: String) -> String): String {

return method(p1, p2)

}

Lock函数第3个参数需要一个lambda表达式，所以在调用时把getResult当作函数传入，注意要确定参数个数、类型、返回值都和其形参一致。

2 如果我们需要调用其他 Class 中的某一个方法时：

var d = Test()

println(lock("param1", "param2", d::getResult)

先实例化类再：：方法

3 为了防止作用域混淆 ， :: 调用的函数如果是类的成员函数或者是扩展函数，必须使用限定符,比如this

注：main方法不是成员函数

# ！！

!! 加在变量之后，如果对象为空，则会报异常。等同于java。

# 返回值：Unit、Nothing、any

Unit

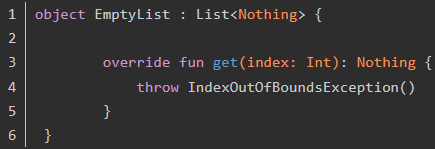
[Kotlin](https://so.csdn.net/so/search?q=Kotlin&spm=1001.2101.3001.7020)也是面向表达的语言。Kotlin中Unit类型实现了与java中void一样的功能。当一个函数没有返回值的时候，我们用Unit来表示这个特征，而不是null，大多数时候我们不需要显示地返回Unit，或者声明一个函数的返回值是Unit，编译器会推断它。

Nothing

如果一个函数返回值是Nothing，那么这个函数永远不会有返回值。

但是我们可以使用Nothing来表达一个从来不存在的返回值

例如get永远不会反回值，这个时候可以用Nothing作为get函数的返回值。



Nothing?唯一允许的值是null，可被用作任何可空类型的空引用。

Any

**Any类型就跟Java的object类型一样，是所有类的父类。**

# LaunchActivity::class.java

获取类的Kclass之后再获取类的class

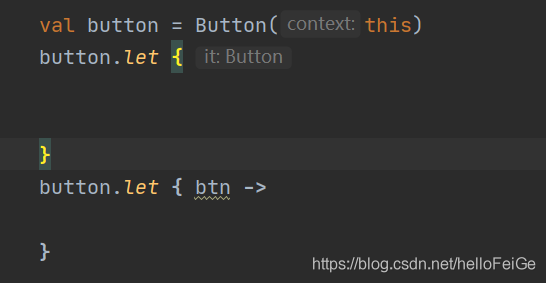
# Kotlin 作用域函数let run apply also with

当需要去定义一个变量在一个特定的作用域范围内时，就可以使用作用域函数



let把调用者T作为一个参数(默认名it可修改（见图）)传递到参数函数，返回值由参数函数的返回值决定

also把调用者T作为一个参数(默认名it可修改)传递到参数函数,返回值：this，也就是谁调用就返回谁



run是实例所在类的一个扩展函数（可用this表示调用者），无参数，返回值

由参数函数的返回值决定

apply是实例所在类的一个扩展函数（可用this表示调用者），无参数，返回值返回值：this，也就是谁调用就返回谁

# "${a\_et.text}"串模版

**串可以包含模板表达式，即：可计算的代码片段，其结果链接到串中。模板表达式以美元符号（$）开始，和简单的名字构成：**

**val i = 10**

**val s = "i = $i" // 计算结果是 "i = 10"**

或是在大括号中的任意表达式：

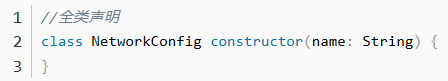
val s = "abc"

val str = "$s.length is ${s.length}" // 计算结果是 "abc.length is 3"

# Kotlin 类和构造参数的创建

kotlin中类是由class声明,kotlin的类声明有三部分, 类名,类头(指定类型参数,主构造函数),类体(花括号),类头和类体都是可选

kotlin可以拥有一个**主构造函数**和多个**次构造函数**,主构造函数是类头的一部分：它跟在类名（与可选的类型参数）后.

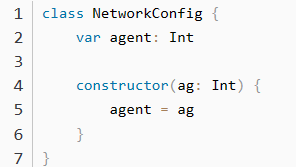


如果主构造函数没有任何注解和可见性修饰符,可以省略constructor

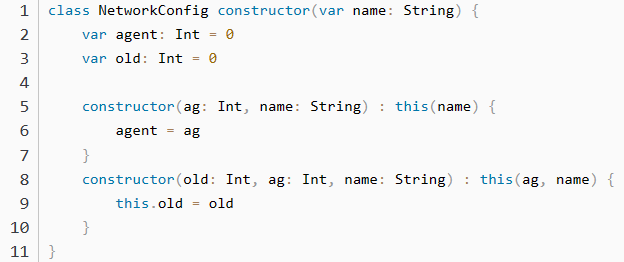


如果没有声明构造参数的类型,就是默认的val,不可修改,如果想要修改得声明var name String

可以声明由constructor的次构造函数:



如果有主构造函数,那么所有的次构造函数需要直接或者间接委托给主构造函数,  
委托到同一个类的另一个构造函数用 this 关键字即可:



* 第一执行的是主构造参数,
* 其次就是属性赋值,
* 第三就是init标识的类的属性初始化器,
* 第四就是次构造参数  
  同时因为对与this关键字的执行顺序也和java中的super一样,先执行this委托的函数

创建类



# Kotlin集合类型

Kotlin集合类型可分为可变和不可变集合类型。List、Set、Map为不可变集合类型，MutableList、MutableSet、MutableMap为可变集合类型

# Kotlin object和companion object 区别

* Object（懒加载单例类）声明（一个类）是延迟加载的，只有当第一次被访问时才会初始化，所以被用来实现单例
* companion object静态函数（伴生对象）是当包含它的类被加载时就初始化了的，这一点和Java的static还是一样的

# ？ ：符号

i == 0 ? "现在" : WeatherTimeUtils.get24Hour(data.getDate()),

i等于0返回现在，否则返回后面的表达式

# @JvmOverloads

在有默认参数值的方法中使用@JvmOverloads注解，则Kotlin就会暴露多个重载方法。

## kotlin 实现静态的方式

在 kotin 语言中其实没有 java [static](https://so.csdn.net/so/search?q=static&spm=1001.2101.3001.7020) 的这个概念，基本都是用一个静态对象来模拟 class 的静态属性和方法，目前有4种实现方式：

* [companion object](https://blog.csdn.net/fitaotao/article/details/112763100?ops_request_misc=%7B%22request%5Fid%22%3A%22167807388216782425149658%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=167807388216782425149658&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-112763100-null-null.142%5ev73%5epc_new_rank,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=kotlin%20%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&spm=1018.2226.3001.4187) - 伴随对象，声明单例的方式（最常用）
* [@JvmField + @JvmStatic 注解](https://blog.csdn.net/fitaotao/article/details/112763100?ops_request_misc=%7B%22request%5Fid%22%3A%22167807388216782425149658%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=167807388216782425149658&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-112763100-null-null.142%5ev73%5epc_new_rank,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=kotlin%20%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&spm=1018.2226.3001.4187) - 使用注解标签声明 static 的部分
* [object 单例](https://blog.csdn.net/fitaotao/article/details/112763100?ops_request_misc=%7B%22request%5Fid%22%3A%22167807388216782425149658%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=167807388216782425149658&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-112763100-null-null.142%5ev73%5epc_new_rank,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=kotlin%20%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&spm=1018.2226.3001.4187) - 静态单例其实和 companion object 类似
* [const](https://blog.csdn.net/fitaotao/article/details/112763100?ops_request_misc=%7B%22request%5Fid%22%3A%22167807388216782425149658%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=167807388216782425149658&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-112763100-null-null.142%5ev73%5epc_new_rank,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=kotlin%20%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&spm=1018.2226.3001.4187) - 包内唯一性，脱离类的束缚，kotlin 的特性，在 java 中会编译生成一个 kotlin.kt 的文件专门对齐提供支持

# const val 和val的区别

const val相当于public final static

val相当于provite final static

前者可以直接调用，后者需要使用getter来调用

创建常量的时候尽量使用前者，避免频繁的函数调用。

# 泛型实化

两个前提：

1、使用inline修饰方法

2、使用reified修饰泛型

# 协程

# Suspend fun

挂起函数、协程

# coroutineScope函数

创建一个协程作用域

# Async函数

在async函数中发起网络请求再调用await（）方法就可以实现只有在两个网络请求都成功响应后才会进一步执行程序。

另外由于只能在协程作用域中才能调用，所以写在coroutineScope内。