# 范围函数介绍

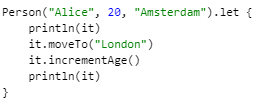
官网教程连接：<https://kotlinlang.org/docs/reference/scope-functions.html>，截图如下:



Kotlin提供多种函数可以在一个给定对象的上下文中执行代码块：let，run，with，apply和also，Kotlin标准库中包含了这几个函数，其唯一目的是在对象的上下文中执行代码块。当您在提供了lambda表达式的对象上调用此类函数时，它会形成一个临时范围。在此范围内，您可以在不使用其名称的情况下访问该对象。这些函数称为范围函数。共有5个：let，run，with，apply，和also。

基本上，这些函数做同样的事情：在对象上执行代码块。不同之处在于这个对象在块内如何使用以及整个表达式的结果是什么。

以下是范围函数的典型用法：

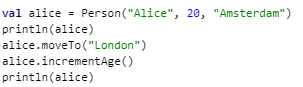


输出结果如下：

Person(name=Alice, age=20, city=Amsterdam)

Person(name=Alice, age=21, city=London)

如果你没使用let，你将不得不引入一个新的变量，并在你使用它时重复它的名字。



范围函数不会引入任何新的技术功能，但它们可以使您的代码更简洁和可读。

由于范围函数的功用相似，为您的情况选择正确的函数可能有点棘手。选择主要取决于您的意图和项目使用的一致性。下面我们将详细介绍范围函数与其使用约定之间的区别。

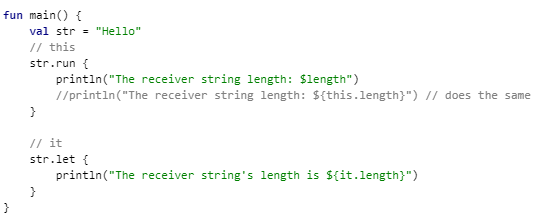
# 范围函数的区别

由于范围函数本质上非常相似，因此理解它们之间的差异非常重要。每个范围函数有两个主要区别：

* 引用上下文对象的方式
* 返回值

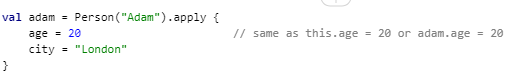
## 上下文对象：this或it

在范围函数的lambda内部，上下文对象可通过一个简短引来代替它的实际名称。每个范围函数使用两种方法之一来访问上下文对象：作为lambda [接收器](https://kotlinlang.org/docs/reference/lambdas.html#function-literals-with-receiver)（this）或作为lambda参数（it）。两者都提供相同的功能，因此我们将针对不同的情况描述每种功能的优缺点，并提供有关其使用的建议。



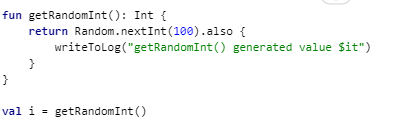
**this**

run，with和apply将上下文对象作为lambda接收器 - 通过关键字this。因此，在他们的lambda中，该对象可用于普通类函数中。在大多数情况下，您在访问接收器对象的成员时可以省略this，从而缩短代码。另一方面，如果省略this，则可能难以区分属性或函数是接收器对象的还是外部对象的。因此，建议将上下文对象作为receiver（this）用于主要在对象上执行操作：调用其函数或赋值属性。

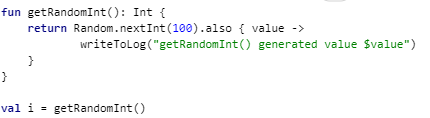


**it**

反过来，let和also将上下文对象作为lambda参数。如果未指定参数名称，则通过隐式默认名称it访问该对象。it比this表达更短并且it通常更容易阅读。但是，在调用对象函数或属性时，您没有隐式的对象可用（即不能直接调用对象的方法和属性）。因此，当对象主要用作函数调用中的参数时it作为上下文对象更好。it如果在代码块中使用多个变量，也会更好。



此外，将上下文对象作为参数传递时，可以为作用域内的上下文对象提供自定义名称



## 返回值

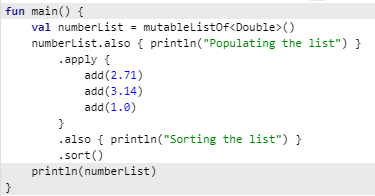
范围函数返回的结果不同：

* apply和also返回上下文对象。
* let，run和with返回lambda结果。

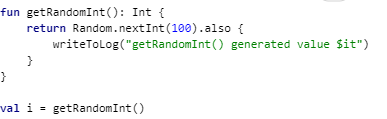
因此，我们可以根据下一步操作选择正确的函数。

**上下文对象**

apply和also返回上下文对象本身，因此，它们可以作为辅助步骤包含在调用链中：您可以继续在它们之后的同一对象上链接函数调用。

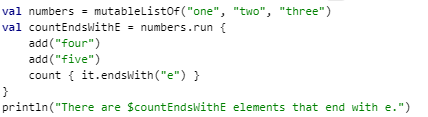


它们也可以在返回上下文对象的函数的return语句中使用。



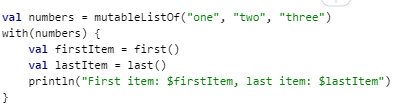
**Lambda结果**

let，run并with返回lambda结果。因此，您可以在将结果分配给变量，对结果进行链接操作等时使用它们。



打印结果为：There are 3 elements that end with e.

此外，您可以忽略返回值并使用范围函数为变量创建临时范围。



打印结果为：First item: one, last item: three

# 范围函数的使用说明

为了帮助您为您的案例选择正确的范围函数，我们将详细描述它们并提供使用建议。从技术上讲，函数在许多情况下是可互换的，因此示例显示了定义常见使用方式的约定。

## let

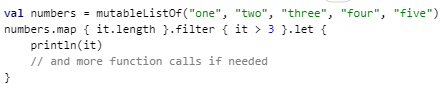
**上下文对象**可用作参数（it）。**返回值**是lambda结果。

let可用于在调用链的结果上调用一个或多个函数。例如，以下代码打印集合上的两个操作的结果：



打印结果：[5, 4, 4]

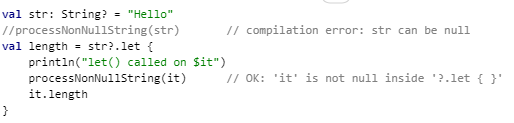
有了let，你可以重写它：



如果代码块包含it作为参数的单个函数，则可以使用方法reference（::）而不是lambda：

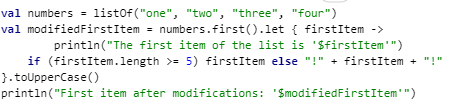


let通常用于仅使用非空值执行代码块。要对非null对象执行操作，请对其使用安全调用运算符?.，和let使用其lambda中的操作进行调用。



输出结果为：let() called on Hello

let的另一种使用情况是引入局限变量，以提高代码可读性。要为上下文对象定义新变量，请将其名称作为lambda参数提供，以便可以使用它而不是默认值it。



输出结果为：

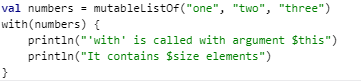
The first item of the list is 'one'

First item after modifications: '!ONE!'

## with

非扩展函数：**上下文对象**作为参数传递，但在lambda中，它可用作receiver（this）。**返回值**是lambda结果。

我们建议with在不提供lambda结果的情况下调用上下文对象上的函数。在代码中，with可以读作“ *使用此对象，执行以下操作。*”

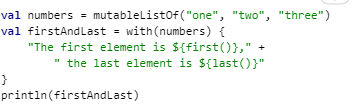


打印结果如下 ：

'with' is called with argument [one, two, three]

It contains 3 elements

另一个用例with是引入一个辅助对象，其属性或函数将用于计算值。



打印结果为：The first element is one, the last element is three

## run

**上下文对象**可用作receiver（this）。**返回值**是lambda结果。

run与with做同样的事情，但是调用为let – 作为上下文对象的一个扩展函数。

当lambda包含对象初始化和返回值的计算时run 很有用。

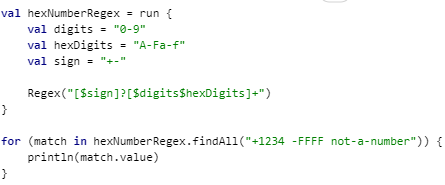


打印结果为：

Result for query 'Default request to port 8080'

Result for query 'Default request to port 8080'

除了在接收器对象上调用run外，您还可以将其用作非扩展功能。非扩展run允许您执行需要表达式的多个语句块。



打印结果如下：

+1234

-FFFF

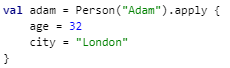
-a

be

## apply

**上下文对象**可用作receiver（this）。**返回值**是对象本身。

使用apply代码块没有return语句，主要是在接收器对象的成员上操作。常见的情况apply是对象配置。此类调用可以读作“ 将以下赋值应用于对象。”



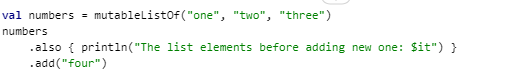
将接收器作为返回值，您可以轻松地将其包含apply到调用链中以进行更复杂的处理。

## also

**上下文对象**可用作参数（it）。**返回值**是对象本身。

also适用于执行将上下文对象作为参数的某些操作。使用also了不改变对象的附加动作，如记录或打印的调试信息。通常，您可以在also不破坏程序逻辑的情况下从调用链中删除调用。

当您also在代码中看到时，您可以将其读作“ 并执行以下操作 ”。



# 范围函数的选择

为了帮助您选择适合您的范围功能，我们提供了它们之间的主要差异表。



以下是根据预期目的选择范围功能的简短指南：

* 在非null对象上执行lambda： let
* 将表达式作为局部范围中的变量引入： let
* 对象配置： apply
* 对象配置和计算结果： run
* 运行需要表达式的语句：非扩展名 run
* 附加效果： also
* 对对象进行分组函数调用： with

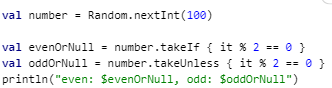
不同功能的用例重叠，因此您可以根据项目或团队中使用的特定约定选择功能。

虽然范围函数是使代码更简洁的一种方法，但是要避免过度使用它们：它会降低代码的可读性并导致错误。避免嵌套范围函数，并在链接它们时要小心：很容易对当前上下文对象和this或it 的值感到困惑。

# takeIf 和 takeUnless

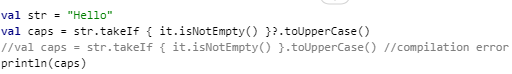
除范围函数外，标准库还包含函数takeIf和takeUnless。这些函数允许您在调用链中嵌入对象状态的检查。

在具有提供谓词的对象上调用时，如果该对象与谓词takeIf匹配，则返回该对象。否则，它返回null。因此，takeIf是单个对象的过滤功能。反过来，takeUnless如果对象与谓词不匹配则返回该对象，如果匹配则返回该null。该对象可用作lambda参数（it）。



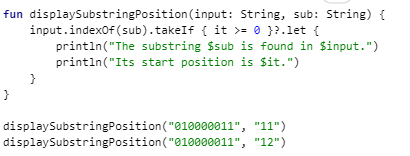
打印结果为：even: null, odd: 63

在takeIf和takeUnless之后链接其他函数时，不要忘记执行空检查或安全调用（?.），因为它们的返回值可以为空。



打印结果：HELLO

takeIf和takeUnless与范围函数一起使用尤其有用。一个很好的例子就是将它们链接起来，let以便在与给定谓词匹配的对象上运行代码块。要执行此操作，请调用takeIf该对象，然后let使用安全调用（?.）进行调用。对于与谓词不匹配的对象，takeIf返回null并且let不会被调用。

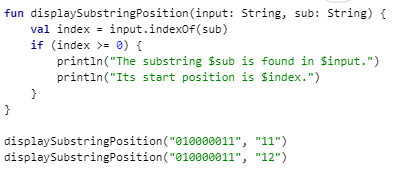


打印结果如下：

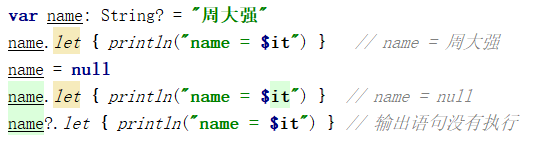
The substring 11 is found in 010000011.

Its start position is 7.

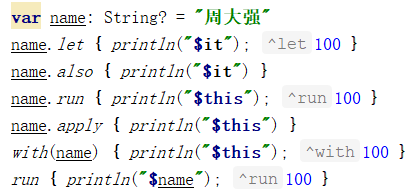
这是没有标准库函数的相同函数的样子：



# 总结



如果没使用?.进行调用，with、run、apply、also跟上面的情况一样，不管对象是否为空，都会执行代码块中的代码，如果使用了?. ，则只有当对象不为null时才执行代码块中的代码，需要注意的是with没法使用?.操作符。



5个范围函数功能都差不多，区别如下：

一、按this、it分：

1. 如果你想使用it代表对象或者起一个名字代表对象，则有两个选择：let和also  
   给对象起别名，可以加类型，也可不加，如下：  
     
   
2. 如果你想使用this代表对象，则有三个选择：run、apply、with

二、按返回值分：

1、如果要返回任意类型，有三个选择：let、run、with

2、如果要返回对象自身，有两个选择：also、apply  
 这个方式是比较好记忆的，因为also和apply这两个单词比较像，都以a字母开头，以这个点进行记忆，能返回自身的函数就是also、apply，自然另外的三个函数即为返回任意类型的：let、run、with

三、综合以上两点做选择

1、如果想以it代表对象，又想返回对象自身，只有一个选择：also

2、如果想以it代表对象，又想返回任意类型，只有一个选择：let

3、如果想以this代表对象，又想返回对象自身，只有一个选择：apply

4、如果想以this代表对象，又想返回任意类型，有两个选择：run、with，嗯？功能重叠？确实重叠了，它们的不同点是，run可以使用?.操作符，而with不可以。所以可以不使用with。

5、如果不需要返回值，则不要选返回对象自身的函数，选择其它函数，当你的代码块最后一行不是表达式时，将返回Unit，即空的意思

返回对象自身或者返回任意类型，这个选择大家都能做出来，根据你的真实需求选择即可。

选it还是选this代表对象呢？如果使用this，则每次调用对方的属性或方法需要this.的方式，当然也可以不写，但是如果不写直接调用对象的属性或方法，则有时会分不清这属性或方法是哪来的？会不会是定义在外部类中的呢？这很容易搞不清，所以最好还是加上this，但是用it也一样啊，it更短，所以it比较好。 如果只是对类做一些初始化代码，没有别的逻辑，其实用this会比较好。

其实做出选择也很简单，就4种情况嘛，但是因为Kotlin起的名字不好，不容易区分容易记混了功能，如果让我来起函数名就好了，如下：

itThis it代表对象，返回对象自己

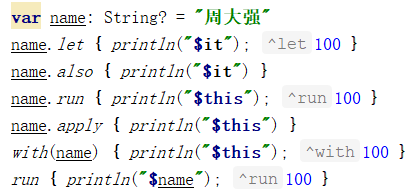
itAny it代表对象，返回任意类型

thisThis this代表对象，返回对象自己

thisAny this代表对象，返回任意类型

虽然说名字长了一点，但是我们一般有代码提示的敲几个字母直接回车就行，也不用全部自己输入，名字长一点，但是使用效率高了就是值得的。

或者可以这样，保存下图：



每次做选择的时候，打开这张图片，要想使用it，一看这图就知道只有let和also可选，要想返回任意类型，一看就只有let、run、with，So Easy，就是每次要打开图片看一下比较麻烦一点，但用的次数多了就记住了！！如果你不去分清不去记，那你可能会经常用某一两个，但是其实有些情况用其它的会更好！

# 范围函数的链式调用



打印结果为：null

实验证明：第1个let使用了?.操作符，只对第1个的let的代码块起作用，后面的let没有使用?.，所以虽然person为null，但是后面的let代码块还是被执行了。

所以总结就是，如果对象为null，想要后面的链式范围函数都不执行，则所有调用范围函数的时候都需要使用?.操作符，如下：



这样，只要person为null，两个let函数中的代码都不会被执行

来一行实战代码：

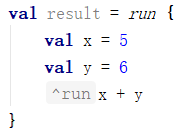


使用let是因为可以返回任意类型，这里返回的是CameraPreview，接着使用also是因为要返回CameraPreview自己本身，这样的代码读起来是有点不好理解，但是也要读得懂，万一别人就这么用呢，如果你需要维护，则必须要读读得懂。

# 使用场景举例

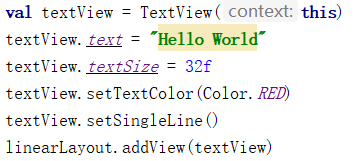
## 1、常用：let、also、apply

## 2、run-隔离代码

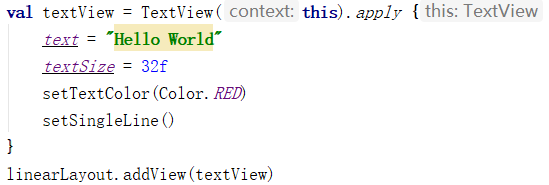


这个情况为把某几行只用于计算一个结果的代码使用run代码块进行封装，这样在读代码的时候我们就很容易分清，代码块里的那些代码就是为了计算出最后的结果的，这样的代码与其它的代码进行了隔离，其它代码就不会受这个代码块的污染，代码块一直行完毕，外面就没法访问x、y了，是否会立即回收x、y变量的内存呢？不得而知，但这样的代码块分工明确，读起来就比较轻松。

## 3、apply-初始化类



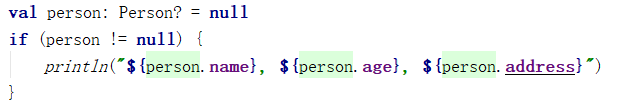
使用apply改写，如下：



这里对TextView的初始化更加清晰。为什么选apply，因为要返回对象自身，可以选also、apply，同时，这种对类的初始化，肯定是连this都不想写啊，所以当然要选apply了，如果选了also是要写it的，省略it是会报错的

## 4、let-只想执行一个代码块

调用一个对象的属性或方法可以使用it或this，使用this的时候可以省略this，但是不写this又容易搞不清楚属性或方法是哪里定义的，所以还是要写this，但写this不如写it简短，所以选择let，如下：



使用let改写，如下：



如果Person写好了toString方法，我们要打印它的所有属性，打印toString是比较好的方法，代码可以简写为：

打印person对象，且不为null时才打印：



## 5、官方的推荐

* 在非null对象上执行lambda： let
* 将表达式作为局部范围中的变量引入： let
* 对象配置： apply
* 对象配置和计算结果： run
* 运行需要表达式的语句：非扩展名 run
* 附加效果： also
* 对对象进行分组函数调用： with