4.6. Широковещательные сообщения (Broadcast). Приёмники трансляций.

Сайт:	Samsung Innovation Campus	Напечатано::	Murad Rezvan
Курс:	Мобильная разработка на Kotlin	Дата:	понедельник, 3 июня 2024, 17:56
Книга:	4.6. Широковещательные сообщения (Broadcast). Приёмники трансляций.		

Оглавление

1. ШИРОКОВЕЩАТЕЛЬНЫЕ СООБЩЕНИЯ

Безопасность ОС Android обеспечивается, в том числе, запуском отдельной виртуальной машины для каждого приложения. Таким образом, другим приложениям недоступны данные, принадлежащие текущему процессу. Так же приложение не может использовать системные возможности в связи со своей локализацией в отдельном информационном потоке. Однако, достаточно часто возникают ситуации, когда приложению необходимы сведения о состоянии или настройках системы, по этому Android публикует сообщения о происходящих системных событиях в своём пространстве. Для чтения этих сообщений у приложения должна быть настроена подписка на подобные рассылки. Понятно, что читать все сообщения подряд приложению нет необходимости, по этому подписка оформляется отдельно на каждое событие. Не менее часты случаи, когда у приложения возникает необходимость оповестить другие приложения (или систему) о своей деятельности: желании использовать данные из других приложений; запуску «тяжеловесного» процесса, для выполнения которого необходимо много энергетических ресурсов устройства; готовности «поделиться» с другими приложениями полученной информацией и т.п. То есть возникает необходимость «общения» приложения с другими процессами в системе. Такой функционал реализуется посредством объектов-трансляций, называемых широковещательными сообщениями (Broadcast). Для регистрации сообщений используется широковещательный приёмник (BroadcastReceiver), для отправки - метод sendBroadcast(), вызываемый в контексте приложения. Само широковещательное сообщение представляется в виде намерения, по этому сообщения могут быть явными (с указанием, кому предназначена трансляция) и неявными (адресованы всем подписчикам).

Обмен трансляциями предназначен для информирования приложений о состояниях системы или других приложений. С завершением трансляции так же завершаются все процессы, запущенные в рамках текущей трансляции. По этому не стоит применять Broadcast для запуска выполнения каких-либо действий или запускать в трансляциях фоновые потоки.

Применение широковещательных возможностей для отслеживания состояния системы более рационально по сравнению с использованием сервисов, поскольку сервис постоянно проявляет активность для контроля состояния, а приёмник находится в ожидании и активизируется только при получении сообщения.

Набор системных трансляций, доступных к прослушиванию в текущем приложении, зависит от используемой операционной системы, а полный список всех сообщений размещён в файле BROADCAST_ACTIONS.TXT в пакете Android SDK/platforms в папке data для каждой платформы.

2. ПРИЁМНИКИ СИСТЕМНЫХ ШИРОКОВЕЩАТЕЛЬНЫХ СООБЩЕНИЙ

Работа приёмника трансляций описывается в отдельном классе (можно во вложенном в класс приложения), наследованном от абстрактного класса `BroadcastReceiver`. Деятельность приёмника программируется в единственном имплементируемом методе `onReceive(Context?, Intent?)`.

```
class MyRec: BroadcastReceiver() {
override fun onReceive(p0: Context?, p1: Intent?) {
    TODO("Not yet implemented")
}
}
```

Выполнение BroadcastReceiver является процессом переднего плана, то есть выполняется в главном потоке. По этой причине тело метода onReceive() не должно содержать «тяжёлых» или «длинных» действий.

Для того, чтобы приёмник начал работать, необходимо провести его регистрацию как составляющей компоненты приложения и настроить фильтры намерений, по которым будет работать приёмник. Если приложение подписывается на несколько рассылок, то под каждое сообщение создаётся свой фильтр намерения

Существует два способа регистрации приёмника: статический и динамический.

Статическая регистрация предусматривает объявление приёмника в файле манифеста как составляющую приложения. В этом случае receiver будет работать даже при неактивном приложении. Но следует помнить, что в этом случае повышается расход батареи устройства. По этой причине, начиная с Android 7.0 разработчики системы ужесточают ограничения на приёмники, зарегистрированные таким образом. То есть объявленный в манифесте BroadcastReceiver на «старой» версии Android будет корректно работать, а на устройствах с операционной системой Android 7.0 и выше будет проигнорирован системой. Список ограничений пополняется с выходом каждой новой версии. Все изменения по мере появления публикуются разработчиками в статье.

Для регистрации приёмника как составляющей приложения в файле Manifest.xml в объекте `<application>` нужно добавить вложенный объект `<receiver>`, в качестве имени указать класс с описанием приёмника и вложить в него фильтры намерений:

```
<receiver android:name=".MainActivity$MyRec">
     <intent-filter>
          <!-- перечисление подписок приёмника-->
     </intent-filter>
 </receiver>
```

Динамическая регистрация приёмника осуществляется непосредственно в коде приложения. Работа делится на три этапа:

1. создание объекта приёмника

```
val receiver = MyRec()
```

2. задание фильтров намерений

```
val filter = IntentFilter().apply {
   // подключение подписок приёмника
}
```

3. регистрация приёмника с настроенными намерениями

```
registerReceiver(receiver, filter)
```

> С версии Android 8.0 все неявные широковещательные сообщения в системе регистрируются только динамически. Статическая регистрация разрешена только для явных и локальных трансляций. Список исключений, для которых разрешена регистрация в манифесте, публикуется разработчиками в документации

В отличие от статической регистрации, динамически зарегистрированные приёмники получают сообщения, на которые подписаны, пока работает контекст их регистрации. То есть приёмник, зарегистрированный в контексте приложения, будет читать сообщения только пока приложение работает.

Для оптимизации расхода заряда батареи устройства необходимо отключать приёмник, если приложение не нуждается в его использовании:

```
unregisterReceiver(receiver)
```

Управление жизнедеятельностью приёмника производится в рамках рабочего контекста.

Регистрация	Разрегистрация
onCreate()	onDestroy()
onStart()	onStop()
onResume()	onPause()

2.1. Упражнение 4.6.1 Создание приёмника системных сообщений

Требуется создать приложение, регистрирующее системные события.

1. Создайте новый проект, назовите его Subscriber.
2. Удалите в классе активности ссылку на контент из xml-разметки `setContentView(R.layout.activity_main)`
3. Создайте новый класс для приёмника системных трансляций. Для этого выполните команду File -> New -> Other -> Broadcast Receiver:

После этого шага в проекте появится новый класс MyReceiver - наследник BroadcastReceiver:

```
class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        // This method is called when the BroadcastReceiver is receiving an Intent broadcast.
        TODO("MyReceiver.onReceive() is not implemented")
    }
}
```

А в манифесте регистрация нового приёмника с именем созданного класса:

```
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
</receiver>
```

4. Заполните метод `onReceive()` выводом тоста с сообщением о полученной посылке

```
override fun onReceive(context: Context, intent: Intent) {
    Toast.makeText(context,
        "получено сообщение: "+intent.toString(),
           Toast.LENGTH_SHORT).show()
    }
```

5.1. Настройте в xml-регистрации в файле манифеста фильтр на прослушивание изменения режима "в самолёте"

```
<intent-filter>
                <action android:name="android.intent.action.AIRPLANE_MODE"/>
            </intent-filter>
```

Этот приёмник будет работать вне контекста приложения. То есть при закрытом приложении регистрация сообщений будет по-прежнему выполняться.

Для тестирования текущего пункта необходимо использовать устройство под управлением системы Android *ниже* API 24. Системой Android 7.0 и выше данный приёмник будет проигнорирован!

а) Запустите приложение, затем включите/выключите на устройстве авиарежим.

б) Завершите работу приложения кнопкой "Назад" и проверьте получение сообщений.

в) Удалите приложение из отложенных и снова переключите авиарежим устройства.

5. 2. Зарегистрируйте приёмник программно.

5.2.1. В классе активности объявите переменную приёмника:

```
val receiver = MyReceiver()
```

5.2.2. Для программного создания приёмника все его подписки необходимо регистрировать в объекте `IntentFilter`. Настройте приёмник на чтение изменения авиарежима устройства:

```
val intentFilterAvia = IntentFilter("android.intent.action.AIRPLANE_MODE")
```

5.2.3. Зарегистрируйте приёмник в методе `onCreate()`

```
registerReceiver(receiver, intentFilterAvia)
```

Приёмник будет зарегистрирован при создании активности, то есть с самого запуска приложения он будет регистрировать все изменения в настройках авиарежима устройства. Разрегистрация приёмника отсутствует, следовательно получился аналог слушателя из п. 5.1.

5.2.4. С целью экономии батареи добавим отмену приёмника. Поскольку регистрация выполнена в методе onCreate(), то разрегистрация должна быть в onDestroy(), поскольку объект приёмника в системе будет существовать, пока существует активность.

```
override fun onDestroy() {
        super.onDestroy()
        unregisterReceiver(receiver)
    }
```

6. Измените метод `onReceive()` на вывод активности сообщений:

```
override fun onReceive(context: Context, intent: Intent) {
    Toast.makeText(context,
       intent.action,
           Toast.LENGTH_SHORT).show()
    }
```

7. В классе активности добавьте ещё одну подписку на состояние подключения по Wi-Fi, используя тот же ресивер

```
registerReceiver(receiver, IntentFilter("android.net.wifi.WIFI_STATE_CHANGED"))
```

Теперь приложение прослушивает два события: изменение авиарежима, влекущее за собой изменении состояния Wi-Fi подключения и отдельно отключение/подключение Wi-Fi.

Протестируйте работу приёмников при включении и отключении модуля Wi-Fi и авиарежимма.

8. Переместите регистрацию и разрегистрацию приёмника в пары методов onStart()/onStop() и onResume()/onPause() и отследите факт получения сообщений.

2.2. Получение данных из намерения

Намерения, по которым приёмник получает сообщения, имеет свойства, зависящие от зарегистрированного сообщения и может прочитать передаваемые данные методами `getExtra()` в зависимости от типа читаемых данных. Для получения значений, передаваемых в неявном намерении, нужно настроить intentFilter() и получить из намерения нужное поле. Например, у сообщения `"android.intent.action.AIRPLANE_MODE"` имеется логическое поле `state`, несущее значение `true` при включенном авиарежиме и `false` при выключенном. Для получения значения этого поля из намерения применяется метод `getBooleanExtra("state",false)` со значением по умолчанию `false`. Если запрашиваемого свойства у намерения нет, то используется переданное значение по умолчанию.

Таким образом появляется возможность программирования поведения приложения в зависимости включения и выключения авиарежима.

2.3. Упражнение 4.6.2. Выполнение действий по широковещательному сообщению.

Внесём доработки в проект из Упражнения 4.6.1.

1. Добавьте в проект вторую активность с двумя кнопками, реализующую простейший кликер. На первой кнопке обработчик будет считать нажатия, на второй - закрытие активности.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SimpleClicker">
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="clicker"
    android:text="CLICK"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="closed"
        android:text="CLOSE"/>
</LinearLayout>
```

2. В классе активности определите методы `clicker()` для подсчёта нажатий и вывода их количества в текстовую подпись кнопки и `closed()` для закрытия активности.

```
fun clicker(view: View){
        (view as Button).setText("Нажатий: " + count++)
    }

 fun closed(view:View){
        finish()
    }
```

3. Организуйте запуск активности с кликером при включении авиарежима. Для этого переопределите метод onReceive()

```
 override fun onReceive(context: Context, intent: Intent) {
    if (intent.getBooleanExtra("state",false))
            startActivity(context,Intent(context,SimpleClicker::class.java),null)
        else
        Toast.makeText(context,
            "Работайте, пожалуйста!",
          Toast.LENGTH_SHORT).show()
    }
```

При включении авиарежима поле `state` намерения примет значение `true`, по этому выполнится запуск активности. При отключении авиарежима приложение выведет тост с сообщением "Работайте, пожалуйста!". Поскольку приёмник так же прослушивает изменения WiFi, то тост "Работайте, пожалуйста!" будет в любом случае выводиться. Закомментируйте регистрацию прослушивания подключения по WiFi для устранения этого повторения.

3. Подписки на сообщения от приложений

Помимо получения системных сообщений, приёмник можно настроить на чтение сообщений, посылаемых другими приложениями. Для этого необходимо знать:

- имя трансляции (некоторая текстовая строка, которой именуется посылка у приложения-отправителя)
- тег и тип получаемых данных, если они имеются.

```
val receiver=MyReceiver()
registerReceiver(receive, IntentFilter("name_broadcast.name_app"))
```

В качестве имени трансляции может быть использована любая строка. Обычно в ней указывают имя пакета, чтобы не было коллизий между посылками в пространстве сообщений системы. То есть регистрация трансляций от приложений ничем не отличается от подписки на системные трансляции. Более того, имеется возможность ограничить круг приложений, чьи трансляции будет читать приёмник. Для этого в качестве свойства намерению нужно установить разрешения и приёмник будет получать сообщения только от приложений, имеющих установленные разрешения.

```
registerReceiver(receiver, IntentFilter("name_broadcast.name_app"),
     Manifest.permission.READ_EXTERNAL_STORAGE,null)
```

Такой приёмник будет читать сообщения только от приложений, имеющих разрешение на чтение внешних накопителей.

> Если обработка получаемых сообщений не сильно отличается по логике, то все подписки можно объединить в один фильтр намерений и в методе onReceive() оператором выбора разобрать трансляции по видам через свойство `action`
>
> ```
> when (intent.action) {
> ...
> }
> ```
>
> В противном случае целесообразно разделение подписок по разным приёмникам. Такой подход делает код более читаемым, а управление жизненным циклом каждого приёмника более гибким.

Для обмена сообщениями между объектами внутри приложения используется класс LocalBroadcastManager, обеспечивающий конфиденциальность данных. Локальная трансляция даёт гарантию, что содержимое сообщения не выйдет за пределы контекста приложения и не станет доступно всем процессам операционной системы.

Локальные трансляции рассмотрены в следующей теме учебника.

3.1. Упражнение 4.6.3. Подписка на рассылки приложений

Скачайте, установите на устройство приложение и запустите его.

Нажимайте на портрет популярного блогера XIX века. Он что-то, несомненно, пишет, но нам этого не видно, поскольку на блог нужно подписаться. К счастью, у нас в предыдущих упражнениях уже создано приложение - подписчик, готовое читать всё, что ему скажешь. Подпишем его на *Козьма_Прутков*. Для этого

1. Создайте в проекте Subscriber второй класс-приёмник *My2Receiver* и переопределите метод `onReceive()` на чтение данных по тегу "text", как сообщает нам блогер.

```
class My2Receiver: BroadcastReceiver(){
    override fun onReceive(p0: Context?, p1: Intent) {
        Toast.makeText(p0,
            p1.getStringExtra("text"),
            Toast.LENGTH_SHORT).show()
    }
}
```

Намерение нужно обязательно объявить существующим, поскольку происходит чтение данных из его поля.

2. Создайте переменную приёмника в классе активности

```
val receiver2 = My2Receiver()
```

и выполните регистрацию приёмника в методе `onCreate()`.

```
registerReceiver(receiver2, IntentFilter("Козьма_Прутков"))
```

Отмену регистрации делать не нужно, поскольку трансляции будут генерироваться другим приложением, то есть при неактивном состоянии нашего подписчика. На версиях Android ниже 8.0 можно зарегистрировать receiver в манифесте, что будет равнозначно выполненной динамически регистрации без разрегистрации.

Нет смысла соединять два приёмника приложения в один, поскольку логика реализации прослушиваний сообщений различна.

3. Запустите Subscriber, переведите его в фоновую работу, активируйте Симулятор блогера и наслаждайтесь умными мыслями.

4. Измените регистрацию приёмника, добавив ещё два аргумента: разрешение на совершение звонков напрямую из приложения и null в качестве расписания получения сообщений

```
registerReceiver(receiver2, IntentFilter("Козьма_Прутков"), Manifest.permission.CALL_PHONE,null)
```

и повторите п.3. Чтение цитат невозможно, потому, что Симулятор блогера не имеет соответственного разрешения на совершение звонков.

5. Измените `Manifest.permission.CALL_PHONE` на `Manifest.permission.INTERNET`. Subscriber снова может читать Козьму Пруткова, поскольку у Симулятора блогера в манифесте имеется разрешение на выход в интернет.

Когда устанете от чтения цитат, переведите устройство в авиарежим и поработайте с кликером.

Готовое приложение можно посмотреть и сравнить с Вашим приложением.

Начать тур для пользователя на этой странице