


# 使用 Kotlin 构建 Android 应用 | Kotlin 迁移指南 (上篇)



谷歌开发者   
已认证的官方帐号

已关注

11 人赞同了该文章

今年五月份的 Google I/O 上，我们正式向全球宣布 Kotlin-first 的这一重要理念，Kotlin 将成为 Android 开发者的首选语言。接下来的几周我们将会为大家连载关于 Kotlin 迁移指南的系列文章，包含 Kotlin 的优势和介绍 (上篇)、迁移到 Kotlin (中篇)，以及使用 Kotlin 的常见问题 (下篇)，帮助开发者们顺利迁移并开始使用 Kotlin 构建 Android 应用。

## 了解 Kotlin，以及使用它的优势

Kotlin 是一种现代的静态设置类型编程语言，可以提高开发者的工作效率，并提升开发者的工作愉悦度。

### 优势 1: 可与 Java 互操作

与 Android SDK 和 Java 程序语言库兼容，Kotlin 代码中可以方便调用 Java 库 (Android Studio 的 Lint 检查亦能与 Kotlin 代码互操作)。

- Kotlin 互操作指南:[developer.android.google.cn...](https://developer.android.google.cn/kotlin)

### 优势 2: 与 IDE 工具兼容

Kotlin 语言由 IntelliJ 的开发团队设计，可与 IntelliJ (以及 Android Studio) 完美搭配使用，Android Studio 为 Kotlin 提供了一流的支持，比如，您可通过内置工具来将 Java 代码转换成 Kotlin 代码。或者借助 “Show Kotlin Bytecode” 工具，您可以在学习 Kotlin 时查看等效的 Java 代码。

### 优势 3: 空安全检测

默认情况下，Kotlin 可避免空指针异常发生。而且可以在开发时而不是运行时发现和避免错误。

```
fun foo(p: Int) { ... }  
foo(null) // 编译器报错  
  
var o: String? = ...  
println(o.toLowerCase()) // 编译器报错
```

△ 上面两个例子都会触发编译器报错，从而避免了在运行时出现崩溃

### 优势 4: 更简洁的代码

Kotlin 有着更简洁明了的语法，可减少样板代码的使用。

```
// Java 语言类代码  
public class User {  
  
    private String firstName;  
    private String lastName;  
  
    public User(String firstName, String lastName) {...}  
    public String getFirstName() {...}
```

```
public void setLastName(String lastName) {...}
}
```



比如上例中的数据类代码，有字段以及对应的 getter 和 setter 方法，虽然都是常规内容，但不免繁琐，而且大量的样本代码也会占用开发者的精力。我们来看看同样的类用 Kotlin 如何编写：

```
// Kotlin 语言，同样的类代码
class User(
    var firstName: String?,
    var lastName: String?
)
```

**Kotlin 还支持扩展方法**，可以给现有的类附加新的方法 (而不需要修改类的原始代码)。比如我们想计算字符串内某个字符出现的次数，通常我们这么做：

```
// 定义方法
fun howMany(string: String, char: Char): Int {
    var count = 0
    val lowerCaseLetter = char.toLowerCase()
    for (i in 0 until string.length) {
        if (lowerCaseLetter == string[i].toLowerCase()) count++
    }
    return count
}

// 计算“Elephant”里有几个“e”
val string = "Elephant"
howMany(string, 'e')
```

有了扩展方法，我们可以直接把 howMany 这个方法添加至 String 类：

```
// 扩展方法
fun String.howMany(char: Char): Int {
    var count = 0
    val lowerCaseLetter = char.toLowerCase()
    for (i in 0 until length) {
        if (lowerCaseLetter == this[i].toLowerCase()) count++
    }
    return count
}

// 执行
val string = "Elephant"
string.howMany('e')
```

如此一来，我们就直接“问”string“你里面有几个‘e’字符”就可以了，这更简洁、自然，可读性也大幅提升。

**Kotlin 还支持指定/默认参数**，这让开发者在编写方法时，不需要为不同参数的版本另写一个方法，而是直接在同一个方法里，通过“?”标出可空参数，通过“=”给出参数的默认值即可。

```
// View.java
public View(Context context, @Nullable AttributeSet attrs) {
    this(context, attrs, 0);
}

public View(Context context, @Nullable AttributeSet attrs, int defStyleAttr) {
    this(context, attrs, defStyleAttr, 0);
}
```

```
public View(Context context, @Nullable AttributeSet attrs, int defStyleAttr, int defStyleRes) {
    // ...
}

// 和上述内容等效的 Kotlin 代码
class View(context: Context, attrs: AttributeSet?, defStyleAttr: Int = 0, defStyleRes: Int = 0) {
    // ...
}
```

△ 使用 Kotlin 仅需要定义一个构造函数即可

### 优势 5: 语言特性带来的进阶功能

Kotlin 也在持续为开发者带来更多高级的语言特性，协程就是一个突出的例子。

Kotlin 里的协程可以理解为从语言级别实现了异步或非阻塞编程，并在 Kotlin 1.3 中开始提供，在 Android 上使用协程可以避免下面的问题：

- 通过主（界面）线程进行调用时可以确保安全（比如在主线程中异步访问数据库）
- 避免在主线程上运行耗时较长的任务（如图像或网络操作）时发生阻塞

比如下面这个例子，使用协程时不会对主线程造成阻塞，并可提高可读性：

```
// 使用回调
fun getData() {
    get("developer.android.google.cn") { result ->
        show(result)
    }
}

// 使用协程
suspend fun getData() {
    val result = get("developer.android.google.cn")
    show(result)
}

suspend fun get(url: String) {...}
```

## 使用 Kotlin 构建 Android 应用

△ Kotlin 推进的时间表

使用 Kotlin 更快速地编写更棒的 Android 应用，自两年前 Android 平台开始支持使用 Kotlin 语言后，我们一直在努力丰富使用 Kotlin 构建的体验和开发效率的提升。我们为 Android 开发者提供了 Android KTX、Android Studio 的支持以及大量的学习资源等。

### Android KTX

自从两年前 Android 平台开始支持 Kotlin 后，我们一直在努力解决 Kotlin 的兼容性问题并丰富其功能，更进一步为大家带来了许多工具来进一步提高开发效率，比如 Android KTX。它是一组适用于 Android 平台开发的扩展，提供对多种常用操作，比如开发者和提供简化的封装，如：



## KTX: 动画

AnimatorKt 能让开发者在动画的各个阶段执行自己的操作。比如以前需要在动画结束时执行操作需要这么做:

```
// Animator API
fun addListener(listener: Animator.AnimatorListener!)

// 应用代码
val animator = ObjectAnimator.ofFloat(...)
anim.addListener(object : AnimatorListenerAdapter() {
    override fun onAnimationEnd(animation: Animator?) {
        println("end!")
    }
})
```

而在 AnimatorKt 里, 只需使用 doOnEnd 即可, 代码被精简成了一行:

```
// AnimatorKt
inline fun Animator.doOnEnd(
    crossinline action: (animator: Animator) -> Unit
)

// 应用代码
val animator = ObjectAnimator.ofFloat(...)
anim.doOnEnd { println("end!") }
```

大家可以参看如下代码了解 AnimatorKt 是如何帮大家精简代码的:

```
inline fun Animator.doOnEnd(crossinline action: (animator: Animator) -> Unit) =
    addListener(onEnd = action)

inline fun Animator.addListener(
    crossinline onEnd: (animator: Animator) -> Unit = {},
    crossinline onStart: (animator: Animator) -> Unit = {},
    crossinline onCancel: (animator: Animator) -> Unit = {},
    crossinline onRepeat: (animator: Animator) -> Unit = {}
): Animator.AnimatorListener {
    val listener = object : Animator.AnimatorListener {
        override fun onAnimationRepeat(animator: Animator) = onRepeat(animator)
        override fun onAnimationEnd(animator: Animator) = onEnd(animator)
        override fun onAnimationCancel(animator: Animator) = onCancel(animator)
        override fun onAnimationStart(animator: Animator) = onStart(animator)
    }
    addListener(listener)
    return listener
}
```

## KTX: Drawables 转化为位图

将可绘制对象转化为位图是不少开发者在处理 UI 时的常用操作, 在以前需要如此操作:

```
// 位图 API
fun createBitmap(width: Int, height: Int, config: Bitmap.Config): Bitmap

// Canvas API
```



```
// 应用代码
val (oldLeft, oldTop, oldRight, oldBottom) = bounds
drawable.setBounds(0, 0, width, height)
val bitmap = Bitmap.createBitmap(width, height, Config.ARGB_8888)
drawable.draw(Canvas(bitmap))
drawable.setBounds(oldLeft, oldTop, oldRight, oldBottom)
```

但如果使用 DrawableKt，只需要如下操作即可，应用代码再次被压缩成了一行：

```
// DrawableKt
fun toBitmap(
    width: Int = intrinsicWidth,
    height: Int = intrinsicHeight,
    config: Config? = null
): Bitmap

// 应用代码
d.toBitmap(width, height)
```

DrawableKt 实际上是使用扩展方法，将开发者需要做的操作封装了起来，从而节省了大量重复工作的时间：

```
fun Drawable.toBitmap(
    @Px width: Int = intrinsicWidth,
    @Px height: Int = intrinsicHeight,
    config: Config? = null
): Bitmap {
    if (this is BitmapDrawable) {
        if (config == null || bitmap.config == config) {
            if (width == intrinsicWidth && height == intrinsicHeight) {
                return bitmap
            }
            return Bitmap.createScaledBitmap(bitmap, width, height, true)
        }
    }
}

val (oldLeft, oldTop, oldRight, oldBottom) = bounds

val bitmap = Bitmap.createBitmap(width, height, config ?: Config.ARGB_8888)
setBounds(0, 0, width, height)
draw(Canvas(bitmap))

setBounds(oldLeft, oldTop, oldRight, oldBottom)
return bitmap
}
```

## Kotlin x Jetpack

在推荐开发者使用 Kotlin 构建应用的同时，Android 团队自己也在大规模的使用 Kotlin，比如下面要跟大家介绍的在 Jetpack 库中的 Kotlin 特性的使用：

### Jetpack 与协程

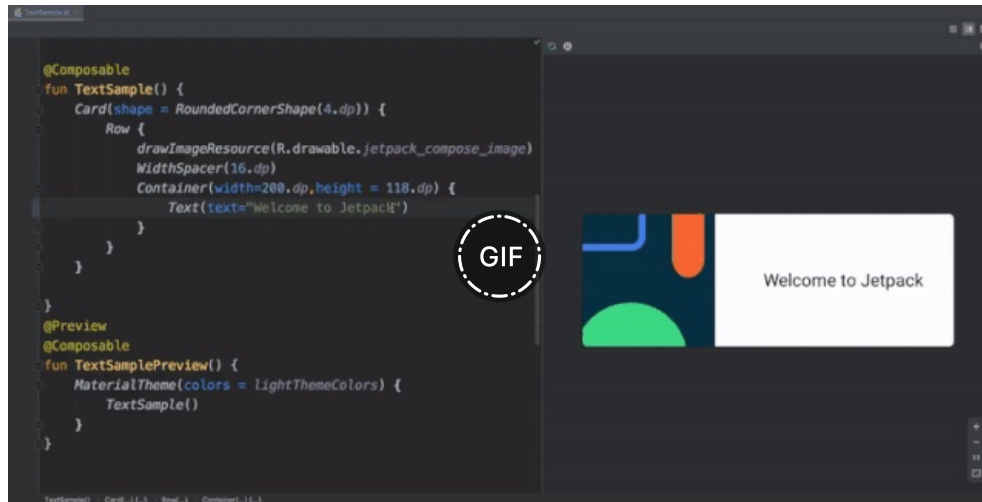
在 Jetpack 的下述组件库里使用了协程的特性：

- Room: suspend 函数

- ViewModel: 协程作用域
- LiveData: 协程构建器 (coroutine builder)



## Jetpack Compose



在上周举办的 Android Dev Summit 2019 大会上，我们发布了 Jetpack Compose 的开发者预览版。Jetpack Compose 可以帮助开发者简化并加速 Android 上的 UI 开发——使用更少的代码、强大的工具和非常直观的 Kotlin API，使您的应用栩栩如生。

我们为开发者们准备了一些 Jetpack Compose 相关的教程，帮助您更直观的体验和了解它的优势：[developer.android.google.cn...](https://developer.android.google.cn/develop/compose)

请持续关注我们接下来时间发布的与 Kotlin 迁移指南相关的文章。

如果您对在 Android 开发中使用 Kotlin 有任何疑问或者想法，欢迎在评论区和我们分享。

[点击这里](#)即刻使用 Kotlin 打造精彩 Android 应用



发布于 2019-10-31

[Android 应用](#) [Kotlin](#) [Android](#)

推荐阅读

Kotlin] 发糖了!

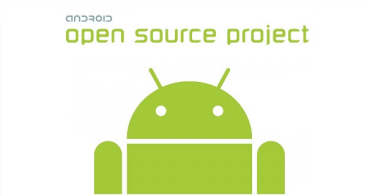
这次不太监了，保证不太监了，我保证！（因为只有一篇在一段时间内摸索中，我在自己的项目里创造并使用了很多的语法糖和库。很多时候这些Kotlin自己没有内置的小东西可以给你的开发带来不少...

发布于Mario...



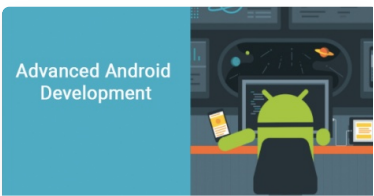
Gradle更小、更快构建APP的奇淫技巧

终端研发部



有没有必要阅读ANDROID源码

发表于中二病也要...



Google 高级 Android 开发课程发布

发表于极光日报

2 条评论

切换为时间排序

写下你的评论...

Goooler 2019-10-31  
第一个例子里面的 int 应该大写  
赞

沃德曼 2019-11-02  
这kotlin越看越像C#  
赞