SAMSUNG

IT Академия Kotlin

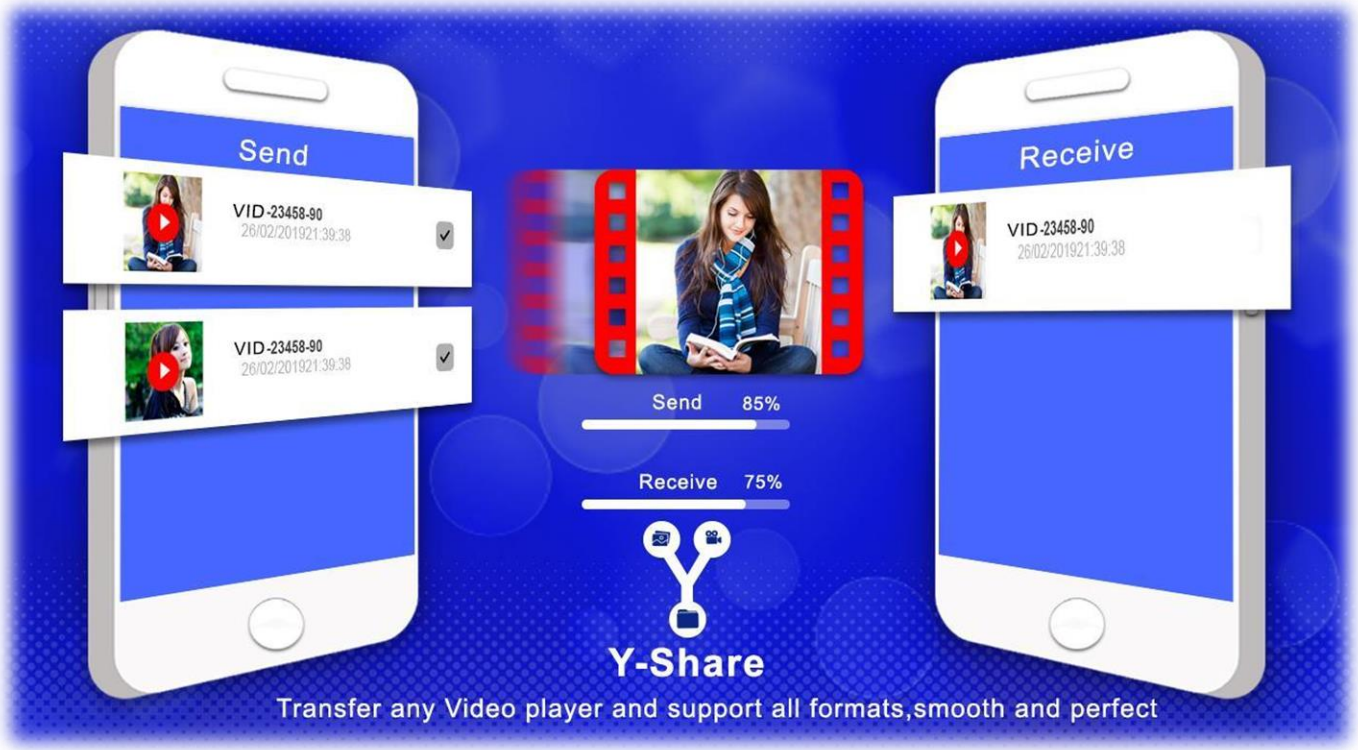
Kotlin Базовый Курс

5.4. Обмен данными и файлами

Обмен простыми данными между приложениями

SAMSUNG IT Академия Kotlin

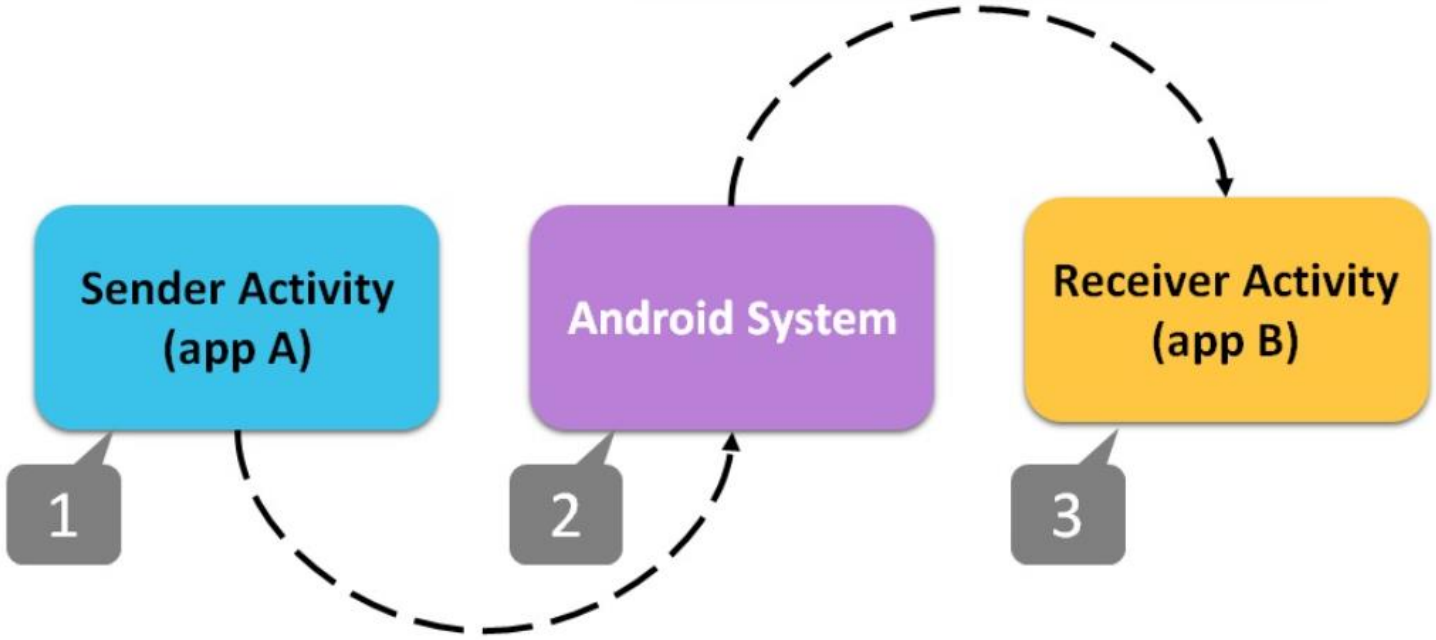
Одной из замечательных опций приложений для Android-приложений - это их способность общаться и интеграции друг с другом. Действительно, зачем изобретать функционал, который не является базисом для вашего приложения, когда он уже есть в другом?



Отправка простых данных другим приложениям

SAMSUNG IT Академия Kotlin

•**Общая таблица (Sharesheet) Android.** Предназначена для отправки информации за пределы приложения и/или непосредственно другому пользователю. Например, совместное использование URL-адреса с другим пользователем.
•**Распознаватель намерений (Intent Resolver) Android.** В большей степени подходит для передачи данных на следующий этап четко определенной задачи.



Отправка простых данных другим приложениям

Для передачи информации необходимо создать намерение и установить для его действия значение Intent.ACTION_SEND. Чтобы отобразить Sharesheet Android, вам нужно вызвать метод Intent.createChooser().

```
val sendIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "Hello, world!!!")
    type = "text/plain"
}
val shareIntent = Intent.createChooser(sendIntent, null)
startActivity(shareIntent)
```

```
val shareIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_STREAM, uri)
    type = "image/jpeg"
}
startActivity(Intent.createChooser(shareIntent, resources.getText(R.string.send_to)))
```

Отправка простых данных другим приложениям

Получающему приложению необходимо разрешение на доступ к данным, на которые указывает URI. Это можно сделать, используя один следующих способов:

•Хранение данных в своем собственном контент-провайдере (ContentProvider). При этом целесообразно убедиться, что другие приложения имеют правильное разрешение на доступ к нему. Разработчики предлагают в качестве предпочтительного механизма использование временных разрешений на URI и предоставление доступа только приложению-получателю. Простой способ такай ContentProvider- использовать класс FileProvider.

•Использование MediaStore Api. Хранилище MediaStore в основном используется для типов MIME видео, аудио и изображений. Но, начиная с Android 3.0 (уровень API 11), в нем можно хранить типы, не относящиеся к мультимедиа.

При отправке данных другому приложению вы должны указать наиболее конкретный тип MIME. Например, при совместном использовании обычного текста следует использовать text/plain. Тип MIME */* использовать не рекомендуется. Далее приведем несколько распространенных типов MIME при отправке простых данных в Android:

•text/plain, text/rtf, text/html, text/json
•image/jpg, image/png, image/gif
•video/mp4, video/3gp
•application/pdf

Отправка простых данных другим приложениям

Получающему приложению необходимо разрешение на доступ к данным, на которые указывает URI. Это можно сделать, используя один следующих способов:

•Хранение данных в своем собственном контент-провайдере (ContentProvider). При этом целесообразно убедиться, что другие приложения имеют правильное разрешение на доступ к нему. Разработчики предлагают в качестве предпочтительного механизма использование временных разрешений на URI и предоставление доступа только приложению-получателю. Простой способ такай ContentProvider- использовать класс FileProvider.

•Использование MediaStore Api. Хранилище MediaStore в основном используется для типов MIME видео, аудио и изображений. Но, начиная с Android 3.0 (уровень API 11), в нем можно хранить типы, не относящиеся к мультимедиа.

При отправке данных другому приложению вы должны указать наиболее конкретный тип MIME. Например, при совместном использовании обычного текста следует использовать text/plain. Тип MIME */* использовать не рекомендуется. Далее приведем несколько распространенных типов MIME при отправке простых данных в Android:

•text/plain, text/rtf, text/html, text/json
•image/jpg, image/png, image/gif
•video/mp4, video/3gp
•application/pdf

Более подробно об отправлении простых данных с использованием Sharesheet Android можно узнать в официальной документации.

Прием простых данных другим приложениям

SAMSUNG IT Академия Kotlin

Для определения того, какие данные приложение готово принять существуют фильтры в манифесте. В приложениях-приемниках сообщений необходимо определить фильтр намерений в манифесте, используя элемент <intent-filter>. Например, если ваше приложение обрабатывает прием текста, одного или нескольких изображений любого типа, файл AndroidManifest.xml может выглядеть так:

```
activity android:name=".MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

Прием простых данных другим приложениям

SAMSUNG

IT Академия Kotlin

```
override fun onCreate(savedInstanceState: Bundle?) {
    //...
    when {
        intent?.action == Intent.ACTION_SEND -> {
            if ("text/plain" == intent.type) {
                handleSendText(intent) // обработка отправляемого текста
            } else if (intent.type?.startsWith("image/") == true) {
                handleSendImage(intent) // обработка одной отправляемой картинки
            }
        }
        intent?.action == Intent.ACTION_SEND_MULTIPLE
                && intent.type?.startsWith("image/") == true -> {
                handleSendMultipleImages(intent) // обработка множества отправляемых картинок
        }
        else -> {
            // обработка других намерений
        }
    }
    //...
}
```

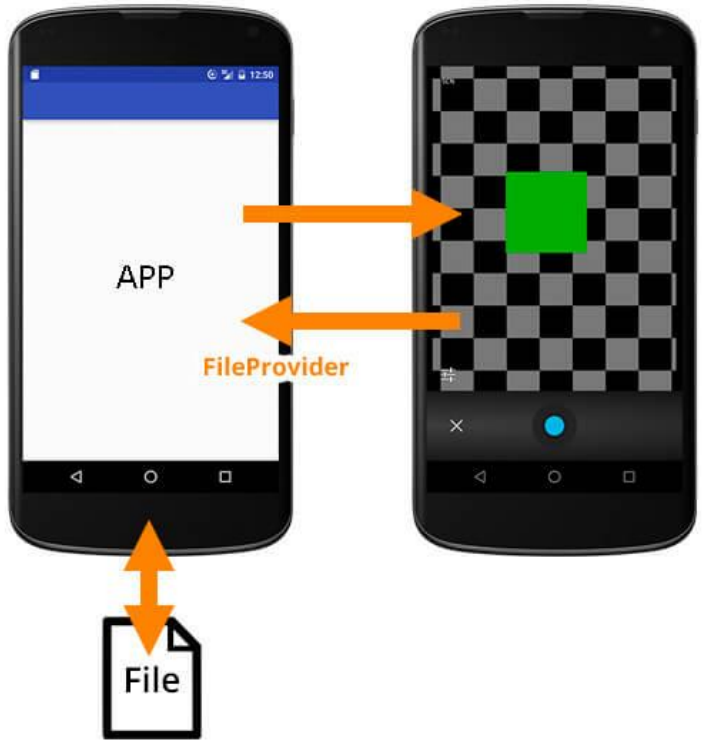
```
private fun handleSendText(intent: Intent) {
    intent.getStringExtra(Intent.EXTRA_TEXT)?.let {
        // обновление UI для отображение отправленного текста
    }
}

private fun handleSendImage(intent: Intent) {
    (intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM) as? Uri)?.let {
        // обновление UI для отображение отправленного изображения
    }
}

private fun handleSendMultipleImages(intent: Intent) {
    intent.getParcelableArrayListExtra<Parcelable>(Intent.EXTRA_STREAM)?.let {
        // обновление UI для отображение множества отправленных изображений
    }
}
```

Обмен файлами между приложениями

В Android существует специальный класс FileProvider, в котором есть метод getUriForFile() для создания URI содержимого файла. FileProvider — это подкласс ContentProvider. Хотя ContentProvider — это компонент, который позволяет вам безопасно делиться любыми данными, FileProvider используется специально для совместного использования внутренних файлов приложения.

Чтобы FileProvider работал, нужно выполнить три следующих действия:

- Определить FileProvider в файле AndroidManifest.xml;
- Создать XML-файл, содержащий все пути, которые FileProvider будет использовать совместно с другими приложениями;
- Связать действительный URI в Intent и активировать его.



Обмен файлами между приложениями

```
<manifest>
    ...
    <application>
        ...
        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="com.mydomain.fileprovider"
            android:exported="false"
            android:grantUriPermissions="true">
            ...
        </provider>
        ...
    </application>
</manifest>
```

После добавления провайдера в манифест приложения необходимо указать каталоги, содержащие файлы, которыми вы хотите поделиться. Чтобы указать каталоги, создайте файл с именем *filepaths.xml* в каталоге *res/xml/* проекта. В этом файле укажите каталоги, добавив элемент XML для каждого каталога. В следующем фрагменте показан пример содержимого *res/xml/filepaths.xml*.

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

Совместное использование файлов приложениями

SAMSUNG

IT Академия Kotlin

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    ...
        <application>
        ...
            <activity
                android:name=".FileSelectActivity"
                android:label="@File Selector" >
                <intent-filter>
                    <action
                        android:name="android.intent.action.PICK"/>
                    <category
                        android:name="android.intent.category.DEFAULT"/>
                    <category
                        android:name="android.intent.category.OPENABLE"/>
                    <data android:mimeType="text/plain"/>
                    <data android:mimeType="image/*"/>
                </intent-filter>
            </activity>
```

Совместное использование файлов приложениями

SAMSUNG IT Академия Kotlin

```
class MainActivity : Activity() {

    // Путь к корню внутреннего хранилища  приложения.
    private lateinit var privateRootDir: File
    // Путь к подкаталогу "images"
    private lateinit var imagesDir: File
    // Массив файлов в подкаталоге images
    private lateinit var imageFiles: Array<File>
    // Массив имен файлов, соответствующих imageFiles
    private lateinit var imageFilenames: Array<String>

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // Настройка намерения для отправки данных запрашивающим приложениям
        resultIntent = Intent("com.example.myapp.ACTION_RETURN_FILE")
        // Получение файлов / подкаталогов внутреннего хранилища
        privateRootDir = filesDir
        // Получение подкаталога files / images;
        imagesDir = File(privateRootDir, "images")
        // Получение файлов в подкаталоге images
        imageFiles = imagesDir.listFiles()
        // Set the Activity's result to null to begin with
        setResult(Activity.RESULT_CANCELED, null)
        // ...
    }
```

Совместное использование файлов приложениями

SAMSUNG

IT Академия Kotlin

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    // Определите слушателя, который реагирует на щелчки по файлу в ListView
    fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
        // Предположим, что имена файлов находятся в массиве imageFilename.
        val requestFile = File(imageFilenames[position])

        // Используйте FileProvider, чтобы получить URI содержимого
        val fileUri: Uri? = try {
            FileProvider.getUriForFile(
                    this@MainActivity,
                    "com.example.myapp.fileprovider",
                    requestFile)
        } catch (e: IllegalArgumentException) {
            Log.e("File Selector",
                    "The selected file can't be shared: $requestFile")
            null
        }
        //...
    }
    //...
}
```

Совместное использование файлов приложениями

SAMSUNG IT Академия Kotlin

Следующий код демонстрирует, как предоставить доступ к файлу для чтения:

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    // Определите слушателя, который реагирует на щелчки по файлу в ListView
    fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
            //...
            if (fileUri != null) {
                // Предоставить временное разрешение на чтение URI контента
                resultIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
                //...
            }
            //...
        }
        //...
    }
```

Совместное использование файлов приложениями

SAMSUNG IT Академия Kotlin

```
override fun onCreate(savedInstanceState: Bundle?) {
    //...
    // Определите слушателя, который реагирует на щелчки по файлу в ListView
    fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
        //...
        if (fileUri != null) {
            //...
            // Поместите тип Uri и MIME в Intent
            resultIntent.setDataAndType(fileUri, contentResolver.getType(fileUri))
            // Установите результат
             setResult(Activity.RESULT_OK, resultIntent)
         } else {
            resultIntent.setDataAndType(null, "")
            setResult(RESULT_CANCELED, resultIntent)
        }
    }
```

Совместное использование файлов приложениями

SAMSUNG IT Академия Kotlin

```
class MainActivity : Activity() {
    private lateinit var requestFileIntent: Intent
    private lateinit var inputPFD: ParcelFileDescriptor
    //...
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        requestFileIntent = Intent(Intent.ACTION_PICK).apply {
            type = "image/jpg"
        }
        //...
    }
   // ...
    private fun requestFile() {
       startActivityForResult(requestFileIntent, 0)
        //...
    }
   // ...
}
```

Совместное использование файлов приложениями

SAMSUNG

IT Академия Kotlin

```
public override fun onActivityResult(requestCode: Int, resultCode: Int, returnIntent: Intent) {
    //Если выбор не сработал
    if (resultCode != Activity.RESULT_OK) {
        // Выйти, ничего не делая
        return
    }
    // Получить URI содержимого файла из входящего Intent
    returnIntent.data?.also { returnUri ->
        /*
         * Попытка открыть файл для чтения, используя возвращенный URI.
         */
        inputPFD = try {
            contentResolver.openFileDescriptor(returnUri, "r")
        } catch (e: FileNotFoundException) {
            e.printStackTrace()
            Log.e("MainActivity", "File not found.")
            return
        }

        // обычный файловый дескриптор для файла
        val fd = inputPFD.fileDescriptor
        ...
    }
}
```

Получение информации о файле

SAMSUNG IT Академия Kotlin

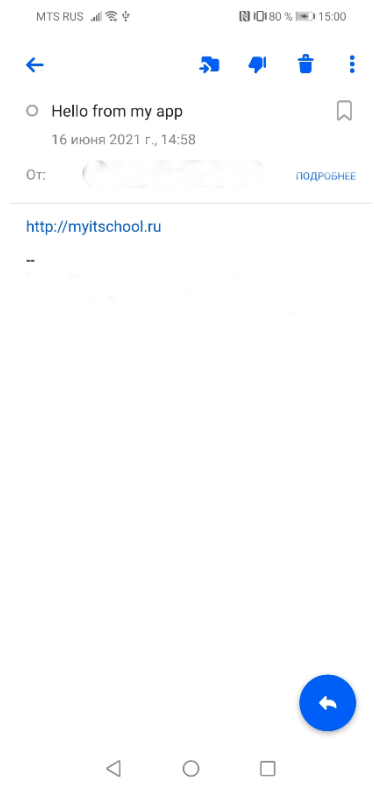
```
val mimeType: String? = returnIntent.data?.let {
    returnUri -> contentResolver.getType(returnUri)
}
```

•DISPLAY_NAME - имя файла (String). Значение такое же, как значение, возвращаемое метом File.getName().
•SIZE - размер файла в байтах (long) Это значение такое же, как значение, возвращаемое методом File.length().

```
returnIntent.data?.let { returnUri ->
    contentResolver.query(returnUri, null, null, null, null)}?.use { cursor ->
        val nameIndex = cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME)
        val sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE)
        cursor.moveToFirst()
        findViewById<TextView>(R.id.filename_text).text = cursor.getString(nameIndex)
        findViewById<TextView>(R.id.filesize_text).text = cursor.getLong(sizeIndex).toString()
}
```

Упражнение

Разработаем простое приложение, которое может делиться содержимым, таким как URL, текст и файлы изображений с другими приложениями, установленными на вашем устройстве Android.



SAMSUNG IT Академия Kotlin

Спасибо