Lambda表达式：

- lambda表达式总是被大括号括着

- 其参数（如果有的话）在->之前声明（参数类型可以省略）

- 函数体（如果存在的话）在->后面

在Kotlin中有一个约定，如果函数的最后一个参数是一个函数，并且传递了一个lambda表达式作为相应的参数，我们可以在圆括号外指定它：

**fun** <T, R> List<T>.map(transform: (T) -> R): List<R> {

**val** result = arrayListOf<R>()

**for** (item **in** **this**)

result.add(transform(item))

**return** result

}

该函数可以如下调用:

**val** doubled = ints.map { value -> value \* 2 }

It：单个参数的隐式名称

如果函数字面值只有一个参数，那么它的声明可以省略（连同->），其名称是it

ints.map { it \* 2 }

这些约定可以写[LINQ-风格](http://msdn.microsoft.com/en-us/library/bb308959.aspx)的代码:

strings.filter { it.length == 5 }.sortedBy { it }.map { it.toUpperCase() }

### 下划线用于未使用的变量（自 1.1 起）

如果 lambda 表达式的参数未使用，那么可以用下划线取代其名称：

map.forEach { \_, value -> println("$value!") }

## Lambda 表达式与匿名函数

一个 lambda 表达式或匿名函数是一个“函数字面值”，即一个未声明的函数， 但立即做为表达式传递。考虑下面的例子：

max(strings, { a, b -> a.length < b.length })

函数 max 是一个高阶函数，换句话说它接受一个函数作为第二个参数。 其第二个参数是一个表达式，它本身是一个函数，即函数字面值。写成函数的话，它相当于：

**fun** compare(a: String, b: String): Boolean = a.length < b.length

### 函数类型

对于接受另一个函数作为参数的函数，我们必须为该参数指定函数类型。 例如上述函数 max 定义如下：

**fun** <T> max(collection: Collection<T>, less: (T, T) -> Boolean): T? {

**var** max: T? = **null**

**for** (it **in** collection)

**if** (max == **null** || less(max, it))

max = it

**return** max

}

参数 less 的类型是 (T, T) -> Boolean，即一个接受两个类型T的参数并返回一个布尔值的函数： 如果第一个参数小于第二个那么该函数返回 true。

在上面第 4 行代码中，less 作为一个函数使用：通过传入两个 T 类型的参数来调用。

如上所写的是就函数类型，或者可以有命名参数，如果你想文档化每个参数的含义的话。

**val** compare: (x: T, y: T) -> Int = ……

如要声明一个函数类型的可空变量，请将整个函数类型括在括号中并在其后加上问号：

**var** sum: ((Int, Int) -> Int)? = **null**

Lambda表达式语法：

Lambda表达式的完整语法形式，即函数类型的字面值如下：

**val** sum = { x: Int, y: Int -> x + y }

Lambda表达式总是被大括号括着，完整语法形式的参数声明放在大括号内，并有可选的类型标注，函数体跟在一个->符号之后。入骨推断出的该lambda的返回不是Unit，那么该lanbda主题中的最后一个表达式会视为返回值

如果把所有可选标注留下：

**val** sum: (Int, Int) -> Int = { x, y -> x + y }

一个lambda表达式只有一个参数是很常见的，Kotlin允许我们不声明唯一的参数，并且将隐含地为我们声明其名称为it：

ints.filter { it > 0 } // 这个字面值是“(it: Int) -> Boolean”类型的

我们可以使用[限定的返回](http://www.kotlincn.net/docs/reference/returns.html" \l "标签处返回)语法从 lambda 显式返回一个值。否则，将隐式返回最后一个表达式的值。因此，以下两个片段是等价的：

ints.filter {

**val** shouldFilter = it > 0

shouldFilter

}

ints.filter {

**val** shouldFilter = it > 0

**return**@filter shouldFilter

}

请注意，如果一个函数接受另一个函数作为最后一个参数，lambda 表达式参数可以在圆括号参数列表之外传递。

[函数](http://www.kotlincn.net/docs/reference/functions.html)

Lambda 表达式

[内联函数](http://www.kotlincn.net/docs/reference/inline-functions.html)

[协程](http://www.kotlincn.net/docs/reference/coroutines.html)

其他

核心库

参考

Java 互操作

JavaScript

工具

常见问题

* [完整 Kotlin 参考（PDF）](http://www.kotlincn.net/docs/kotlin-docs.pdf)
* [完整 Kotlin 参考（字大PDF）](https://www.gitbook.com/download/pdf/book/hltj/kotlin-reference-chinese)
* [完整 Kotlin 参考（ePUB）](https://www.gitbook.com/download/epub/book/hltj/kotlin-reference-chinese)
* [完整 Kotlin 参考（Mobi）](https://www.gitbook.com/download/mobi/book/hltj/kotlin-reference-chinese)

[Edit Page](https://github.com/hltj/kotlin-web-site-cn/edit/master/pages/docs/reference/lambdas.md" \o "Edit this page on GitHub" \t "http://www.kotlincn.net/docs/reference/_blank)

# 高阶函数和 lambda 表达式

## 高阶函数

高阶函数是将函数用作参数或返回值的函数。 这种函数的一个很好的例子是 lock()，它接受一个锁对象和一个函数，获取锁，运行函数并释放锁：

**fun** <T> lock(lock: Lock, body: () -> T): T {

lock.lock()

**try** {

**return** body()

}

**finally** {

lock.unlock()

}

}

让我们来检查上面的代码：body 拥有[函数类型](http://www.kotlincn.net/docs/reference/lambdas.html" \l "函数类型)：() -> T， 所以它应该是一个不带参数并且返回 T 类型值的函数。 它在 try-代码块内部调用、被 lock 保护，其结果由lock（）函数返回。

如果我们想调用 lock() 函数，我们可以把另一个函数传给它作为参数（参见[函数引用](http://www.kotlincn.net/docs/reference/reflection.html" \l "函数引用)）：

**fun** toBeSynchronized() = sharedResource.operation()

**val** result = lock(lock, ::toBeSynchronized)

通常会更方便的另一种方式是传一个 [lambda 表达式](http://www.kotlincn.net/docs/reference/lambdas.html" \l "lambda-表达式与匿名函数)：

**val** result = lock(lock, { sharedResource.operation() })

Lambda 表达式在[下文会有更详细的](http://www.kotlincn.net/docs/reference/lambdas.html" \l "lambda-表达式与匿名函数)描述，但为了继续这一段，让我们看一个简短的概述：

* lambda 表达式总是被大括号括着；
* 其参数（如果有的话）在 -> 之前声明（参数类型可以省略）；
* 函数体（如果存在的话）在 -> 后面。

在 Kotlin 中有一个约定，如果函数的最后一个参数是一个函数，并且你传递一个 lambda 表达式作为相应的参数，你可以在圆括号之外指定它：

lock (lock) {

sharedResource.operation()

}

高阶函数的另一个例子是 map()：

**fun** <T, R> List<T>.map(transform: (T) -> R): List<R> {

**val** result = arrayListOf<R>()

**for** (item **in** **this**)

result.add(transform(item))

**return** result

}

该函数可以如下调用:

**val** doubled = ints.map { value -> value \* 2 }

请注意，如果 lambda 是该调用的唯一参数，则调用中的圆括号可以完全省略。

### it：单个参数的隐式名称

另一个有用的约定是，如果函数字面值只有一个参数， 那么它的声明可以省略（连同 ->），其名称是 it。

ints.map { it \* 2 }

这些约定可以写[LINQ-风格](http://msdn.microsoft.com/en-us/library/bb308959.aspx)的代码:

strings.filter { it.length == 5 }.sortedBy { it }.map { it.toUpperCase() }

### 下划线用于未使用的变量（自 1.1 起）

如果 lambda 表达式的参数未使用，那么可以用下划线取代其名称：

map.forEach { \_, value -> println("$value!") }

### 在 lambda 表达式中解构（自 1.1 起）

在 lambda 表达式中解构是作为[解构声明](http://www.kotlincn.net/docs/reference/multi-declarations.html" \l "在-lambda-表达式中解构自-11-起)的一部分描述的。

## 内联函数

使用[内联函数](http://www.kotlincn.net/docs/reference/inline-functions.html)有时能提高高阶函数的性能。

## Lambda 表达式与匿名函数

一个 lambda 表达式或匿名函数是一个“函数字面值”，即一个未声明的函数， 但立即做为表达式传递。考虑下面的例子：

max(strings, { a, b -> a.length < b.length })

函数 max 是一个高阶函数，换句话说它接受一个函数作为第二个参数。 其第二个参数是一个表达式，它本身是一个函数，即函数字面值。写成函数的话，它相当于：

**fun** compare(a: String, b: String): Boolean = a.length < b.length

### 函数类型

对于接受另一个函数作为参数的函数，我们必须为该参数指定函数类型。 例如上述函数 max 定义如下：

**fun** <T> max(collection: Collection<T>, less: (T, T) -> Boolean): T? {

**var** max: T? = **null**

**for** (it **in** collection)

**if** (max == **null** || less(max, it))

max = it

**return** max

}

参数 less 的类型是 (T, T) -> Boolean，即一个接受两个类型T的参数并返回一个布尔值的函数： 如果第一个参数小于第二个那么该函数返回 true。

在上面第 4 行代码中，less 作为一个函数使用：通过传入两个 T 类型的参数来调用。

如上所写的是就函数类型，或者可以有命名参数，如果你想文档化每个参数的含义的话。

**val** compare: (x: T, y: T) -> Int = ……

如要声明一个函数类型的可空变量，请将整个函数类型括在括号中并在其后加上问号：

**var** sum: ((Int, Int) -> Int)? = **null**

### Lambda 表达式语法

Lambda 表达式的完整语法形式，即函数类型的字面值如下：

**val** sum = { x: Int, y: Int -> x + y }

lambda 表达式总是被大括号括着， 完整语法形式的参数声明放在大括号内，并有可选的类型标注， 函数体跟在一个 -> 符号之后。如果推断出的该 lambda 的返回类型不是 Unit，那么该 lambda 主体中的最后一个（或可能是单个）表达式会视为返回值。

如果我们把所有可选标注都留下，看起来如下：

**val** sum: (Int, Int) -> Int = { x, y -> x + y }

一个 lambda 表达式只有一个参数是很常见的。 如果 Kotlin 可以自己计算出签名，它允许我们不声明唯一的参数，并且将隐含地为我们声明其名称为 it：

ints.filter { it > 0 } // 这个字面值是“(it: Int) -> Boolean”类型的

我们可以使用[限定的返回](http://www.kotlincn.net/docs/reference/returns.html" \l "标签处返回)语法从 lambda 显式返回一个值。否则，将隐式返回最后一个表达式的值。因此，以下两个片段是等价的：

ints.filter {

**val** shouldFilter = it > 0

shouldFilter

}

ints.filter {

**val** shouldFilter = it > 0

**return**@filter shouldFilter

}

请注意，如果一个函数接受另一个函数作为最后一个参数，lambda 表达式参数可以在圆括号参数列表之外传递。 参见 [callSuffix](http://www.kotlincn.net/docs/reference/grammar.html" \l "callSuffix) 的语法。

### 匿名函数

上面提供的 lambda 表达式语法缺少的一个东西是指定函数的返回类型的能力。在大多数情况下，这是不必要的。因为返回类型可以自动推断出来。然而，如果确实需要显式指定，可以使用另一种语法： 匿名函数 。

**fun**(x: Int, y: Int): Int = x + y

匿名函数看起来非常像一个常规函数声明，除了其名称省略了。其函数体可以是表达式（如上所示）或代码块：

**fun**(x: Int, y: Int): Int {

**return** x + y

}

参数和返回类型的指定方式与常规函数相同，除了能够从上下文推断出的参数类型可以省略：

ints.filter(**fun**(item) = item > 0)

匿名函数的返回类型推断机制与正常函数一样：对于具有表达式函数体的匿名函数将自动推断返回类型，而具有代码块函数体的返回类型必须显式指定（或者已假定为 Unit）。

请注意，匿名函数参数总是在括号内传递。 允许将函数留在圆括号外的简写语法仅适用于 lambda 表达式。

Lambda表达式与匿名函数之间的另一个区别是[非局部返回](http://www.kotlincn.net/docs/reference/inline-functions.html" \l "非局部返回)的行为。一个不带标签的 return 语句总是在用 fun 关键字声明的函数中返回。这意味着 lambda 表达式中的 return 将从包含它的函数返回，而匿名函数中的 return 将从匿名函数自身返回。

。。。。。

待定继续学习。。。。