**Лекция 7 Намерения**

В этой лекции обсуждается, как использовать [**Intents**](https://developer.android.com/guide/components/intents-filters.html) для связи между различными Activity и Applications. Система Intent позволяет Activity общаться, даже если у них нет ссылок друг на друга (и поэтому мы не можем просто вызвать метод для них).

Эта лекция ссылается на код, найденный по адресу <https://github.com/info448/lecture07-intents> . Обратите внимание, что для этой демонстрации вам понадобится рабочая камера на вашем устройстве. Чтобы включить камеру в эмуляторе, используйте Tools > Android > AVDменю для изменения эмулятора и выберите «веб-камера» для опции фронтальной камеры. Убедитесь, что она включена, запустив приложение «Камера».

Намерение — это **сообщение** , которое передается между компонентами приложения, позволяя им общаться!

* Большая часть объектной коммуникации, которую мы делаем, осуществляется через *прямой вызов метода* ; у вас есть ссылка на Объект, а затем вы вызываете метод для него. Мы также видели *обратные вызовы событий* , где при событии один из наших обратных вызовов выполняется системой, но на самом деле это просто обертка вокруг *прямого вызова метода* через шаблон Наблюдателя.
* Намерения работают немного по-другому: они позволяют нам создавать объект, который может быть «дан» другому компоненту (читай: Activity), который затем может ответить, получив его. Это похоже на обратный вызов события, но работает на немного более высоком системном уровне.

Намерения были ранее представлены в [Лекции 3.](https://info448.github.io/activities.html#activity-intents) Но повторюсь: вы можете думать о намерениях как о письмах, которые вы отправляете по почте: они адресованы определенной цели (например, другому Activity — точнее Context), и имеют место для некоторых данных, называемых **extras,** которые можно поместить внутрь (хранить в Bundle). Когда конверт придет, получатель может получить эти данные и что-то с ними сделать… и, возможно, отправить ответ обратно.

Обратите внимание, что существует несколько различных видов намерений; мы рассмотрим примеры каждого из них.

**7.1 Намерения относительно другой деятельности (явные)**

Самый базовый тип Intent — это Intent, отправленный в определенную Activity/Context, например, чтобы сказать Activity открыться. Это был тот же тип Intent, который был представлен в Лекции 3.

*//java*

*// context, target*

Intent intent = **new** Intent(MainActivity.this, SecondActivity.class);

startActivity(intent);

*//kotlin*

*// context, target*

**val** intent = Intent(this@MainActivity, SecondActivity::**class**.java)

Мы создаем экземпляр Intent (указывая, Contextоткуда и куда Classон должен быть доставлен), а затем используем это сообщение для запуска Activity, доставляя Intent через startActivity()метод.

* Метод startActivity()фактически означает: «Это намерение, используемое для запуска действия: тот, кто получает это намерение, должен начать!» Вы можете представить это так, как будто письма должны быть доставлены в почтовый ящик «начать действие».

Этот тип Intent называется **Explicit Intent,** потому что мы *явно* говорим, какой цели мы хотим его получить. Это письмо к определенному Activity.

**Дополнительно**

Мы также можем указать некоторые дополнительные данные внутри нашего конверта. Эти данные называются **Extras** . Это Bundle(набор примитивных пар ключ-значение), которые мы можем использовать для передачи *ограниченной* информации!

intent.putExtra("package.name.key", "value");

* В документации Android отмечается, что лучше всего включать полное имя пакета в ключи Bundle, чтобы избежать любых коллизий или неправильного прочтения данных (поскольку Intents могут перемещаться за пределы приложения). Также есть некоторые предопределенные значения ключей (константы), которые вы можете и будете использовать в Intentклассе.

Затем мы можем получить дополнительные данные из намерения в действии, которое его получает:

*//java*

*//in onCreate()*

Bundle extras = getIntent().getExtras(); *//All activities are started with an Intent!*

String value = extras.getString("key");

*//kotlin*

**val** extras = intent.extras *//All Activities are started with an Intent! (so have that property)*

**val** value = extras.getString("key")

Таким образом, мы можем заставить Activity общаться и даже обмениваться информацией между собой! Ура!

**7.2 Намерения для другого приложения (неявные)**

Мы можем отправлять Intent в наши собственные Activity, но мы также можем адресовать их *другим* приложениям. При вызове других приложений мы обычно используем **Implicit Intent** .

* Это немного похоже на письма со [странными адресами](http://www.theguardian.com/world/2015/jul/18/postman-turns-detective-to-deliver-letter-with-cryptic-address-in-ireland) , но которые все равно доставляются. «Для того парня в конце квартала с красным почтовым ящиком».

Неявное намерение включает в себя **Действие** и некоторые **Данные** . ***Действие*** говорит, что цель должна *сделать* с намерением (Командой), а ***Данные*** дают более подробную информацию о том, на чем следует запустить это действие.

* **Действия** могут быть такими, как ACTION\_VIEWпросмотр некоторых данных или ACTION\_PICKвыбор элемента из списка. Полный список см. в разделе [«Стандартные действия»](https://developer.android.com/reference/android/content/Intent.html)[21](https://info448.github.io/intents.html#fn21) .
	+ ACTION\_MAINявляется наиболее распространенным (просто запустите Activity, как если бы это была «главная» точка запуска). Поэтому, когда мы не указываем ничего другого, используется это!
* **Данные** содержат подробную информацию о том, что делать с действием (например, Uri to VIEWили Contact to DIAL).
	+ Дополнительные данные подтверждают эти данные!

Например, если мы указываем DIALдействие, то мы говорим, что хотим, чтобы наше намерение было доставлено приложению, способному набрать телефонный номер.

* *Если есть более одного приложения, поддерживающего это действие, пользователю нужно будет выбрать одно!* Это ключ: мы не говорим, какое именно приложение использовать, а только то, какая функциональность нам нужна для поддержки! Это своего рода абстракция!

*//java*

Intent intent = **new** Intent(Intent.ACTION\_DIAL);

intent.setData(Uri.parse("tel:206-685-1622"));

**if** (intent.resolveActivity(getPackageManager()) != **null**) {

 startActivity(intent);

}

*//kotlin*

**val** intent = Intent(Intent.ACTION\_DIAL)

intent.**data** = Uri.parse("tel:206-685-1622")

**if** (intent.resolveActivity(packageManager) != **null**) {

 startActivity(intent)

}

Здесь мы указали *Действие* ( ACTION\_DIAL) для нашего Намерения, а также некоторые *Данные* (номер телефона, преобразованный в Uri). resolveActivity()Метод ищет, какое Действие получит наше действие — мы проверяем, что оно не равно нулю, прежде чем пытаться запустить его.

* Это должно позволить нам «дозвониться»!

Обратите внимание, что мы можем открывать все виды приложений. См. [Common Intents](https://developer.android.com/guide/components/intents-common.html)[22](https://info448.github.io/intents.html#fn22) для списка общих неявных событий (с полезными примерами!).

**7.3 Намерения для ответа**

Мы использовали Intents для запуска Activity, но что, если мы хотим получить результат *из* Activity? То есть, что, если мы хотим найти Contact или сделать фотографию, а затем иметь возможность использовать номер Contact или показать фотографию?

Для этого мы создадим Intent таким же образом, но используем другой метод для их запуска: startActivityForResult(). Это запустит разрешенную Activity. Но как только это Действие будет завершено, запущенная Activity отправит нам обратно *другой* Intent, на который мы затем сможем отреагировать, чтобы обработать результат.

* Это немного похоже на примечание к письму: «Срочно, пожалуйста, ответьте!»

Ради интереса давайте сделаем это с помощью камеры — [запустим камеру,](http://developer.android.com/training/camera/photobasics.html) чтобы сделать снимок, а затем получим снимок и покажем его в приложении ImageView.

* Обратите внимание, что на вашем эмуляторе должна быть включена эмуляция камеры!

*//java*

static final int REQUEST\_IMAGE\_CAPTURE = 1;

**private** void dispatchTakePictureIntent() {

 Intent takePictureIntent = **new** Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

 **if** (takePictureIntent.resolveActivity(getPackageManager()) != **null**) {

 startActivityForResult(takePictureIntent, REQUEST\_IMAGE\_CAPTURE);

 }

}

*//kotlin*

**private** **val** REQUEST\_IMAGE\_CAPTURE = 1

**fun** dispatchTakePictureIntent() {

 **val** takePictureIntent = Intent(MediaStore.ACTION\_IMAGE\_CAPTURE)

 **if** (takePictureIntent.resolveActivity(packageManager) != **null**) {

 startActivityForResult(takePictureIntent, REQUEST\_IMAGE\_CAPTURE)

 }

}

* Мы начинаем с указания Intent, который использует MediaStore.ACTION\_IMAGE\_CAPTUREдействие (действие для «сделать неподвижное изображение и вернуть его»). Затем мы отправим это Intent для получения результата, аналогично другим неявным Intent.
* Мы также включаем второй аргумент: «код запроса». Код запроса используется для того, чтобы отличить это намерение от других, которые мы можем отправить (что-то вроде тега). Он дает нам знать, что Result Intent («RSVP») относится к нашему письму «сделал снимок», а не к какому-то другому отправленному нами Intent.
* Обратите внимание, что мы могли бы передать Extra для того, куда мы хотим сохранить большой файл изображения. Однако мы собираемся оставить это и работать только с миниатюрой для этой демонстрации. Подробности см. в руководстве [23.](http://developer.android.com/training/camera/photobasics.html#TaskPath)[Эта](https://info448.github.io/intents.html#fn23) тема также рассматривается в более поздней главе.

Чтобы обработать Intent «response», нам нужно предоставить обратный вызов, который будет выполнен, когда прибудет Intent. Этот обратный вызов называется onActivityResult().

*//java*

@Override

**protected** void onActivityResult(int requestCode, int resultCode, Intent data) {

 **if** (requestCode == REQUEST\_IMAGE\_CAPTURE && resultCode == RESULT\_OK) {

 Bundle extras = data.getExtras();

 Bitmap imageBitmap = (Bitmap) extras.get("data");

 ImageView imageView = (ImageView) findViewById(R.id.img\_thumbnail);

 imageView.setImageBitmap(imageBitmap); *//show the image*

 }

}

*//kotlin*

**override** **fun** onActivityResult(requestCode: Int, resultCode: Int, **data**: Intent) {

 **if** (requestCode == REQUEST\_IMAGE\_CAPTURE && resultCode == Activity.RESULT\_OK) {

 **val** extras = **data**.extras?.apply {

 **val** imageBitmap = **get**("data") **as** Bitmap

 **val** imageView = findViewById<ImageView>(R.id.img\_thumbnail)

 imageView.setImageBitmap(imageBitmap)

 }

 }

}

* Мы можем получить информацию о Intent, который мы получаем из params, включая то, какой Intent привел к результату ( requestCode), и статус этого запроса ( *result*Code). Мы можем получить доступ к возвращенным данным (например, изображению), получив "data"значение из extras.
* Обратите внимание, что это [Bitmap](http://developer.android.com/reference/android/graphics/Bitmap.html), который является классом Android, представляющим растровое изображение. Мы поиграем с Bitmaps больше через пару недель, потому что мне нравится графика.

**7.4 Прослушивание намерений**

Мы можем отправлять неявные намерения, которые могут быть услышаны другими приложениями, но что, если мы хотим получать неявные намерения сами? Что, если *мы* хотим иметь возможность обрабатывать набор номера?!

Чтобы получить неявное намерение, нам нужно объявить, что наша Activity способна обработать этот запрос. Поскольку мы указываем аспект нашего приложения (о котором должны знать другие приложения), мы сделаем это в использовании Manifestтого, что называется **<intent-filter>**.

* Идея в том, что мы «слышим» все намерения и «фильтруем» те, которые имеют отношение к нам. Как сортировка нежелательной почты.

Тег <intent-filter>вложен в элемент, к которому он применяется (например, <activity>). Фактически, вы можете видеть, что там уже есть один: который отвечает на MAINдействие, отправленное с LAUNCHERкатегорией (то есть он отвечает на намерения из средства запуска приложения).

Аналогично мы можем указать три «части» фильтра:

* фильтр <action android:name="action">, описывающий действие, на которое мы можем отреагировать.
* фильтр <data ...>, который определяет аспекты данных, которые мы принимаем (например, отвечать только на те URI, которые выглядят как телефонные номера)
* фильтр <category android:name="category">, который по сути является частью «больше информации». Вы можете увидеть [«Стандартные категории»](http://developer.android.com/reference/android/content/Intent.html) в документации.
* Обратите внимание, что вы *должны* включить DEFAULTкатегорию для получения неявных намерений. Это категория, используемая startActivity()и startActivityForResult().

Вы можете включить несколько действий, данных и тегов категории. Вам просто нужно убедиться, что вы можете обрабатывать все возможные комбинации, выбранные из каждого типа (несколько типов объединяются с помощью «или», а не «и»).

В ответ на эту команду набора номера:

**<activity** android:name=".SecondActivity"**>**

 **<intent-filter>**

 **<action** android:name="android.intent.action.DIAL"**/>**

 **<category** android:name="android.intent.category.DEFAULT" **/>**

 **<data** android:scheme="tel" **/>** *<!-- telephone Uris -->*

 **</intent-filter>**

**</activity>**

Еще больше примеров вы можете увидеть в [Intent](http://developer.android.com/reference/android/content/Intent.html)документации.

**7.5 Трансляции и приемники**

Есть еще один тип Intent, о котором я хочу поговорить сегодня: [**Broadcasts**](http://developer.android.com/reference/android/content/BroadcastReceiver.html) . Broadcast — это сообщение, которое может получить *любое* приложение. В отличие от Explicit и Implicit Intents, broadcasts слышны всей системе — все, что вы «кричите» с помощью broadcast, доступно всем (так что подумайте о проблемах безопасности при их использовании!)

* Если продолжить метафору с почтой-улиткой, то это, возможно, массовые рассылки.

За исключением того, кто их получает, трансляции работают так же, как и обычные неявные намерения! Мы создаем Intentс действием и данными (и категорией и дополнительными материалами…). Но вместо использования startActivity()метода мы используем sendBroadcast()метод. Теперь это намерение могут услышать все Activitiesна телефоне.

* Мы пропустим демонстрацию из-за нехватки времени и мотивации… но мы начнем создавать трансляции позже в ходе курса.

Для нас более распространенным, чем отправка трансляций, будет *прием* трансляций; то есть мы хотим прослушивать и отвечать на трансляции Системы, которые производятся (такие вещи, как события питания, статус Wi-Fi и т. д.). Это также будет включать прослушивание таких вещей, как входящие телефонные звонки или текстовые сообщения!

Мы можем получать трансляции, используя [BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html). Это базовый класс, который используется любым классом, который может получать намерения трансляций. Мы **создаем подкласс** и реализуем onReceive(Context, Intent)обратный вызов для обработки при получении трансляций.

*//java*

**public** void onReceive(Context context, Intent intent) {

 Log.v("TAG", "received! "+intent.toString());

 **if**(intent.getAction() == Intent.ACTION\_BATTERY\_LOW){

 Toast.makeText(context, "Battery is low!", Toast.LENGTH\_SHORT).show();

 }

}

*//kotlin*

**override** **fun** onReceive(context: Context, intent: Intent) {

 Log.v("TAG", "received! ${intent.toString()}")

 **when**(intent.action) {

 Intent.ACTION\_BATTERY\_LOW -> {

 Toast.makeText(context, "Battery is low!", Toast.LENGTH\_SHORT).show();

 }

 }

}

* Хотя для этого мы можем использовать Android Studio, мы сделаем это вручную, чтобы увидеть все шаги.

Но для того, чтобы **зарегистрировать** наш приемник (чтобы Intents прошли мимо его стола), нам также нужно указать его в Manifest. Мы делаем это, включая <receiver>элемент внутрь нашего <application>. Обратите внимание, что это *не* Activity, а отдельный компонент! Мы можем поместить <intent-filter>внутрь этого , чтобы отфильтровать трансляции, которые нам интересны.

**<receiver** android:name=".PowerReceiver"**>**

 **<intent-filter>**

 **<action** android:name="android.intent.action.ACTION\_POWER\_CONNECTED" **/>**

 **<action** android:name="android.intent.action.ACTION\_POWER\_DISCONNECTED" **/>**

 **<action** android:name="android.intent.action.BATTERY\_CHANGED" **/>**

 **<action** android:name="android.intent.action.BATTERY\_OKAY" **/>**

 *<!-- no category because not for an activity! -->*

 **</intent-filter>**

**</receiver>**

Начиная с Oreo, регистрация *неявных* трансляций в Manifest [не допускается](https://developer.android.com/about/versions/oreo/background#broadcasts) и не будет приниматься правильно, за некоторыми [исключениями](https://developer.android.com/guide/components/broadcast-exceptions) . Вместо этого вам нужно будет зарегистрировать приемники в коде (описано ниже). Обратите внимание, что установка значения targetSdkменее 25 является временным решением.

Мы можем легко проверить эти события питания с помощью эмулятора. На кнопке «дополнительные параметры» (три точки внизу) на панели инструментов эмулятора мы можем перейти на Batteryвкладку, где мы можем эффективно изменить состояние батареи устройства (на которое наше приложение может реагировать!). Не забудьте сначала «отключиться» от зарядного устройства переменного тока!

Мы также можем *зарегистрировать* эти приемники в коде (а не в манифесте). Это хорошо, если мы хотим только временно прослушивать какие-то события или если мы хотим определять их intent-filterна лету:

*//java*

IntentFilter batteryFilter = **new** IntentFilter();

batteryFilter.addAction(Intent.ACTION\_BATTERY\_LOW);

batteryFilter.addAction(Intent.ACTION\_BATTERY\_OKAY);

batteryFilter.addAction(Intent.ACTION\_POWER\_CONNECTED);

batteryFilter.addAction(Intent.ACTION\_POWER\_DISCONNECTED);

**this**.registerReceiver(**new** PowerReceiver(), batteryFilter);

*//kotlin*

**val** batteryFilter = IntentFilter()

batteryFilter.addAction(Intent.ACTION\_BATTERY\_LOW)

batteryFilter.addAction(Intent.ACTION\_BATTERY\_OKAY)

batteryFilter.addAction(Intent.ACTION\_POWER\_CONNECTED)

batteryFilter.addAction(Intent.ACTION\_POWER\_DISCONNECTED)

**this**.registerReceiver(PowerReceiver(), batteryFilter)

* Мы intent-filterтакже динамически объявляем! Это можно использовать не только для BroadcastReceivers, но Activitiesи также, если вы хотите динамически изменить, может ли Activity получать Intent (например, иногда я могу совершать телефонные звонки, а иногда нет).

**7.6 Меню**

Намерения позволяют нам выполнять все виды коммуникаций и действий. Но для управления всеми этими опциями было бы полезно ввести несколько дополнительных компонентов пользовательского интерфейса — и поэтому в этом разделе мы поговорим о [***Меню***](https://developer.android.com/guide/topics/ui/menus.html) .

Меню в основном доступны через [**App Bar,**](https://developer.android.com/training/appbar/index.html) также известную как **Action Bar** . Это своего рода «заголовок» для вашего приложения, предоставляющий выделенное пространство для навигации и взаимодействия (например, через меню).

* 24 — это особый тип, [который](https://info448.github.io/intents.html#fn24) чаще всего используется в качестве панели приложений, предлагая определенный «внешний вид», общий для приложений Android. По этой причине «панель приложений» и «панель действий» часто используются взаимозаменяемо. Однако можно включать панели инструментов (и, соответственно, меню) в любое место макета. Подробнее см. в [разделе Настройка панели приложений .](http://developer.android.com/training/appbar/setting-up.html)[ActionBar](https://developer.android.com/reference/android/support/v7/app/ActionBar.html)[Toolbar](https://developer.android.com/reference/android/support/v7/widget/Toolbar.html)

Основное применение панели действий — место для хранения [***меню***](https://developer.android.com/guide/topics/ui/menus.html) . Меню (в частности, [**меню опций**](https://developer.android.com/guide/topics/ui/menus.html#options-menu) ) — это набор элементов (например, кнопок), которые отображаются на панели действий. Меню могут быть указаны как в , Activityтак и в Fragment; если они объявлены в обоих местах, они объединяются в одно меню на панели действий. Это позволяет вам легко создавать «контекстно-зависимые» меню опций, которые доступны только для соответствующего фрагмента, при этом фрагменты остаются модульными и самодостаточными.

* Чтобы предоставить фрагменту меню, вызовите метод setHasOptionsMenu(true)фрагмента onCreate().
* *Интересный факт:* до API 11 меню опций отображалось в виде кнопок в нижней части экрана!

Меню, как и все другие элементы, с которыми сталкивается пользователь, определяются как ресурсы XML, в частности, типа **menu**. Вы можете создать новый ресурс меню через Android studio, используя File > New > Android resource file, а затем выбрав Menuтип ресурса. Это создаст файл XML с основным [<menu>](https://developer.android.com/reference/android/view/Menu.html)элементом.

Опции можно добавлять в меню, указывая дочерние элементы XML, в частности [<item>](https://developer.android.com/reference/android/view/MenuItem.html)элементы. Общие <item>атрибуты включают:

* **android:id**: уникальный идентификатор, используемый для ссылки на определенную опцию в коде Java
* **android:title**( **обязательный** атрибут): текст для отображения для опции. Как текст, видимый пользователю, содержимое в идеале должно быть определено как ресурс XML String.
* **app:showAsAction**: следует ли отображать опцию в панели действий или сворачивать ее под кнопкой «три точки». Обратите внимание, что при работе с appcompatбиблиотекой эта опция использует appпространство имен (вместо android); вам нужно будет включить это пространство имен в <menu>с атрибутом xmlns:app="http://schemas.android.com/apk/res-auto".
* **android:icon**: изображение, используемое при отображении опции в виде кнопки в меню.

Вы можете использовать один из множества значков, встроенных в Android, который обозначается как "@android:drawable/ic\_\*". Полный список содержится в [Android Drawables](http://androiddrawables.com/)[25](https://info448.github.io/intents.html#fn25) , хотя не все элементы рисования доступны публично через Android Studio.

* **android:orderInCategory**: используется для упорядочивания элементов в меню (или в группе). Это действует как «приоритет» (по умолчанию 0; сначала идет низкий). Такая приоритезация может быть полезна, если вы хотите добавить предложения о том, должны ли параметры фрагмента располагаться до или после параметров активности.

Полный список опций см. [в руководстве по ресурсам меню](https://developer.android.com/guide/topics/resources/menu-resource.html)[26 !](https://info448.github.io/intents.html#fn26)

Можно включить **один уровень** подменю ( <menu>элемент внутри <item>элемента). Элементы меню также можно группировать, помещая их внутрь элемента <group>. Все элементы в группе будут отображаться или скрываться вместе, и их можно дополнительно упорядочить внутри этой группы. Сгруппированные значки также можно сделать [отмечаемыми](https://developer.android.com/guide/topics/ui/menus.html#checkable) .

Чтобы отобразить меню в запущенном приложении, нам нужно указать Action Bar, какой ресурс меню он должен использовать (ресурсов может быть много). Для этого мы переопределяем onCreateOptionsMenu()обратный вызов в Activity или Fragment, а затем используем объект компонента MenuInflaterдля раскрытия меню:

*//java*

**public** boolean onCreateOptionsMenu(Menu menu) {

 MenuInflater inflater = getMenuInflater();

 inflater.inflate(R.menu.main\_menu, menu); *//inflate into this menu*

 **return** **true**;

}

*//kotlin*

**override** **fun** onCreateOptionsMenu(menu: Menu): Boolean {

 **val** inflater = menuInflater

 inflater.inflate(R.menu.main\_menu, menu) *//inflate into this menu*

 **return** **true**

}

* Эта процедура концептуально похожа на то, как метод фрагмента onViewCreated()раздувает фрагмент в активность. В этом случае меню раздувается в панель действий.

Мы можем реагировать на выбор пунктов меню, переопределяя onOptionsItemSelected()обратный вызов. По соглашению в Java мы используем a switchв , item.getItemId()чтобы определить, какой пункт был выбран, а затем действуем соответствующим образом. В Kotlin используйте whenвыражение, которое будет возвращено.

*//java*

**public** boolean onOptionsItemSelected(MenuItem item) {

 **switch**(item.getItemId()){

 **case** R.id.menu\_item1 :

 *//do thing;*

 **return** **true**;

 **default**:

 **return** **super**.onOptionsItemSelected(item);

 }

}

*//kotlin*

**override** **fun** onOptionsItemSelected(item: MenuItem): Boolean {

 **return** **when** (item.itemId) {

 R.id.menu\_item1 -> {

 *//do thing*

 **true**

 }

 **else** -> **super**.onOptionsItemSelected(item)

 }

}

* ( defaultЕсли выбранный элемент не обрабатывается ни одним случаем), мы передаем обратный вызов наверх superдля проверки компонентов «более высокого уровня». Например, если опция меню не обрабатывается фрагментом (потому что фрагмент его не добавил), событие может быть передано вверх через фреймворк для последующей обработки действием (которое его добавило).
* Этот метод должен возвращать, trueесли выбор был обработан (и, таким образом, не должен рассматриваться кем-либо еще). Верните, falseесли вы хотите, чтобы другие компоненты (например, другие фрагменты) также могли реагировать на эту опцию.

Есть много других пунктов меню, которые также можно разместить на панели действий. Мы также можем добавить [представления действий](https://developer.android.com/training/appbar/action-views.html) , которые обеспечивают более сложные взаимодействия, чем просто нажатие кнопок (например, включая строку поиска). [Поставщик действий](https://developer.android.com/training/appbar/action-views.html#action-provider) (например [ShareActionProvider](https://developer.android.com/training/sharing/shareaction.html), ) — это действие с собственным настраиваемым макетом, которое при щелчке разворачивается в отдельное представление.

**Действия Просмотры**

В дополнение к этим простым кнопкам мы также можем включать более сложные расширяемые виджеты, называемые [Action Views](https://developer.android.com/training/appbar/action-views.html) . Наиболее распространенным из них является [SearchView](https://developer.android.com/reference/android/support/v7/widget/SearchView.html) , который помещает «поле поиска» в App Bar!

Однако в качестве немного более простой демонстрации мы настроим [ShareActionProvider](https://developer.android.com/training/sharing/shareaction.html), который автоматически предоставляет раскрывающийся элемент меню с полным списком приложений, с которыми мы можем поделиться нашими данными (например, кнопка «быстрый обмен с этими сайтами социальных сетей»).

* Для справки я рекомендую посмотреть [документацию класса](http://developer.android.com/reference/android/support/v7/widget/ShareActionProvider.html) , чтобы узнать, как это настроить (она гораздо понятнее, чем обучающие документы).

Начнем с добавления еще одного элемента в XML нашего меню. Он будет выглядеть как другие элементы, за исключением того, что у него будет дополнительное поле, app:actionProviderClassуказывающее, что этот элемент является поставщиком действий (виджет, который предоставляет некоторые действия)

**<item**

 android:id="@+id/menu\_item\_share"

 android:title="Share"

 app:showAsAction="ifRoom"

 app:actionProviderClass="android.support.v7.widget.ShareActionProvider"

 **/>**

Затем мы добавим этот пункт в наше меню.onCreateOptionsMenu()

*//java*

MenuItem item = menu.findItem(R.id.menu\_item\_share);

ShareActionProvider shareProvider = (ShareActionProvider)MenuItemCompat.getActionProvider(item);

Intent intent = **new** Intent(Intent.ACTION\_DIAL);

intent.setData(Uri.parse("tel:206-685-1622"));

shareProvider.setShareIntent(intent);

*//kotlin*

**val** item = menu.findItem(R.id.menu\_item\_share)

**val** shareProvider = MenuItemCompat.getActionProvider(item) **as** ShareActionProvider

**val** intent = Intent(Intent.ACTION\_DIAL)

intent.**data** = Uri.parse("tel:206-685-1622")

shareProvider.setShareIntent(intent)

* Мы получаем доступ к элементу с помощью findItem()(аналогично findViewById()), а затем преобразуем его в ShareActonProvider(убедитесь, что вы используете поддерживаемую версию!)
* Затем мы можем указать *неявное намерение* , которое мы хотим, чтобы эта «кнопка «Поделиться»» могла выполнять. Обычно это будет использовать ACTION\_SENDдействие (например, для обмена изображением или текстом), но мы будем использовать DIALдействие, потому что у нас есть несколько дозвонщиков, но на самом деле нет много SENDответчиков на эмуляторе.

Затем в пункте меню будет выпадать список со всеми различными действиями, которые resolveобрабатывают это неявное намерение! И теперь мы глубоко объединили намерения и меню!

**Контекстные меню**

В дополнение к меню опций, доступным в панели действий, мы также можем указать [контекстные меню](https://developer.android.com/guide/topics/ui/menus.html#context-menu) , которые всплывают, когда пользователь долго нажимает на элемент. Это работает аналогично использованию меню опций, но с другим набором обратных вызовов:

* При настройке макета представления (например, в Activity onCreate()) мы указываем, что у элемента есть контекстное меню, используя registerForContextMenu()метод, передавая ему параметр, Viewдля которого мы хотим создать меню.
* Укажите контекстное меню для использования через onCreateContextMenu()обратный вызов. Это работает точно так же, как настройка меню опций.
	+ Фактически, контекстное меню может даже использовать *то же меню,* что и меню опций! Это повторное использование является одним из преимуществ определения пользовательского интерфейса как XML.
* И, отражая меню опций, реагировать на выбор пунктов контекстного меню с помощью onContextItemSelected()обратного вызова.

В этом разделе представлено очень краткое введение в меню, но есть много более сложных взаимодействий, которые они поддерживают. Я *настоятельно* рекомендую вам прочитать [руководство](https://developer.android.com/guide/topics/ui/menus.html) , чтобы узнать, какие функции могут быть доступны.

Если вы когда-нибудь будете использовать приложение и задаетесь вопросом «как они добавили эту функцию интерфейса?», поищите! Почти всегда есть задокументированная процедура и пример предоставления такого рода компонента.

**7.7 Пример намерения: SMS**

Дополнительный пример для любопытных.

Одно из конкретных применений Intent — работа с текстовыми сообщениями (SMS, Short Messaging Service, самая популярная форма передачи данных в мире). В частности, Intent используются для отправки и получения SMS-сообщений (вы можете получить список уже полученных сообщений с помощью ContentProvided, описанного в следующей лекции).

* *Важное примечание:* API SMS *кардинально* изменились в KitKat (API 19). Поэтому мы собираемся убедиться, что это наш минимум, чтобы мы могли получить все полезные методы и поддерживать новые вещи (проверьте gradle, чтобы убедиться!).

Главное, что следует отметить об отправке SMS, это то, что с KitKat каждая система имеет клиент сообщений *по умолчанию* — это единственное приложение, которое может отправлять сообщения. К счастью, API позволяет вам получить доступ к сервисам этого клиента сообщений, чтобы отправить сообщение *через* него:

SmsManager smsManager = SmsManager.getDefault();

smsManager.sendTextMessage("5554", **null**, "This is a test message!", **null**, **null**);

*// target, message*

* 5554— это «номер телефона» эмулятора по умолчанию (порт, на котором он работает).

Нам также понадобится разрешение: <uses-permission android:name="android.permission.SEND\_SMS" />(в манифесте). Обратите внимание, что вам нужно будет использовать SDK 22 или ниже, чтобы это работало без дополнительных подробностей; см. лекцию о разрешениях для получения дополнительной информации.

Вы можете увидеть, что это работает, посмотрев на входящие в приложении Сообщения... но есть и другой способ! Если вы посмотрите [документацию по этому методу](http://developer.android.com/reference/android/telephony/SmsManager.html#sendTextMessage(java.lang.String,%20java.lang.String,%20java.lang.String,%20android.app.PendingIntent,%20android.app.PendingIntent))[27](https://info448.github.io/intents.html#fn27) , вы увидите, что последние два параметра предназначены для [PendingIntents](http://developer.android.com/reference/android/app/PendingIntent.html): один для отправки сообщений и один для доставки сообщений.

* Что такое PendingIntent? Детали не *очень* читабельны… По сути, это обертка вокруг , Intentкоторую мы передаем **другому** классу. Затем, когда этот класс получает наш PendingIntentи реагирует на него, он может выполнить Intent(команду), которую мы ему отправили, как будто это Activityмы (ух ты).
	+ Это как если бы вы попросили у профессора рекомендательное письмо и дали бы ему конверт с маркой (что вы всегда должны делать!). Когда они получат посылку, они могут отправить свое письмо, используя *ваш* конверт и марку!
	+ Альтернативный вариант: «Я отправлю тебе ключи от машины по почте, когда получишь их, можешь приехать и забрать меня».
* Итак, идея в том, что мы указываем, что Intentдолжно быть доставлено, когда сообщение будет отправлено (что Intentстановится «ожидающим»). По сути, это позволяет нам отправлять намерения в ответ на какой-то другой тип событий.

Давайте продолжим и настроим его:

**public** static final String ACTION\_SMS\_STATUS = "edu.uw.intentdemo.ACTION\_SMS\_STATUS";

*//...*

Intent intent = **new** Intent(ACTION\_SMS\_STATUS);

PendingIntent pendingIntent = PendingIntent.getBroadcast(MainActivity.this, 0, intent, 0);

smsManager.sendTextMessage("5554", **null**, "This is a test message!", pendingIntent, **null**);

Здесь мы делаем несколько шагов:

* Мы определяем наше собственное действие. Это просто String, но с пространством имен, чтобы избежать конфликтов
* Затем мы создаем **неявное намерение** для этого действия.
* И затем создайте PendingIntent. Мы используем getBroadcast()метод, чтобы указать, что Intent должен быть отправлен через Broadcast (см getActivity(). startActivity()).
	+ Первый параметр — это контекст, который должен отправить намерение, затем код запроса (например, для обратных вызовов результата, если мы этого хотим), затем Intentжелаемое ожидание и, наконец, любые дополнительные флаги (пока их нет).

И тогда мы сможем BroadcastReceiverотреагировать на это Intentтак же, как и на любое другое!

**if**(intent.getAction() == MainActivity.ACTION\_SMS\_STATUS) {

 **if** (getResultCode() == Activity.RESULT\_OK) {

 Toast.makeText(context, "Message sent!", Toast.LENGTH\_SHORT).show();

 }

 **else** {

 Toast.makeText(context, "Error sending message", Toast.LENGTH\_SHORT).show();

 }

}

* **Не забудьте** добавить наше пользовательское намерение в <intent-filter>!

Подробнее об этом мы поговорим PendingIntentsв следующей главе, когда поговорим об уведомлениях.

1. <https://developer.android.com/reference/android/content/Intent.html>[↩](https://info448.github.io/intents.html#fnref21)
2. [https://developer.android.com/guide/comComponents/intents-common.html](https://developer.android.com/guide/components/intents-common.html)[↩](https://info448.github.io/intents.html#fnref22)
3. <http://developer.android.com/training/camera/photobasics.html#TaskPath>[↩](https://info448.github.io/intents.html#fnref23)
4. <http://developer.android.com/reference/android/support/v7/app/ActionBar.html>[↩](https://info448.github.io/intents.html#fnref24)
5. <http://androiddrawables.com/>[↩](https://info448.github.io/intents.html#fnref25)
6. <https://developer.android.com/guide/topics/resources/menu-resource.html>[↩](https://info448.github.io/intents.html#fnref26)
7. <https://developer.android.com/reference/android/telephony/SmsManager.html>[↩](https://info448.github.io/intents.html#fnref27)