SAMSUNG

IT Академия Kotlin

Kotlin Базовый Курс

5.2. Общее хранилище файлов

Обзор общего хранилища файлов

Android предоставляет API для хранения и доступа к следующим типам общих данных:
Медиа. В Android есть общедоступные каталоги для медиаресурсов: фото, аудиофайлы и т.д. Приложение может получить доступ к медиаконтенту с использованием MediaStore API платформы, о котором речь пойдет позже.
Документы. В системе есть специальный каталог для хранения документов (например файлов PDF или книг использующие в формате EPUB). Для доступа к этим файлам можно использовать Storage Access Framework.
Датасеты. На Android 11 (уровень API 30) и выше система кэширует большие наборы данных, используемые несколькими приложениями. Данные наборы могут использоваться например при решении задач машинного обучения. Приложения могут получить доступ к этим общим наборам данных с помощью API BlobStoreManager.

Доступ к медиафайлам из общего хранилища

SAMSUNG IT Академия Kotlin

```
val projection = arrayOf(MediaStore.Images.Media._ID, MediaStore.Images.Media.DISPLAYNAME,
MediaStore.Images.Media.DATE_TAKEN)
val selection = "${MediaStore.Images.Media.DATE_TAKEN} >= ?"
val selectionArgs = arrayOf dateToTimestamp(day = 24, month = 7, year = 2019).toString())
val sortOrder = "${MediaStore.Images.Media.DATE_TAKEN} DESC” getApplication().contentResolver.query(
MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
projection,
selection,
selectionArgs,
sortOrder)?.use { cursor -> imageList = addImagesFromCursor(cursor) }
```

Открытие медиафайла

Чтобы открыть медиафайл с помощью файлового дескриптора, необходимо задействовать логику, аналогичную следующей:

```
val resolver = applicationContext.contentResolver
val readOnlyMode = "r" resolver.openFileDescriptor(content-uri, readOnlyMode).use { pfd ->
    // операции с pdf
}
```

Открыть медиафайл с помощью файлового потока можно следующим образом:

```
val resolver = applicationContext.contentResolver
resolver.openInputStream(content-uri).use {
    stream -> // операции с файловым потоком
}
```

Сохранение данных

Пример сохранения данных с помощью Scoped Storage:

```
val values = ContentValues().apply { put(MediaStore.Images.Media.DISPLAY_NAME, name)
put(MediaStore.Images.Media.MIME_TYPE, “image/jpeg")
put(MediaStore.Images.Media.RELATIVE_PATH, “Pictures/$bucketName/")
put(MediaStore.Images.Media.IS_PENDING, 1) }
val collection = MediaStore.Images.Media.getContentUri(MediaStore.VOLUME_EXTERNAL_PRIMARY)
val imageUri = context.contentResolver.insert(collection, values)
context.contentResolver.openOutputStream(imageUri).use { out ->
    bmp.compress(Bitmap.CompressFormat.JPEG, 90, out)
}
values.clear()
values.put(MediaStore.Images.Media.IS_PENDING, 0)
context.contentResolver.update(imageUri, values, null, null)
```

Сохранение данных

SAMSUNG IT Академия Kotlin

Чтобы удалить файл мультимедиа с помощью MediaStore API можно задействовать логику заложенную в следующем фрагменте кода:

```
try {
    getApplication().contentResolver.delete(
    image.contentUri,"${MediaStore.Images.Media._ID} = ?",
    arrayOf(image.id.toString()) )
}
catch (securityException: SecurityException) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        val recoverableSecurityException =
        securityException as? RecoverableSecurityException
        ?: throw securityException
        pendingDeleteImage = image
        _permissionNeededForDelete.postValue(
        recoverableSecurityException.userAction.actionIntent.intentSender )
    } else { throw securityException }
}
```

Сохранение данных

SAMSUNG IT Академия Kotlin

Чтобы вызвать метод contentResolver.delete(), его следует поместить в блок try, потому что он может вызвать исключение безопасности во время выполнения.
Начиная с Android R появились методы для получение массового доступа для изменения (createWriteRequest) и удаления (createDeleteRequest) данных. Например удаление можно осуществить следующим образом:

```
fun deleteMediaBulk(context: Context, media: List): IntentSender {
    val uris = media.map { it.uri }
    return MediaStore.createDeleteRequest(context.contentResolver, uris).intentSender
}
```

Доступ к документам и другим файлам из общего хранилища

SAMSUNG

IT Академия Kotlin

Storage Access Framework поддерживает следующие варианты использования для доступа к файлам и другим документам:

- **Создание нового файла.** При этом в намерения передается константа ACTION_CREATE_DOCUMENT. В данном варианте пользователям позволено сохранять файл в определенном месте.
- **Открытие файла или документа**. При этом в намерения передается константа ACTION_OPEN_DOCUMENT, что позволяет пользователям выбрать конкретный документ или файл для открытия.
- **Предоставление доступа к содержимому каталога**. Доступно с версии Android 5.0 и выше. При этом в намерения передается константа ACTION_OPEN_DOCUMENT_TREE. Позволяет пользователям выбирать определенный каталог, предоставляя приложению доступ ко всем его файлам и подкаталогам.

```
const val CREATE_FILE = 1
private fun createFile(pickerInitialUri: Uri) {
    val intent = Intent(Intent.ACTION_CREATE_DOCUMENT).apply {
        addCategory(Intent.CATEGORY_OPENABLE) type = "application/pdf"
        putExtra(Intent.EXTRA_TITLE, "invoice.pdf")
        putExtra(DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
}
startActivityForResult(intent, CREATE_FILE) }
```

Доступ к документам и другим файлам из общего хранилища

После того, как пользователь выбрал каталог, нам все еще нужно обработать Uri результата в методе onActivityResult.

```
override fun onActivityResult( requestCode: Int, resultCode: Int, resultData: Intent?) {
    if (requestCode == CREATE_FILE && resultCode == Activity.RESULT_OK) {
        //Данные результата содержат URI для каталога, выбранного пользователем.
        resultData?.data?.also { uri -> // сохраните ваши данные с помощью `uri` }
    }
}
```

В следующем фрагменте кода показано, как создать и вызвать намерение для открытия документа PDF:

```
const val PICK_PDF_FILE = 2
fun openFile(pickerInitialUri: Uri) {
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {
        addCategory(Intent.CATEGORY_OPENABLE) type = "application/pdf"
        putExtra(DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
    }
    startActivityForResult(intent, PICK_PDF_FILE)
}
```

Доступ к документам и другим файлам из общего хранилища

SAMSUNG IT Академия Kotlin

В следующем фрагменте кода показано, как создать и вызвать намерение для открытия каталога:

```
fun openDirectory(pickerInitialUri: Uri) {
    // Выберите каталог с помощью средства выбора файлов системы.
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT_TREE).apply {
        // Можно указать URI для каталога, который должен открываться в средстве выбора файлов при его загрузке.
    putExtra(DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
    }
    startActivityForResult(intent, your-request-code)
}
```

После того как пользователь выбрал файл или каталог с помощью средства выбора файлов системы, вы можете получить URI выбранного элемента, используя следующий код в onActivityResult():

```
override fun onActivityResult( requestCode: Int, resultCode: Int, resultData: Intent?) {
    if (requestCode == CREATE_FILE && resultCode == Activity.RESULT_OK) {
        // Данные результата содержат URI для документа или каталога, выбранного пользователем.
        resultData?.data?.also { uri -> // Выполняйте операции с документом, используя его URI. }
    }
}
```

Доступ к документам и другим файлам из общего хранилища

SAMSUNG IT Академия Kotlin

В следующем фрагменте кода показано, как открыть файл растрового изображения с учетом его URI:

```
val contentResolver = applicationContext.contentResolver
@Throws(IOException::class)
private fun getBitmapFromUri(uri: Uri): Bitmap {
    val parcelFileDescriptor: ParcelFileDescriptor = contentResolver.openFileDescriptor(uri, "r")
    val fileDescriptor: FileDescriptor = parcelFileDescriptor.fileDescriptor
    val image: Bitmap = BitmapFactory.decodeFileDescriptor(fileDescriptor)
    parcelFileDescriptor.close()
    return image
}
```

Следующий фрагмент кода перезаписывает содержимое документа по URI:

```
val contentResolver = applicationContext.contentResolver
private fun alterDocument(uri: Uri) {
    try {
        contentResolver.openFileDescriptor(uri, "w")?.use {
            FileOutputStream(it.fileDescriptor).use {
                it.write( ("Overwritten at ${System.currentTimeMillis()}\n") .toByteArray() )
            }
        }
    } catch (e: FileNotFoundException) { e.printStackTrace()
    } catch (e: IOException) { e.printStackTrace() } }
```

Сохранение пар "ключ-значение"

Если есть небольшая коллекция из пар "ключ-значение", которые вы хотите сохранить, необходимо использовать SharedPreferences API . Объект SharedPreferences указывает на файл, содержащий пары ключ-значение, и предоставляет простые методы для их чтения и записи.

Сохранение пар "ключ-значение"

SAMSUNG IT Академия Kotlin

Чтобы получить экземпляр класса SharedPreferences для получения доступа к настройкам в коде приложения используются следующие методы:
•getPreferences()— внутри активности, чтобы обратиться к определенному для активности предпочтению;
•getSharedPreferences() — внутри активности, чтобы обратиться к предпочтению на уровне приложения

```
val sharedPref = activity?.getSharedPreferences( getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

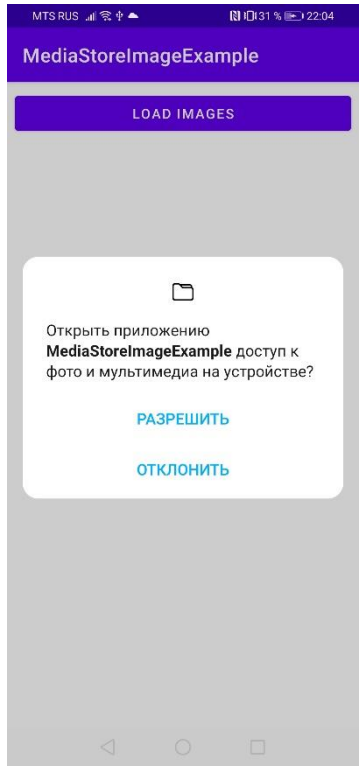
```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```

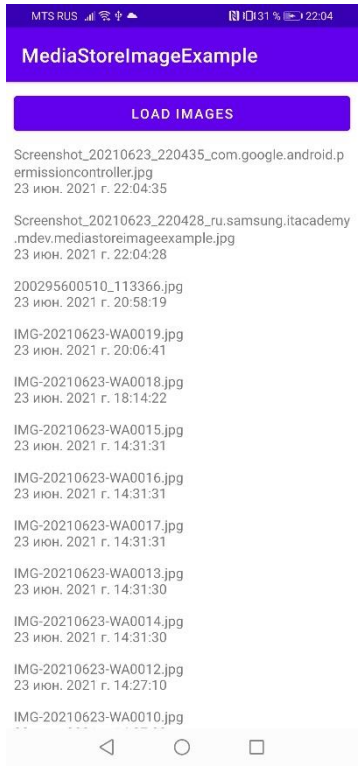
```
val sharedPreferences = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPreferences.edit()) {
    putInt(getString(R.string.mySrting), newMyString) apply()
}
```

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
val defaultValue = resources.getInteger(R.integer.my_int_default_key)
val myValue = sharedPref.getInt(getString(R.string.my_saved_int_value_key), defaultValue)
```

Упражнение

Разработаем приложение, в котором по нажатию кнопки в текстовом поле будут отображаться имена файлов двадцати последних сохраненных на устройстве изображений вместе с датой их последнего изменения.





SAMSUNG IT Академия Kotlin

Спасибо