# Android开发中使用Kotlin

## （1） Kotlin环境配置

参考：

1. <https://zhuanlan.zhihu.com/p/70224521>
2. <https://zhuanlan.zhihu.com/p/48805971>

3】 <https://developer.android.com/kotlin/add-kotlin>

### 【1】 安装Kotlin插件

Android Studio 从 3.0（preview）版本开始将内置 Kotlin 插件。

如果你的环境还没有安装Kotlin，那么打开 Settings ( Mac 为 Preferences) 面板，在右侧找到 Plugins 选项 ，搜索框输入 "Kotlin" 查找，然后安装即可，安装完成之后需要重启 Android Studio。

图形用户界面, 应用程序

描述已自动生成

### 【2】 工程设置

#### 《1》 Kotlin Module

新建工程，在工程向导中选择Kotlin语言，然后就可以直接开始Kotlin开发了。

#### 《2》 在现有Java工程中添加Kotlin支持

如果你想在现有Java工程中添加Kotlin支持，也将非常简单，步骤如下：

1. 选择Project视图

2. 在工程根目录build.gradle中的buildscript节点添加kotlin版本号：

|  |
| --- |
| buildscript {  ext.kotlin\_version = 'xx'  ...  } |

这里xx要和kotlin plugin中相同。

3. 在工程根目录build.gradle中的buildscript节点的dependencies子节点添加一行依赖：

|  |
| --- |
| buildscript {  dependencies {  // 添加这一行依赖  classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin\_version"  }  } |

4. 在项目module的 build.gradle 的顶部添加以下代码：

|  |
| --- |
| apply plugin: 'kotlin-android'  apply plugin: 'kotlin-android-extensions' |

或

|  |
| --- |
| plugins **{** id **'kotlin-android'** id **'kotlin-android-extensions' }** |

其中：

* 'kotlin-android'

<https://developer.android.com/kotlin/add-kotlin#add>

必须。

这个是在AS中能用kt开发所必须的Kotlin Android plugin。

* 'kotlin-android-extensions'

非必须。

用这个主要是为了替代findViewById。

具体见下“Kotlin Android Extensions”。

5. 在项目module的 build.gradle 的dependencies节点里添加以下代码：

|  |
| --- |
| dependencies {  implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin\_version"  } |

注：不同版本的依赖库

1.2.0以前使用这个依赖库

implementation "org.jetbrains.kotlin:kotlin-stdlib-jre8:$kotlin\_version"

1.2.0+版本使用这个依赖库

implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin\_version"

依赖库名称含义：

* implementation “org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin\_version”：1.2.0版本以前使用的配置，兼容java7
* implementation “org.jetbrains.kotlin:kotlin-stdlib-jre8:$kotlin\_version”：1.2.0以前使用的配置，兼容java8
* implementation “org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin\_version”1.2.0+版本以前使用的配置，兼容java7
* implementation “org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin\_version”1.2.0+版本使用的配置，兼容java8

## （2） Java和kotlin混合开发

参考：

1. <https://zhuanlan.zhihu.com/p/48805971>

<https://www.jianshu.com/p/9220227cdfb3>（优先看，两者同一篇）

### 【1】 Java转Kotlin

<https://zhuanlan.zhihu.com/p/70224521>

1. java代码复制进.kt文件
2. 右键.java->**Convert Java File To Kotlin File**

### 【2】 Kotlin转Java

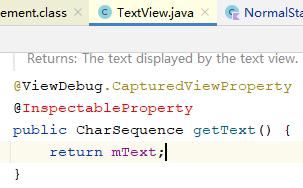
<https://www.jianshu.com/p/cde0282ddf11>

### 【3】 Kotlin调java

实例：



statement.appStatement.text点进去：



### 【x】 java调Kotlin

### 【x】 同个module同时使用Java和Kotlin

参考：<https://zhuanlan.zhihu.com/p/48805971>

#### 《1》 给kotlin文件单独配置文件路径

在一个module里面同时使用kotlin和java两种语言做开发。 为了便于区分，需要在该module的build.gradle文件做配置。

默认情况，Kotlin文件存放在src/main/java/文件夹下。若是需要隔开java文件和Kotlin文件，我们可以指定一个专门存放Kotlin文件的文件夹：

首先在src/main目录下创建文件夹kotlin，然后做以下配置：

|  |
| --- |
| android {  // 配置一个路径专门存放kotlin文件  sourceSets {  main.java.srcDirs += 'src/main/kotlin'  } |

如下

图形用户界面, 文本, 应用程序

描述已自动生成

#### 《2》 代码里使用

##### 1） 若主工程是kotlin，部分需Java写

##### 2） 若主工程是java，部分需kotlin写

###### 1》 模板

比如Activity是Java写的，工具类是kotlin写的，实例代码如下：

*MainActivity：*

|  |
| --- |
| **package** com.example.kotlintest1;  **import** androidx.appcompat.app.AppCompatActivity;  **import** android.content.Context; **import** android.os.Bundle; **import** android.widget.Toast;  **import** liyanghui.ResourcesUtils;  **public class** MainActivity **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main***);  init(**this**);  }   **private void** init(Context context) {  **int** statusBarHeight = ResourcesUtils.***Companion***.getStatusBarHeight(context);  Toast.*makeText*(context, **"状态栏高度为："** + statusBarHeight + **"像素"**,Toast.***LENGTH\_SHORT***).show();  } } |

*ResourcesUtils：*

|  |
| --- |
| **package** liyanghui  **import** android.content.Context  */\*\*  \* 描述：  \* 资源工具类  \*/* **class** ResourcesUtils {   *// 伴生对象 这里面包裹的方法 实际上类似于java的静态方法* **companion object** {  */\*\*  \* 获取状态栏高度  \** ***@return*** *状态栏高度  \*/* **fun** getStatusBarHeight(context: Context): Int {  *//获取status\_bar\_height资源的ID* **val** resourceId = context.*resources*.getIdentifier(**"status\_bar\_height"**, **"dimen"**, **"android"**)  **return if** (resourceId > 0) {  *//根据资源ID获取响应的尺寸值* context.*resources*.getDimensionPixelSize(resourceId)  } **else** {  dip2px(context, 40f)  }  }   */\*\*  \* dp 转 px(像素)  \*/* **fun** dip2px(context: Context, dpValue: Float?): Int {  **val** scale = context.*resources*.*displayMetrics*.**density  return** (dpValue!! \* scale + 0.5f).toInt()  }   */\*\*  \* px(像素) 的单位 转 dp  \*/* **fun** px2dip(context: Context, pxValue: Float?): Int {  **val** scale = context.*resources*.*displayMetrics*.**density  return** (pxValue!! / scale + 0.5f).toInt()  }   }  } |

项目结构：

图形用户界面, 文本, 应用程序

描述已自动生成

###### 2》 注意点

1》 不要在kotlin包下定义和java包下相同 包名.类名 的文件

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

# Android KTX

<https://developer.android.com/kotlin/ktx>

# Kotlin Android Extensions

## 是什么

<https://android-developers.googleblog.com/2020/11/the-future-of-kotlin-android-extensions.html>

The [Android Kotlin Extensions Gradle plugin](https://plugins.gradle.org/plugin/org.jetbrains.kotlin.android.extensions) (not to be confused with [Android KTX](https://developer.android.com/kotlin/ktx)) was released in 2017 and brought two new conveniences to Android development in Kotlin:

* **Synthetics** let you replace calls to findViewById with kotlinx.android.synthetic bindings.
* **Parcelize**allows you to remove boilerplate and easily create [Parcelables](https://developer.android.com/reference/android/os/Parcelable) through the @Parcelize annotation.

## 使用

### 在AS中配置

<https://blog.csdn.net/qq_43616001/article/details/114809745>

<https://blog.csdn.net/qq_35762397/article/details/109537079>

<https://android-developers.googleblog.com/2020/11/the-future-of-kotlin-android-extensions.html> 

在module的build.gradle顶部添加kotlin-android-extensions插件，语法为：

图形用户界面, 文本

描述已自动生成

或

文本

描述已自动生成

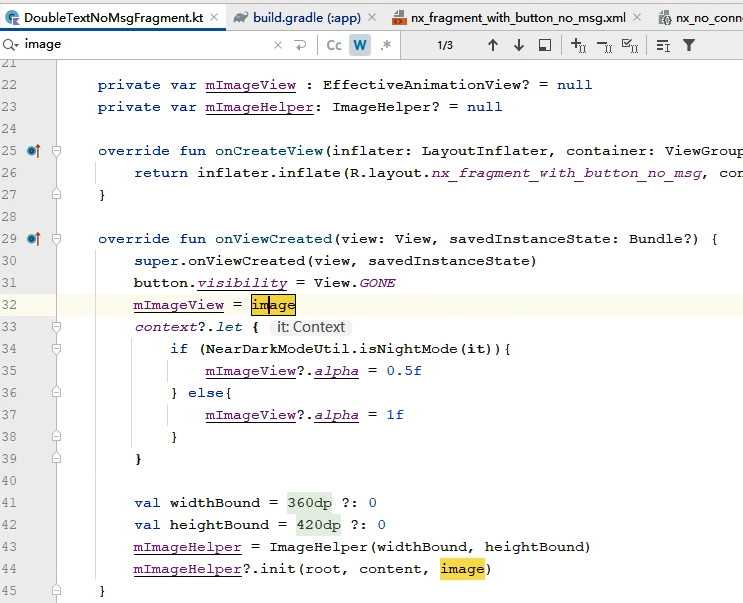
### 在代码中使用

<https://juejin.cn/post/6844903741053730823>

## 实例

### 例1：

Uikitdemo-状态与流程-空白页



这里的image指向



### 例2：

两个layout

图形用户界面, 文本

描述已自动生成

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

然后同时引入两个layout synthetic

图形用户界面, 文本, 应用程序

描述已自动生成

编译报错

图形用户界面, 文本, 应用程序

描述已自动生成

把其中一个注释解决

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 缺点

<https://android-developers.googleblog.com/2020/11/the-future-of-kotlin-android-extensions.html>

Since then, we have released [View Binding for Android](https://developer.android.com/topic/libraries/view-binding), an officially supported library that has deep integration with the Android build toolchain and provides similar functionality as Kotlin synthetics. While we continue to recommend Parcelize, a number of drawbacks have appeared with using Kotlin synthetics:

* They pollute the global namespace
* They don’t expose nullability information
* They only work in Kotlin code

## 已deprecated

<https://android-developers.googleblog.com/2020/11/the-future-of-kotlin-android-extensions.html>