Activity – жизненный цикл

Содержание

- Операции
- Стек переходов назад
- Управление задачами
- Реализация пользовательского интерфейса
- Задание макета
- Объявление операции в манифесте
- Использование фильтров намерений
- Управление жизненным циклом операций
- Реализация обратных вызовов жизненного цикла
- Обработка изменений в конфигурации

Операции

Activity — это компонент приложения, который выдает экран.

Каждой операции присваивается окно для прорисовки соответствующего пользовательского интерфейса.

Обычно окно отображается во весь экран, однако его размер может быть меньше, и оно может размещаться поверх других окон.

При запуске новой операции она помещается в стек переходов назад и новая операция отображается для пользователя.

Стек переходов назад работает по принципу «последним вошёл — первым вышел», поэтому после того как пользователь завершил текущую операцию и нажал кнопку *Назад*, текущая операция удаляется из стека (и уничтожается), и возобновляется предыдущая операция.

Стек переходов назад

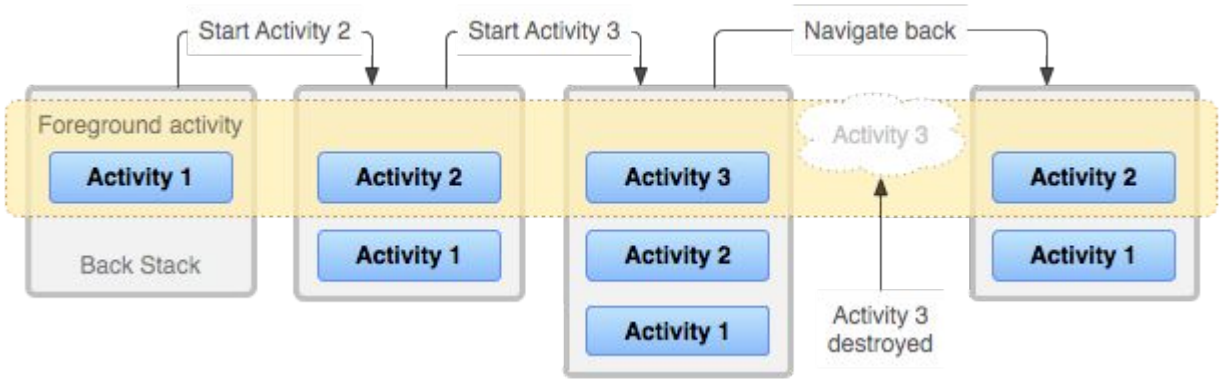
Когда текущая операция запускает другую, новая операция помещается в вершину стека и получает фокус.

Предыдущая операция остается в стеке, но ее выполнение останавливается.

Когда операция останавливается , система сохраняет текущее состояние ее пользовательского интерфейса.

Когда пользователь нажимает кнопку *Назад*, текущая операция удаляется из вершины стека (операция уничтожается) и возобновляется работа предыдущей операции (восстанавливается предыдущее состояние ее пользовательского интерфейса).

Операции в стеке никогда не переупорядочиваются, только добавляются в стек и удаляются из него — добавляются в стек при запуске текущей операцией и удаляются, когда пользователь выходит из нее с помощью кнопки *Назад*.



Управление задачами

Для большинства приложений способ, которым Android управляет задачами и стеком переходов назад, описанный выше, — помещение всех операций последовательно в одну задачу в стек «последним вошёл — первым вышел», — работает хорошо,

Однако вы можете решить, что вы хотите прервать обычное поведение.

Вы можете совершать это с помощью атрибутов в элементе манифеста <activity> и с помощью флагов в намерении, которое вы передаете в startActivity().

В этом смысле главными атрибутами <activity> которые вы можете использовать, являются следующие:

taskAffinity

launchMode

allowTaskReparenting

clearTaskOnLaunch

alwaysRetainTaskState

finishOnTaskLaunch

А главными флагами намерений, которые вы можете использовать, являются следующие:

FLAG_ACTIVITY_NEW_TASK

FLAG_ACTIVITY_CLEAR_TOP

Тэг <activity> с атрибутами

```
<activity android:allowEmbedded=["true" | "false"]
          android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:autoRemoveFromRecents=["true" | "false"]
          android:banner="drawable resource"
          android:clearTaskOnLaunch=["true" | "false"]
          android:colorMode=[ "hdr" | "wideColorGamut"]
          android:configChanges=["mcc", "mnc", "locale",
                    "touchscreen", "keyboard", "keyboardHidden",
                    "navigation", "screenLayout", "fontScale",
                    "uiMode", "orientation", "density",
                    "screenSize", "smallestScreenSize"]
…….
</activity>
```

Создание операции

Чтобы создать операцию, сначала необходимо создать подкласс класса Activity (или воспользоваться существующим его подклассом).

В таком подклассе необходимо реализовать методы обратного вызова, которые вызывает система при переходе операции из одного состояния своего жизненного цикла в другое, например при создании, остановке, возобновлении или уничтожении операции.

Вот два наиболее важных метода обратного вызова:

- onCreate()

Этот метод необходимо обязательно реализовать, поскольку система вызывает его при создании вашей операции. В своей реализации вам необходимо инициализировать ключевые компоненты операции.

Наиболее важно именно здесь вызвать setContentView() для определения макета пользовательского интерфейса операции.

- onPause()

Система вызывает этот метод в качестве первого признака выхода пользователя из операции (однако это не всегда означает, что операция будет уничтожена).

Реализация пользовательского интерфейса

Для реализации пользовательского интерфейса операции используется иерархия представлений—объектов, полученных из класса View.

Каждое представление отвечает за определенную область окна операции и может реагировать на действия пользователей. Например, представлением может быть кнопка, нажатие на которую приводит к выполнению определенного действия.

В Android предусмотрен набор уже готовых представлений, которые можно использовать для создания дизайна макета и его организации.

- Виджеты — это представления с визуальными (и интерактивными) элементами, например, кнопками, текстовыми полями, чекбоксами или просто изображениями.
- Макеты — это представления, полученные из класса ViewGroup, обеспечивающие уникальную модель компоновки для своих дочерних представлений, таких как линейный макет, сетка или относительный макет. Также можно создать подкласс для классов View и ViewGroup (или воспользоваться существующими подклассами), чтобы создать собственные виджеты и макеты, и затем применить их к макету своей операции.

Задание макета

Чаще всего для задания макета с помощью представлений используется XML-файл макета, сохраненный в ресурсах приложения.

Чтобы задать макет в качестве пользовательского интерфейса операции, можно использовать метод setContentView(), передав в него идентификатор ресурса для макета.

Однако вы также можете создать новые View в коде вашей операции и создать иерархию представлений.

Для этого вставьте View в ViewGroup, а затем используйте этот макет, передав корневой объект ViewGroup в метод setContentView().

Объявление операции в манифесте

Чтобы операция стала доступна системе, ее необходимо объявить в файле манифеста. Для этого откройте файл манифеста и добавьте элемент <activity> в качестве дочернего для элемента <application>. Например:

```
<manifest ... >
    <application ... >
        <activity android:name=".ExampleActivity" />
        ...
    </application ... >
    ...
</manifest >
```

Существует несколько других атрибутов, которые можно включить в этот элемент, чтобы определить такие свойства, как метка операции, значок операции или тема оформления для пользовательского интерфейса операции.

Единственным обязательным атрибутом является android:name — он определяет имя класса операции. После публикации вашего приложения вам не следует переименовывать его, поскольку это может нарушить некоторые функциональные возможности приложения, например, ярлыки приложения

Использование фильтров намерений

Элемент <activity> также может задавать различные фильтры намерений — с помощью элемента <intent-filter> — для объявления того, как другие компоненты приложения могут активировать операцию.

При создании нового приложения с помощью инструментов Android SDK в заготовке операции, создаваемой автоматически, имеется фильтр намерений, который объявляет операцию. Эта операция реагирует на выполнение «основного» действия, и ее следует поместить в категорию переход средства запуска.

Фильтр намерений выглядит следующим образом.

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Элемент <action> указывает, что это «основная» точка входа в приложение. Элемент <category>указывает, что эту операцию следует указать в средстве запуска приложений системы (чтобы пользователи могли запускать эту операцию).

Использование фильтров намерений

Если приложение планируется создать самодостаточным и запретить другим приложениям активировать его операции, то других фильтров намерений создавать не нужно.

В этом случае только в одной операции должно иметься «основное» действие, и ее следует поместить в категорию средства запуска, как в примере выше. В операциях, которые не должны быть доступны для других приложений, не следует включать фильтры намерений. Вы можете самостоятельно запустить такие операции с помощью явных намерений.

Однако, если вам необходимо, чтобы операция реагировала на неявные намерения, получаемые от других приложений (а также из вашего приложения), для операции необходимо определить дополнительные фильтры намерений.

Для каждого типа намерения, на который необходимо реагировать, необходимо указать объект <intent-filter>, включающий элемент <action> и необязательный элемент <category> или <data> (или оба этих элемента).

Эти элементы определяют тип намерения, на который может реагировать ваша операция.

Запуск операции

- **Для запуска другой операции достаточно вызвать метод startActivity(), передав в него объект Intent, который описывает запускаемую операцию.**
- **В намерении указывается либо точная операция для запуска, либо описывается тип операции, которую вы хотите выполнить (после чего система выбирает для вас подходящую операцию, которая может даже находиться в другом приложении).**
- **Намерение также может содержать небольшой объем данных, которые будут использоваться запущенной операцией.**
- **При работе с собственным приложением зачастую требуется лишь запустить нужную операцию. Для этого необходимо создать намерение, которое явно определяет требуемую операцию с помощью имени класса.**
- **Ниже представлен пример запуска одной операцией другой операции с именем SignInActivity.**

Может потребоваться выполнить некоторое действие,

В приложении могут отсутствовать такие действия, поэтому вы можете воспользоваться операциями из других приложений, имеющихся на устройстве, которые могут выполнять требуемые действия.

Как раз в этом случае намерения особенно полезны — можно создать намерение, которое описывает необходимое действие, после чего система запускает его из другого приложения. При наличии нескольких операций, которые могут обработать намерение, пользователь может выбирать, какое из них следует использовать.

Например, если пользователю требуется предоставить возможность отправить электронное письмо, можно создать следующее намерение:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Дополнительный компонент EXTRA_EMAIL, добавленный в намерение, представляет собой строковый массив адресов электронной почты для отправки письма.

Когда почтовая программа реагирует на это намерение, она считывает дополнительно добавленный строковый массив и помещает имеющиеся в нем адреса в поле получателя в окне создания письма.

При этом запускается операция почтовой программы, а после того, как пользователь завершит требуемые действия, возобновляется ваша операция.

Запуск операции для получения результата

В некоторых случаях после запуска операции может потребоваться получить результат.

- Для этого вызовите метод startActivityForResult() (вместо startActivity()).
- Чтобы получить результат после выполнения последующей операции, реализуйте метод обратного вызова onActivityResult().
- По завершении последующей операции она возвращает результат в объекте Intent в вызванный метод onActivityResult().

Пусть, пользователю потребуется выбрать один из контактов, чтобы ваша операция могла выполнить некоторые действия с информацией об этом контакте.

Пример создания намерения и обработки результата.

```
private void pickContact() {
        // Create an intent to "pick" a contact, as defined by the content provider URI
        Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
        startActivityForResult(intent, PICK_CONTACT_REQUEST);
    }


@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
        if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
            // Perform a query to the contact's content provider for the contact's name
            Cursor cursor = getContentResolver().query(data.getData(),
            new String[] {Contacts.DISPLAY_NAME}, null, null, null);
            if (cursor.moveToFirst()) { // True if the cursor is not empty
                int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
                String name = cursor.getString(columnIndex);
                // Do something with the selected contact's name…      }    }}
```

Комментарий

В этом примере демонстрируется базовая логика, которой следует руководствоваться при использовании метода onActivityResult() для обработки результата выполнения операции.

Первое условие проверяет, успешен ли запрос, и если он успешен, то результат для resultCode будет RESULT_OK;

также проверяется, известен ли запрос, для которого получен этот результат, и в этом случае requestCode соответствует второму параметру, отправленному в метод startActivityForResult().

Здесь код обрабатывает результат выполнения операции путем запроса данных, возвращенных в Intent (параметр data).

При этом ContentResolver выполняет запрос к поставщику контента, который возвращает объект Cursor, обеспечивающий считывание запрошенных данных.

Завершение операции

Для завершения операции достаточно вызвать ее **метод** finish(). Также для завершения отдельной операции, запущенной ранее, можно вызвать метод finishActivity().

Примечание. В большинстве случаев вам не следует явно завершать операцию с помощью этих методов.

Система Android выполняет такое управление за вас, поэтому вам не нужно завершать ваши собственные операции.

Вызов этих методов может отрицательно сказаться на ожидаемом поведении приложения.

Их следует использовать исключительно тогда, когда вы абсолютно не хотите, чтобы пользователь возвращался к этому экземпляру операции.

Управление жизненным циклом операций

Управление жизненным циклом операций путем реализации методов обратного вызова имеет важное значение при разработке надежных и гибких приложений. На жизненный цикл операции напрямую влияют его связи с другими операциями, его задачами и стеком переходов назад.

Существует всего три состояния операции:

Возобновлена **Операция выполняется на переднем плане экрана и отображается для пользователя. (Это состояние также иногда называется «Выполняется».)**

Приостановлена **На переднем фоне выполняется другая операция, которая отображается для пользователя, однако эта операция по-прежнему не скрыта. То есть поверх текущей операции отображается другая операция, частично прозрачная или не занимающая полностью весь экран. Приостановленная операция полностью активна (объект Activity по-прежнему находится в памяти, в нем сохраняются все сведения о состоянии и информация об элементах, и он также остается связанным с диспетчером окон), однако в случае острой нехватки памяти система может завершить ее.**

Остановлена **Операция полностью перекрывается другой операцией (теперь она выполняется в фоновом режиме).**

Остановленная операция по-прежнему активна (объект Activity по-прежнему находится в памяти, в нем сохраняются все сведения о состоянии и информация об элементах, но объект больше *не* связан с диспетчером окон).

Реализация обратных вызовов жизненного цикла

При переходе операции из одного вышеописанного состояния в другое, уведомления об этом реализуются через различные методы обратного вызова.

Все методы обратного вызова представляют собой привязки, которые можно переопределить для выполнения подходящего действия при изменении состояния операции.

Указанная ниже базовая операция включает каждый из основных методов жизненного цикла.

```
public class ExampleActivity extends Activity {
```

Код обратных вызовов жизненного цикла

```
@Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // The activity is being created.    }
    @Override
    protected void onStart() {
       super.onStart();
       // The activity is about to become visible.    }
    @Override
    protected void onResume() {
       super.onResume();
       // The activity has become visible (it is now "resumed").    }
    @Override
    protected void onPause() {
       super.onPause();
       // Another activity is taking focus (this activity is about to be "paused").    }
    @Override
    protected void onStop() {
       super.onStop();
       // The activity is no longer visible (it is now "stopped“)    }
    @Override
    protected void onDestroy() {
       super.onDestroy();
       // The activity is about to be destroyed.    }
```

Три вложенных цикла в жизненном цикле операции

Весь жизненный цикл операции происходит между вызовом метода onCreate() и вызовом **метода onDestroy()**.

Ваша операция должна выполнить настройку «глобального» состояния (например, определение макета) в методе onCreate(), а затем освободить все оставшиеся в onDestroy() ресурсы.

Например, если в вашей операции имеется поток, выполняющийся в фоновом режиме, для загрузки данных по сети, операция может создать такой поток в методе onCreate(), а затем остановить его в методе onDestroy().

Видимый жизненный цикл операции происходит между вызовами методов onStart() и onStop(). В течение этого времени операция отображается на экране, где пользователь может взаимодействовать с ней.

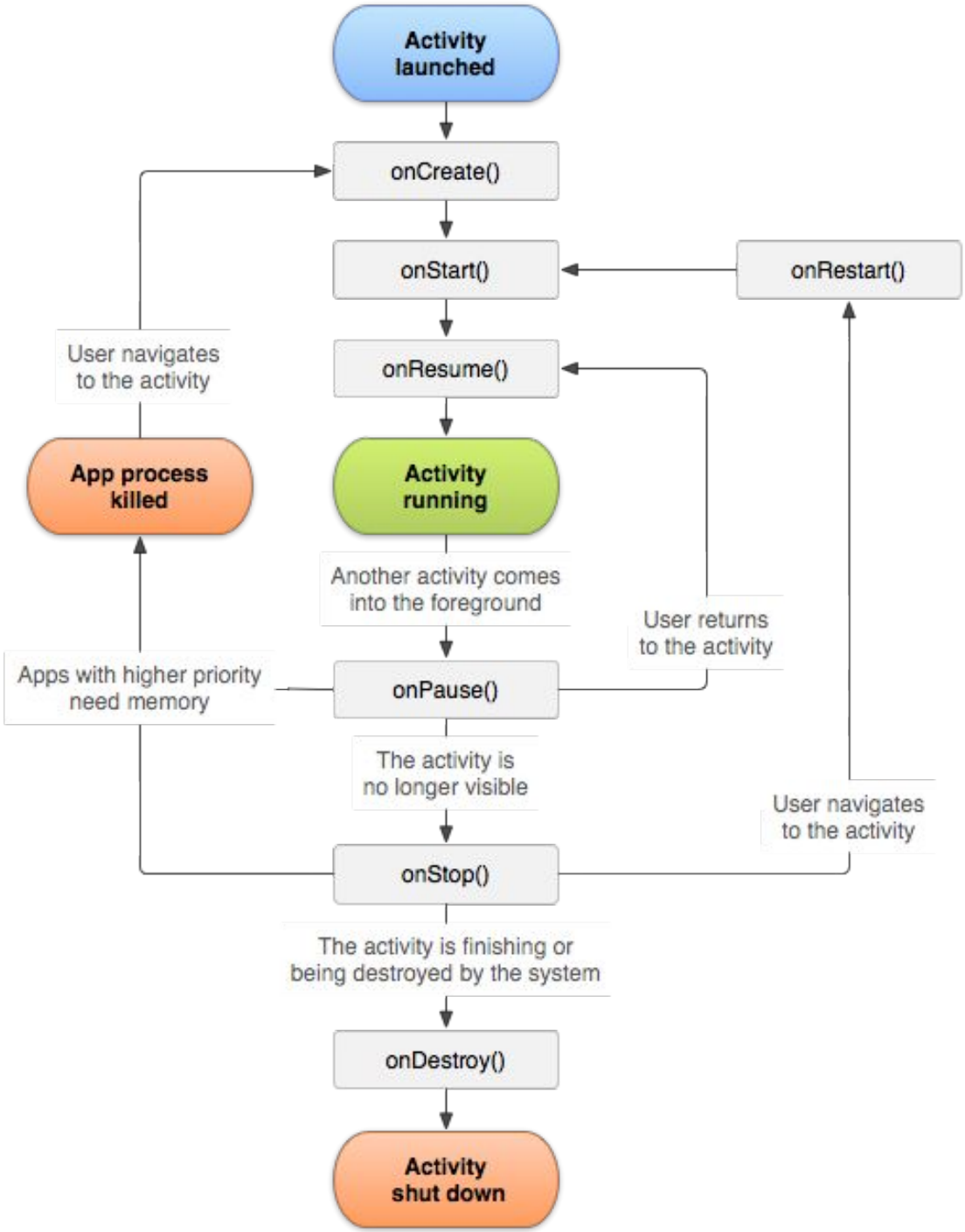
Три вложенных цикла в жизненном цикле операции

В промежутке между вызовами этих двух методов можно сохранить ресурсы, необходимые для отображения операции для пользователя. Например, можно зарегистрировать объект BroadcastReceiver в методе onStart() для отслеживания изменений, влияющих на пользовательский интерфейс, а затем отменить его регистрацию в методе onStop(), когда пользователь больше не видит отображаемого.

В течение всего жизненного цикла операции система может несколько раз вызывать методы onStart() и onStop(), поскольку операция то отображается для пользователя, то скрывается от него.

Жизненный цикл операции, выполняемый на переднем плане, происходит между вызовами методов onResume() и onPause(). В течение этого времени операция выполняется на фоне всех прочих операций и отображается для пользователя. Операция может часто уходить в фоновый режим и выходить из него — например, метод onPause() вызывается при переходе устройства в спящий режим или при появлении диалогового окна.

Поскольку переход в это состояние может выполняться довольно часто, код в этих двух методах должен быть легким.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Сохранение состояния операции

В случае приостановки или полной остановки операции ее состояние сохраняется.

Это действительно так, поскольку объект Activity при этом по-прежнему находится в памяти , и вся информация о ее элементах и текущем состоянии по-прежнему активна. Поэтому любые вносимые пользователем в операции изменения сохраняются, и когда операция возвращается на передний план (когда она «возобновляется»), эти изменения остаются в этом объекте.

Сохранение состояния операции

Однако когда система уничтожает операцию в целях восстановления памяти, объект Activity уничтожается, в результате чего системе не удается просто восстановить состояние операции для взаимодействия с ней.

Вместо этого системе необходимо повторно создать объект Activity, если пользователь возвращается к нему. Но пользователю неизвестно, что система уже уничтожила операцию и создала ее повторно, поэтому, возможно, он ожидает, что операция осталась прежней.

В этой ситуации можно обеспечить сохранение важной информации о состоянии операции путем реализации дополнительного метода обратного вызова, который позволяет сохранить информацию о вашей операции: onSaveInstanceState().

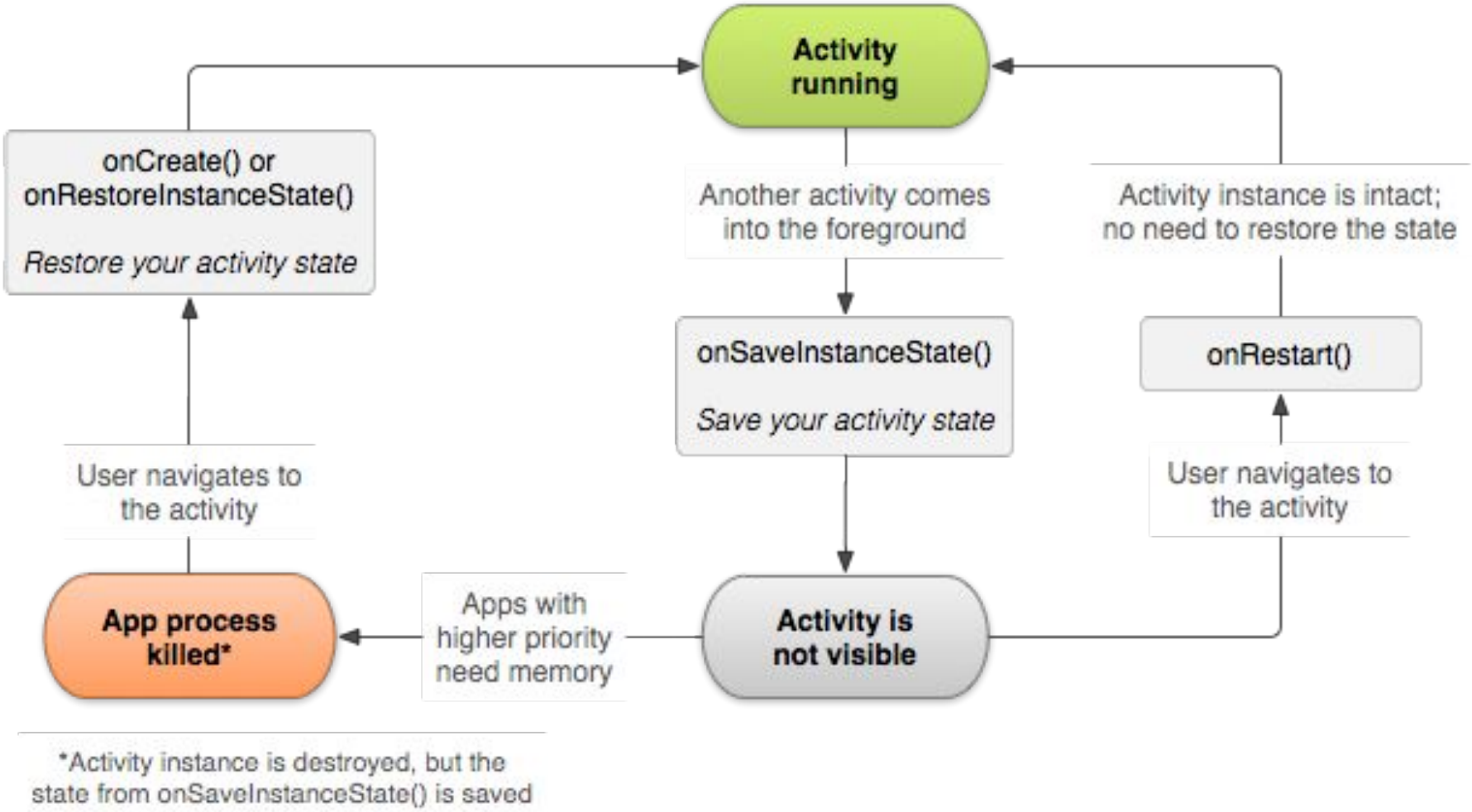
Роль объекта Bundle

Прежде чем сделать операцию доступной для уничтожения, система вызывает метод onSaveInstanceState().

Система передает в этот метод объект Bundle, в котором можно сохранить информацию о состоянии операции в виде пар «имя-значение», используя для этого такие методы, как putString() и putInt().

Затем, если система завершает процесс вашего приложения и пользователь возвращается к вашей операции, система повторно создает операцию и передает объект Bundle в оба метода: onCreate() и onRestoreInstanceState().

С помощью любого из этих методов можно извлечь из объекта Bundle сохраненную информацию о состоянии операции и восстановить ее. Если такая информация отсутствует, то объект Bundle передается с нулевым значением (это происходит в случае, когда операция создается в первый раз).



Два способа возврата операции к отображению для пользователя в неизмененном состоянии: уничтожение операции с последующим ее повторным созданием,

- когда операция должна восстановить свое ранее сохраненное состояние,
- или остановка операции и ее последующее восстановление в неизмененном состоянии.

Реализация по умолчанию метода onSaveInstanceState()

Однако, даже если вы не реализуете метод onSaveInstanceState(), часть состояния операции восстанавливается реализацией по умолчанию метода onSaveInstanceState() класса Activity.

В частности, реализация по умолчанию вызывает соответствующий метод onSaveInstanceState() для каждого объекта View в макете, благодаря чему каждое представление может предоставлять ту информацию о себе, которую следует сохранить.

Почти каждый виджет в платформе Android реализует этот метод необходимым для себя способом так, что любые видимые изменения в пользовательском интерфейсе автоматически сохраняются и восстанавливаются при повторном создании операции.

- Например, виджет EditText сохраняет любой текст, введенный пользователем, а виджет CheckBox сохраняет информацию о том, был ли установлен флажок.

От вас требуется лишь указать уникальный идентификатор (с атрибутом android:id) для каждого виджета, состояние которого необходимо сохранить.

- Если виджету не присвоен идентификатор, то системе не удастся сохранить его состояние.

О сохранении по умолчанию

Вы также можете явно отключить сохранение информации о состоянии представления в макете.

Для этого задайте для атрибута android:saveEnabled значение "false" или вызовите метод setSaveEnabled().

Обычно отключать сохранение такой информации не требуется , однако это может потребоваться в случаях, когда восстановить состояние пользовательского интерфейса операции необходимо другим образом.

Несмотря на то что реализация метода onSaveInstanceState() по умолчанию позволяет сохранить полезную информацию о пользовательском интерфейсе вашей операции, вам по-прежнему может потребоваться **переопределить** ее для сохранения дополнительной информации.

О сохранении по умолчанию

- Например, может потребоваться сохранить значения элементов, которые изменялись в течение жизненного цикла операции (которые могут **коррелировать со значениями, восстановленными в пользовательском интерфейсе,** однако элементы, содержащие эти значения пользовательского интерфейса, по умолчанию не были восстановлены).

Поскольку реализация метода onSaveInstanceState() по умолчанию позволяет сохранить состояние пользовательского интерфейса, в случае , если вы переопределите метод с целью сохранить дополнительную информацию о состоянии, перед выполнением каких-либо действий вы всегда можете вызвать реализацию суперкласса для метода onSaveInstanceState().

Точно так же реализацию суперкласса onRestoreInstanceState() следует вызывать в случае ее переопределения, чтобы реализация по умолчанию могла сохранить состояния представлений.

Возможность приложения восстанавливать свое состояние

При изменении ориентации экрана система уничтожает и повторно создает операцию, чтобы применить альтернативные ресурсы, которые могут быть доступны для новой конфигурации экрана.

Обработка изменений в конфигурации

Некоторые конфигурации устройств могут изменяться в режиме выполнения (например, ориентация экрана, доступность клавиатуры и язык).

В таких случаях Android повторно создает выполняющуюся операцию (система сначала вызывает метод onDestroy(), а затем сразу же вызывает метод onCreate()). Такое поведение позволяет приложению учитывать новые конфигурации путем автоматической перезагрузки в приложение альтернативных ресурсов, которые вы предоставили (например, различные макеты для разных ориентаций и экранов разных размеров).

Если операция разработана должным образом и должным образом поддерживает перезапуск после изменения ориентации экрана и восстановление своего состояния, как описано выше, ваше приложение можно считать более устойчивым к другим непредвиденным событиям в жизненном цикле операции.

Лучший способ обработки такого перезапуска — сохранить и восстановить состояние операции с помощью методов onSaveInstanceState() и onRestoreInstanceState() (или onCreate()), как описано в предыдущем разделе.

Согласование операций

Когда одна операция запускает другую, в жизненных циклах обеих из них происходит переход из одного состояния в другое.

Первая операция приостанавливается и заврешается (однако она не будет остановлена, если она по-прежнему видима на фоне), а вторая операция создается.

В случае, если эти операции обмениваются данным, сохраненными на диске или в другом месте, важно понимать, что первая операция не останавливается полностью до тех пор, пока не будет создана вторая операция.

Порядок обратных вызовов жизненного цикла четко определен, в частности, когда в одном и том же процессе находятся две операции , и одна из них запускает другую.

Ниже представлен порядок выполнения действий в случае, когда **операция А запускает операцию Б.**

1. Выполняется метод onPause() операции А.
2. Последовательно выполняются методы onCreate(), onStart() и onResume() операции Б. (Теперь для пользователя отображается операция Б.)
3. Затем, если операция А больше не отображается на экране, выполняется ее метод onStop().

Такая предсказуемая последовательность выполнения обратных вызовов жизненного цикла позволяет управлять переходом информации из одной операции в другую.

Например, если после остановки первой операции требуется выполнить запись в базу данных, чтобы следующая операция могла считать их, то запись в базу данных следует выполнить во время выполнения метода onPause(), а не во время выполнения метода onStop().

Литература

- https://developer.android.com/guide/components/activities.html#Lifecycle
- http://developer.alexanderklimov.ru/android/theory/whatsnew.php