创建项目

首先创建一个新项目，选择“Empty Activity”。

将项目命名为“CameraXApp”，软件包名称更改为“com.android.example.cameraxapp”。选择Kotlin语言开发，设定最低支持的API Level 21（CameraX 所需的最低级别）。

添加 Gradle 依赖

打开项目的模块（Module）的build.gradle 文件，并添加 CameraX 依赖项：

dependencies {

def camerax\_version = "1.1.0-beta01"

implementation "androidx.camera:camera-core:${camerax\_version}"

implementation "androidx.camera:camera-camera2:${camerax\_version}"

implementation "androidx.camera:camera-lifecycle:${camerax\_version}"

implementation "androidx.camera:camera-video:${camerax\_version}"

implementation "androidx.camera:camera-view:${camerax\_version}"

implementation "androidx.camera:camera-extensions:${camerax\_version}"

}

1

2

3

4

5

6

7

8

9

10

CameraX 需要一些属于 Java 8 的方法，因此需要相应地设置编译选项（实际上比较新的Android Studio版本会默认设置）。在 android 代码块的末尾，紧跟在 buildTypes 之后，添加以下代码：

compileOptions {

sourceCompatibility JavaVersion.VERSION\_1\_8

targetCompatibility JavaVersion.VERSION\_1\_8

}

1

2

3

4

因为在项目中使用了ViewBinding，在 android{} 代码块末尾添加如下代码：

buildFeatures {

viewBinding true

}

1

2

3

左侧出现提示时（灯泡状），点击 Sync Now，于是便可以在应用中使用 CameraX 了。

创建项目布局

本项目中，涉及以下的功能：

CameraX PreviewView（用于预览相机图片/视频）。

用于控制图片拍摄的标准按钮。

用于开始/停止视频拍摄的标准按钮。

用于放置 2 个按钮的垂直指南。

打开res/layout/activity\_main.xml 的 activity\_main 布局文件，并将其替换为以下代码。

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<androidx.camera.view.PreviewView

android:id="@+id/viewFinder"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

<Button

android:id="@+id/image\_capture\_button"

android:layout\_width="110dp"

android:layout\_height="110dp"

android:layout\_marginBottom="50dp"

android:layout\_marginEnd="50dp"

android:elevation="2dp"

android:text="@string/take\_photo"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintEnd\_toStartOf="@id/vertical\_centerline" />

<Button

android:id="@+id/video\_capture\_button"

android:layout\_width="110dp"

android:layout\_height="110dp"

android:layout\_marginBottom="50dp"

android:layout\_marginStart="50dp"

android:elevation="2dp"

android:text="@string/start\_capture"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintStart\_toEndOf="@id/vertical\_centerline" />

<androidx.constraintlayout.widget.Guideline

android:id="@+id/vertical\_centerline"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical"

app:layout\_constraintGuide\_percent=".50" />

</androidx.constraintlayout.widget.ConstraintLayout>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

更新res/values/strings.xml 文件

<resources>

<string name="app\_name">CameraXApp</string>

<string name="take\_photo">Take Photo</string>

<string name="start\_capture">Start Capture</string>

<string name="stop\_capture">Stop Capture</string>

</resources>

1

2

3

4

5

6

布局界面效果如图

编写 MainActivity.kt 代码

将 MainActivity.kt 中的代码替换为以下代码，但保留软件包名称不变。它包含 import 语句、将要实例化的变量、要实现的函数以及常量。

系统已实现 onCreate()，供我们检查相机权限、启动相机、为照片和拍摄按钮设置 onClickListener()，以及实现 cameraExecutor。虽然系统已经实现 onCreate()，但在实现文件中的方法之前，相机将无法正常工作。

package com.android.example.cameraxapp

import android.Manifest

import android.content.ContentValues

import android.content.pm.PackageManager

import android.os.Build

import android.os.Bundle

import android.provider.MediaStore

import androidx.appcompat.app.AppCompatActivity

import androidx.camera.core.ImageCapture

import androidx.camera.video.Recorder

import androidx.camera.video.Recording

import androidx.camera.video.VideoCapture

import androidx.core.app.ActivityCompat

import androidx.core.content.ContextCompat

import com.android.example.cameraxapp.databinding.ActivityMainBinding

import java.util.concurrent.ExecutorService

import java.util.concurrent.Executors

import android.widget.Toast

import androidx.camera.lifecycle.ProcessCameraProvider

import androidx.camera.core.Preview

import androidx.camera.core.CameraSelector

import android.util.Log

import androidx.camera.core.ImageAnalysis

import androidx.camera.core.ImageCaptureException

import androidx.camera.core.ImageProxy

import androidx.camera.video.FallbackStrategy

import androidx.camera.video.MediaStoreOutputOptions

import androidx.camera.video.Quality

import androidx.camera.video.QualitySelector

import androidx.camera.video.VideoRecordEvent

import androidx.core.content.PermissionChecker

import java.nio.ByteBuffer

import java.text.SimpleDateFormat

import java.util.Locale

typealias LumaListener = (luma: Double) -> Unit

class MainActivity : AppCompatActivity() {

private lateinit var viewBinding: ActivityMainBinding

private var imageCapture: ImageCapture? = null

private var videoCapture: VideoCapture<Recorder>? = null

private var recording: Recording? = null

private lateinit var cameraExecutor: ExecutorService

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

viewBinding = ActivityMainBinding.inflate(layoutInflater)

setContentView(viewBinding.root)

// Request camera permissions

if (allPermissionsGranted()) {

startCamera()

} else {

ActivityCompat.requestPermissions(

this, REQUIRED\_PERMISSIONS, REQUEST\_CODE\_PERMISSIONS)

}

// Set up the listeners for take photo and video capture buttons

viewBinding.imageCaptureButton.setOnClickListener { takePhoto() }

viewBinding.videoCaptureButton.setOnClickListener { captureVideo() }

cameraExecutor = Executors.newSingleThreadExecutor()

}

private fun takePhoto() {}

private fun captureVideo() {}

private fun startCamera() {}

private fun allPermissionsGranted() = REQUIRED\_PERMISSIONS.all {

ContextCompat.checkSelfPermission(

baseContext, it) == PackageManager.PERMISSION\_GRANTED

}

override fun onDestroy() {

super.onDestroy()

cameraExecutor.shutdown()

}

companion object {

private const val TAG = "CameraXApp"

private const val FILENAME\_FORMAT = "yyyy-MM-dd-HH-mm-ss-SSS"

private const val REQUEST\_CODE\_PERMISSIONS = 10

private val REQUIRED\_PERMISSIONS =

mutableListOf (

Manifest.permission.CAMERA,

Manifest.permission.RECORD\_AUDIO

).apply {

if (Build.VERSION.SDK\_INT <= Build.VERSION\_CODES.P) {

add(Manifest.permission.WRITE\_EXTERNAL\_STORAGE)

}

}.toTypedArray()

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

请求必要的权限

应用需要获得用户授权才能打开相机；录制音频也需要麦克风权限；在 Android 9 § 及更低版本上，MediaStore 需要外部存储空间写入权限。在此步骤中，我们将实现这些必要的权限。

打开 AndroidManifest.xml，然后将以下代码行添加到 application 标记之前。

<uses-feature android:name="android.hardware.camera.any" />

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"

android:maxSdkVersion="28" />

1

2

3

4

5

添加 android.hardware.camera.any 可确保设备配有相机。指定 .any 表示它可以是前置摄像头，也可以是后置摄像头。

如果在未指定 .any 的情况下使用 android.hardware.camera，并且设备未配有后置摄像头（例如大多数 Chromebook），那么相机将无法正常运行。第二行代码会添加对该相机的使用权限。

然后，复制代码到MainActivity.kt. 中。

override fun onRequestPermissionsResult(

requestCode: Int, permissions: Array<String>, grantResults:

IntArray) {

if (requestCode == REQUEST\_CODE\_PERMISSIONS) {

if (allPermissionsGranted()) {

startCamera()

} else {

Toast.makeText(this,

"Permissions not granted by the user.",

Toast.LENGTH\_SHORT).show()

finish()

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

其中代码

if (requestCode == REQUEST\_CODE\_PERMISSIONS) {

}

1

2

3

检查请求代码是否正确；否则，忽略它。代码

if (allPermissionsGranted()) {

startCamera()

}

1

2

3

判断是否已授予权限，若是则调用 startCamera() （进行预览）；如果未授予权限，系统会显示一个消息框，通知用户未授予权限。

else {

Toast.makeText(this,

"Permissions not granted by the user.",

Toast.LENGTH\_SHORT).show()

finish()

}

1

2

3

4

5

6

运行应用，可发现应用程序请求使用摄像头和麦克风：

实现 Preview 用例

在相机应用中，取景器用于让用户预览他们拍摄的照片。我们将使用 CameraX Preview 类实现取景器。

如需使用 Preview，首先需要定义一个配置，然后系统会使用该配置创建用例的实例。生成的实例就是绑定到 CameraX 生命周期的内容。填充之前的startCamera() 函数

private fun startCamera() {

val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

cameraProviderFuture.addListener({

// Used to bind the lifecycle of cameras to the lifecycle owner

val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

// Preview

val preview = Preview.Builder()

.build()

.also {

it.setSurfaceProvider(viewBinding.viewFinder.surfaceProvider)

}

// Select back camera as a default

val cameraSelector = CameraSelector.DEFAULT\_BACK\_CAMERA

try {

// Unbind use cases before rebinding

cameraProvider.unbindAll()

// Bind use cases to camera

cameraProvider.bindToLifecycle(

this, cameraSelector, preview)

} catch(exc: Exception) {

Log.e(TAG, "Use case binding failed", exc)

}

}, ContextCompat.getMainExecutor(this))

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

以下分析上述代码。创建 ProcessCameraProvider 的实例。这用于将相机的生命周期绑定到生命周期所有者。这消除了打开和关闭相机的任务，因为 CameraX 具有生命周期感知能力。

val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

1

向 cameraProviderFuture 添加监听器。添加 Runnable 作为一个参数。添加 ContextCompat.getMainExecutor() 作为第二个参数。这将返回一个在主线程上运行的 Executor。其具体形式：

cameraProviderFuture.addListener(Runnable {}, ContextCompat.getMainExecutor(this))

1

在 Runnable 中，添加 ProcessCameraProvider。它用于将相机的生命周期绑定到应用进程中的 LifecycleOwner。

val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

1

初始化 Preview 对象，在其上调用 build，从取景器中获取 Surface 提供程序，然后在预览上进行设置。

val preview = Preview.Builder()

.build()

.also {

it.setSurfaceProvider(viewBinding.viewFinder.surfaceProvider)

}

1

2

3

4

5

创建 CameraSelector 对象，然后选择 DEFAULT\_BACK\_CAMERA。

val cameraSelector = CameraSelector.DEFAULT\_BACK\_CAMERA

1

创建一个 try 代码块。在此块内，确保没有任何内容绑定到 cameraProvider，然后将 cameraSelector 和预览对象绑定到 cameraProvider。

try {

cameraProvider.unbindAll()

cameraProvider.bindToLifecycle(

this, cameraSelector, preview)

}

1

2

3

4

5

有多种原因可能会导致此代码失败，例如应用不再获得焦点。将此代码封装到 catch 块中，以便在出现故障时记录日志。

catch(exc: Exception) {

Log.e(TAG, "Use case binding failed", exc)

}

1

2

3

最后，运行应用，可以看到相机预览

实现 ImageCapture 用例（拍照功能）

其他用例与 Preview 非常相似。首先，定义一个配置对象，该对象用于实例化实际用例对象。若要拍摄照片，需要实现 takePhoto() 方法，该方法会在用户按下 photo 按钮时调用。填充takePhoto() 方法的代码：

private fun takePhoto() {

// Get a stable reference of the modifiable image capture use case

val imageCapture = imageCapture ?: return

// Create time stamped name and MediaStore entry.

val name = SimpleDateFormat(FILENAME\_FORMAT, Locale.US)

.format(System.currentTimeMillis())

val contentValues = ContentValues().apply {

put(MediaStore.MediaColumns.DISPLAY\_NAME, name)

put(MediaStore.MediaColumns.MIME\_TYPE, "image/jpeg")

if(Build.VERSION.SDK\_INT > Build.VERSION\_CODES.P) {

put(MediaStore.Images.Media.RELATIVE\_PATH, "Pictures/CameraX-Image")

}

}

// Create output options object which contains file + metadata

val outputOptions = ImageCapture.OutputFileOptions

.Builder(contentResolver,

MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI,

contentValues)

.build()

// Set up image capture listener, which is triggered after photo has

// been taken

imageCapture.takePicture(

outputOptions,

ContextCompat.getMainExecutor(this),

object : ImageCapture.OnImageSavedCallback {

override fun onError(exc: ImageCaptureException) {

Log.e(TAG, "Photo capture failed: ${exc.message}", exc)

}

override fun

onImageSaved(output: ImageCapture.OutputFileResults){

val msg = "Photo capture succeeded: ${output.savedUri}"

Toast.makeText(baseContext, msg, Toast.LENGTH\_SHORT).show()

Log.d(TAG, msg)

}

}

)

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

首先，获取对 ImageCapture 用例的引用。如果用例为 null，退出函数。如果在设置图片拍摄之前点按“photo”按钮，它将为 null。如果没有 return 语句，应用会在该用例为 null 时崩溃。

val imageCapture = imageCapture ?: return

1

接下来，创建用于保存图片的 MediaStore 内容值。使用时间戳，确保 MediaStore 中的显示名是唯一的。

val name = SimpleDateFormat(FILENAME\_FORMAT, Locale.US)

.format(System.currentTimeMillis())

val contentValues = ContentValues().apply {

put(MediaStore.MediaColumns.DISPLAY\_NAME, name)

put(MediaStore.MediaColumns.MIME\_TYPE, "image/jpeg")

if(Build.VERSION.SDK\_INT > Build.VERSION\_CODES.P) {

put(MediaStore.Images.Media.RELATIVE\_PATH, "Pictures/CameraX-Image")

}

}

1

2

3

4

5

6

7

8

9

创建一个 OutputFileOptions 对象。在该对象中，可以指定所需的输出内容。我们希望将输出保存在 MediaStore 中，以便其他应用可以显示它，因此，添加 MediaStore 条目。

val outputOptions = ImageCapture.OutputFileOptions

.Builder(contentResolver,

MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI,

contentValues)

.build()

1

2

3

4

5

对 imageCapture 对象调用 takePicture()。传入 outputOptions、执行器和保存图片时使用的回调。接下来，需要完成回调。

imageCapture.takePicture(

outputOptions, ContextCompat.getMainExecutor(this),

object : ImageCapture.OnImageSavedCallback {}

)

1

2

3

4

如果图片拍摄失败或保存图片失败，添加错误情况以记录失败。

override fun onError(exc: ImageCaptureException) {

Log.e(TAG, "Photo capture failed: ${exc.message}", exc)

}

1

2

3

如果拍摄未失败，即表示照片拍摄成功！将照片保存到之前创建的文件中，显示消息框，让用户知道照片已拍摄成功，并输出日志语句。

override fun onImageSaved(output: ImageCapture.OutputFileResults) {

val savedUri = Uri.fromFile(photoFile)

val msg = "Photo capture succeeded: $savedUri"

Toast.makeText(baseContext, msg, Toast.LENGTH\_SHORT).show()

Log.d(TAG, msg)

}

1

2

3

4

5

6

找到 startCamera() 方法，并将此代码复制到要预览的代码下方。

imageCapture = ImageCapture.Builder().build()

1

最后，在 try 代码块中更新对 bindToLifecycle() 的调用，以包含新的用例：

cameraProvider.bindToLifecycle(

this, cameraSelector, preview, imageCapture)

1

2

此时，该方法将如下所示：

private fun startCamera() {

val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

cameraProviderFuture.addListener({

// Used to bind the lifecycle of cameras to the lifecycle owner

val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

// Preview

val preview = Preview.Builder()

.build()

.also {

it.setSurfaceProvider(viewFinder.surfaceProvider)

}

imageCapture = ImageCapture.Builder()

.build()

// Select back camera as a default

val cameraSelector = CameraSelector.DEFAULT\_BACK\_CAMERA

try {

// Unbind use cases before rebinding

cameraProvider.unbindAll()

// Bind use cases to camera

cameraProvider.bindToLifecycle(

this, cameraSelector, preview, imageCapture)

} catch(exc: Exception) {

Log.e(TAG, "Use case binding failed", exc)

}

}, ContextCompat.getMainExecutor(this))

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

重新运行应用，然后按 Take Photo。屏幕上应该会显示一个消息框，会在日志中看到一条消息。

这时可以查看本地的图片库，查看刚刚拍摄的图片。

实现 ImageAnalysis 用例

需要Android 10 以及更高版本的设备，建议使用实体设备来测试这部分代码

使用 ImageAnalysis 功能可让相机应用变得更加有趣。它允许定义实现 ImageAnalysis.Analyzer 接口的自定义类，并使用传入的相机帧调用该类。无需管理相机会话状态，甚至无需处理图像；与其他生命周期感知型组件一样，仅绑定到应用所需的生命周期就足够了。

将此分析器添加为 MainActivity.kt 中的内部类。分析器会记录图像的平均亮度。如需创建分析器，我们会替换实现 ImageAnalysis.Analyzer 接口的类中的 analyze 函数。

private class LuminosityAnalyzer(private val listener: LumaListener) : ImageAnalysis.Analyzer {

private fun ByteBuffer.toByteArray(): ByteArray {

rewind() // Rewind the buffer to zero

val data = ByteArray(remaining())

get(data) // Copy the buffer into a byte array

return data // Return the byte array

}

override fun analyze(image: ImageProxy) {

val buffer = image.planes[0].buffer

val data = buffer.toByteArray()

val pixels = data.map { it.toInt() and 0xFF }

val luma = pixels.average()

listener(luma)

image.close()

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

在类中实现 ImageAnalysis.Analyzer 接口后，只需在 ImageAnalysis中实例化一个 LuminosityAnalyzer 实例（与其他用例类似），并再次更新 startCamera() 函数，然后调用 CameraX.bindToLifecycle() 即可：

接下来更新startCamera()，将以下代码添加到 imageCapture 代码下方。

val imageAnalyzer = ImageAnalysis.Builder()

.build()

.also {

it.setAnalyzer(cameraExecutor, LuminosityAnalyzer { luma ->

Log.d(TAG, "Average luminosity: $luma")

})

}

1

2

3

4

5

6

7

更新 cameraProvider 上的 bindToLifecycle() 调用，以包含 imageAnalyzer。

cameraProvider.bindToLifecycle(

this, cameraSelector, preview, imageCapture, imageAnalyzer)

1

2

现在，完整的方法将如下所示：

private fun startCamera() {

val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

cameraProviderFuture.addListener({

// Used to bind the lifecycle of cameras to the lifecycle owner

val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

// Preview

val preview = Preview.Builder()

.build()

.also {

it.setSurfaceProvider(viewBinding.viewFinder.surfaceProvider)

}

imageCapture = ImageCapture.Builder()

.build()

val imageAnalyzer = ImageAnalysis.Builder()

.build()

.also {

it.setAnalyzer(cameraExecutor, LuminosityAnalyzer { luma ->

Log.d(TAG, "Average luminosity: $luma")

})

}

// Select back camera as a default

val cameraSelector = CameraSelector.DEFAULT\_BACK\_CAMERA

try {

// Unbind use cases before rebinding

cameraProvider.unbindAll()

// Bind use cases to camera

cameraProvider.bindToLifecycle(

this, cameraSelector, preview, imageCapture, imageAnalyzer)

} catch(exc: Exception) {

Log.e(TAG, "Use case binding failed", exc)

}

}, ContextCompat.getMainExecutor(this))

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

立即运行应用！它会大约每秒在 logcat 中生成一个类似于下面的消息。

D/CameraXApp: Average luminosity: ...

1

实现 VideoCapture 用例（拍摄视频）

VideoCapture 用例是 CameraX 用例系列的新成员，建议使用实体设备进行测试

CameraX 在 1.1.0-alpha10 版中添加了 VideoCapture 用例，并且从那以后一直在改进。注意，VideoCapture API 支持很多视频捕获功能，因此，为了使此项目易于管理，仅演示如何在 MediaStore 中捕获视频和音频。

将以下代码复制到captureVideo() 方法：该方法可以控制 VideoCapture 用例的启动和停止。

// Implements VideoCapture use case, including start and stop capturing.

private fun captureVideo() {

val videoCapture = this.videoCapture ?: return

viewBinding.videoCaptureButton.isEnabled = false

val curRecording = recording

if (curRecording != null) {

// Stop the current recording session.

curRecording.stop()

recording = null

return

}

// create and start a new recording session

val name = SimpleDateFormat(FILENAME\_FORMAT, Locale.US)

.format(System.currentTimeMillis())

val contentValues = ContentValues().apply {

put(MediaStore.MediaColumns.DISPLAY\_NAME, name)

put(MediaStore.MediaColumns.MIME\_TYPE, "video/mp4")

if (Build.VERSION.SDK\_INT > Build.VERSION\_CODES.P) {

put(MediaStore.Video.Media.RELATIVE\_PATH, "Movies/CameraX-Video")

}

}

val mediaStoreOutputOptions = MediaStoreOutputOptions

.Builder(contentResolver, MediaStore.Video.Media.EXTERNAL\_CONTENT\_URI)

.setContentValues(contentValues)

.build()

recording = videoCapture.output

.prepareRecording(this, mediaStoreOutputOptions)

.apply {

if (PermissionChecker.checkSelfPermission(this@MainActivity,

Manifest.permission.RECORD\_AUDIO) ==

PermissionChecker.PERMISSION\_GRANTED)

{

withAudioEnabled()

}

}

.start(ContextCompat.getMainExecutor(this)) { recordEvent ->

when(recordEvent) {

is VideoRecordEvent.Start -> {

viewBinding.videoCaptureButton.apply {

text = getString(R.string.stop\_capture)

isEnabled = true

}

}

is VideoRecordEvent.Finalize -> {

if (!recordEvent.hasError()) {

val msg = "Video capture succeeded: " +

"${recordEvent.outputResults.outputUri}"

Toast.makeText(baseContext, msg, Toast.LENGTH\_SHORT)

.show()

Log.d(TAG, msg)

} else {

recording?.close()

recording = null

Log.e(TAG, "Video capture ends with error: " +

"${recordEvent.error}")

}

viewBinding.videoCaptureButton.apply {

text = getString(R.string.start\_capture)

isEnabled = true

}

}

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

以下分析这段代码。首先检查是否已创建 VideoCapture 用例：如果尚未创建，则不执行任何操作。

val videoCapture = videoCapture ?: return

1

在 CameraX 完成请求操作之前，停用界面；在后续步骤中，它会在已注册的 VideoRecordListener 内重新启用。

viewBinding.videoCaptureButton.isEnabled = false

1

如果有正在进行的录制操作，将其停止并释放当前的 recording。当所捕获的视频文件可供应用使用时，我们会收到通知。

val curRecording = recording

if (curRecording != null) {

curRecording.stop()

recording = null

return

}

1

2

3

4

5

6

为了开始录制，创建一个新的录制会话。首先，创建预定的 MediaStore 视频内容对象，将系统时间戳作为显示名（以便可以捕获多个视频）。

val name = SimpleDateFormat(FILENAME\_FORMAT, Locale.US)

.format(System.currentTimeMillis())

val contentValues = ContentValues().apply {

put(MediaStore.MediaColumns.DISPLAY\_NAME, name)

put(MediaStore.MediaColumns.MIME\_TYPE, "video/mp4")

if (Build.VERSION.SDK\_INT > Build.VERSION\_CODES.P) {

put(MediaStore.Video.Media.RELATIVE\_PATH,

"Movies/CameraX-Video")

}

}

1

2

3

4

5

6

7

8

9

10

使用外部内容选项创建 MediaStoreOutputOptions.Builder。

val mediaStoreOutputOptions = MediaStoreOutputOptions

.Builder(contentResolver,

MediaStore.Video.Media.EXTERNAL\_CONTENT\_URI)

1

2

3

将创建的视频 contentValues 设置为 MediaStoreOutputOptions.Builder，并构建我们的 MediaStoreOutputOptions 实例。

.setContentValues(contentValues)

.build()

1

2

将输出选项配置为 VideoCapture 的 Recorder 并启用录音：

videoCapture

.output

.prepareRecording(this, mediaStoreOutputOptions)

.withAudioEnabled()

1

2

3

4

在此录音中启用音频。

.apply {

if (PermissionChecker.checkSelfPermission(this@MainActivity,

Manifest.permission.RECORD\_AUDIO) ==

PermissionChecker.PERMISSION\_GRANTED)

{

withAudioEnabled()

}

}

1

2

3

4

5

6

7

8

启动这项新录制内容，并注册一个 lambda VideoRecordEvent 监听器。

.start(ContextCompat.getMainExecutor(this)) { recordEvent ->

//lambda event listener

}

1

2

3

当相机设备开始请求录制时，将“Start Capture”按钮文本切换为“Stop Capture”。

is VideoRecordEvent.Start -> {

viewBinding.videoCaptureButton.apply {

text = getString(R.string.stop\_capture)

isEnabled = true

}

}

1

2

3

4

5

6

完成录制后，用消息框通知用户，并将“Stop Capture”按钮切换回“Start Capture”，然后重新启用它：

is VideoRecordEvent.Finalize -> {

if (!recordEvent.hasError()) {

val msg = "Video capture succeeded: " +

"${recordEvent.outputResults.outputUri}"

Toast.makeText(baseContext, msg, Toast.LENGTH\_SHORT)

.show()

Log.d(TAG, msg)

} else {

recording?.close()

recording = null

Log.e(TAG, "Video capture succeeded: " +

"${recordEvent.outputResults.outputUri}")

}

viewBinding.videoCaptureButton.apply {

text = getString(R.string.start\_capture)

isEnabled = true

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

在 startCamera() 中，将以下代码放置在 preview 创建行之后。这将创建 VideoCapture 用例。

val recorder = Recorder.Builder()

.setQualitySelector(QualitySelector.from(Quality.HIGHEST))

.build()

videoCapture = VideoCapture.withOutput(recorder)

1

2

3

4

（可选）同样在 startCamera() 中，通过删除或注释掉以下代码来停用 imageCapture 和 imageAnalyzer 用例：

/\* comment out ImageCapture and ImageAnalyzer use cases

imageCapture = ImageCapture.Builder().build()

val imageAnalyzer = ImageAnalysis.Builder()

.build()

.also {

it.setAnalyzer(cameraExecutor, LuminosityAnalyzer { luma ->

Log.d(TAG, "Average luminosity: $luma")

})

}

\*/

1

2

3

4

5

6

7

8

9

10

11

将 Preview + VideoCapture 用例绑定到生命周期相机。仍在 startCamera() 内，将 cameraProvider.bindToLifecycle() 调用替换为以下代码：

// Bind use cases to camera

cameraProvider.bindToLifecycle(this, cameraSelector, preview, videoCapture)

1

2

构建并运行项目。

录制一些剪辑：

按“START CAPTURE”按钮。注意，图片说明会变为“STOP CAPTURE”。

录制几秒钟或几分钟的视频。

按“STOP CAPTURE”按钮（和 start capture 按钮是同一个按钮）。

视频录制测试可以看到视频被成功保存至媒体库。

————————————————

版权声明：本文为CSDN博主「llfjfz」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/llfjfz/article/details/129924593