2.2. Жизненный цикл активности

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 2.2. Жизненный цикл активности

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:42

Оглавление

2.2.1 Активность (Activity)

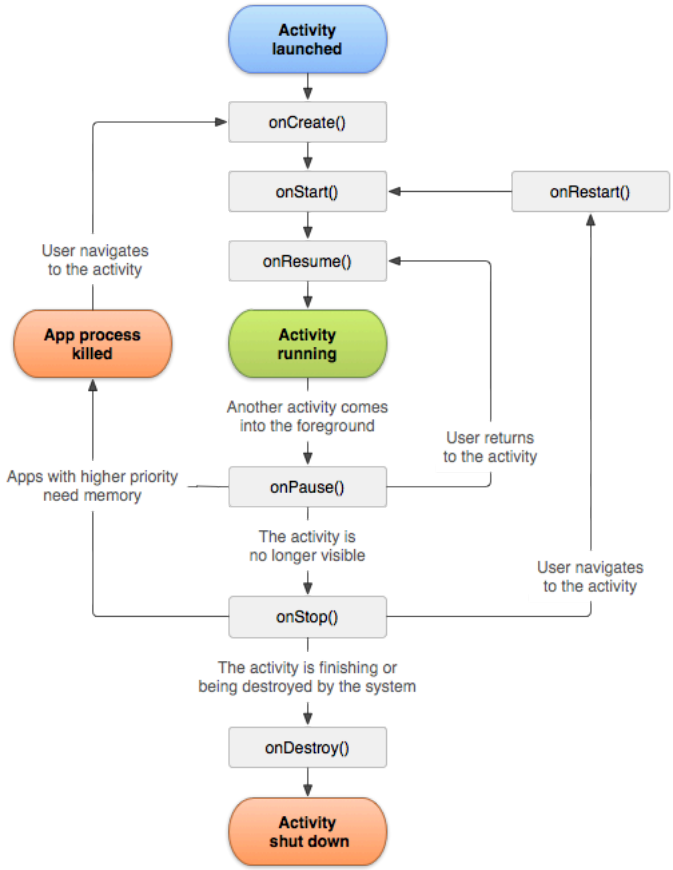
Android-приложение - это набор объектов типа *Активность (Activity)* или его наследников, в том числе наследников класса *FragmentActivity*. Каждая активность представляет окно на экране устройства (обычно занимает весь экран), в котором происходит деятельность приложения. На активности размещаются *представления (View)*- базовый класс для виджетов (Widgets). Для каждого View на экране выделяется прямоугольная область определённого размера, указанного при описании представления, в которой располагается его содержимое. Активность реализует визуальный интерфейс пользователя. Если в проекте несколько окон, то и активностей будет несколько, причем активности будут независимы друг от друга. Это значит, что при работе одной активности (состояние Active) остальные будут находиться в одном из состояний: Paused, Stoped или Inactive, в зависимости от их приоритета в системе. Высший приоритет - процесс переднего плана - у работающей (Active) активности, далее по убыванию идут приостановленная (Paused), остановленная (Stoped) и неактивная (Inactive). Приоритет можно указать в качестве свойства активности в файле манифеста. Операционная система сама принимает решение об уничтожении активности, если возникает необходимость в освобождении оперативной памяти. Сначала разрушаются активности с самым низким приоритетом. Работающая активность системой уничтожена никогда не будет. Размеры и положение активности не обязательно должны совпадать с размером экрана, можно устанавливать пользовательские параметры, в том числе программировать всплывающие окна. Поскольку на первом же шаге работы мастера создания проекта выбирается шаблон экрана, то после сборки в приложении имеется одна активность. Макет построенной активности описан в файле разметки res/layout/activity_main.xml, программная реализация - в файле кода MainActivity.kt.

```
package ru.samsung.itacademy.mdev.firstproject

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Активность MainActivity наследуется от класса AppCompatActivity, который входит в пакет androidx в следующей иерархии: androidx.activity.ComponentActivity -> androidx.fragment.app.FragmentActivity -> androidx.appcompat.app.AppCompatActivity. Базовый класс Activity содержит порядка 200 методов, в том числе методы обратного вызова, которые вызываются операционной системой в зависимости от состояния активности. Наследуемые методы обратного вызова обязательно используют родительский метод, по этому первая строка метода представляет собой вызов метода суперкласса *super.onCreate(savedInstanceState)* Являясь наследником Activity, данный класс наследует все методы родительского класса. При создании проекта Android Studio переопределяет основной метод *onCreate()*, без которого активность не будет создана. Этот метод вызывается, как только происходит запуск активности и она переходит в состояние Active. Смена состояний активности под воздействием методов обратного вызова образует *жизненный цикл активности*, схема которого изображена на рисунке 2.2.1.



Каждая активность регистрируется в файле манифеста в виде объекта <activity> внутри родительского <application>.

2.2.2 Отслеживание состояний объектов

Поскольку система не сообщает, в какой момент выполняется какой метод, то для отслеживания деятельности приложения используют экземпляры классов, умеющие выводить текстовые сообщения. Эти же классы используются для отладки приложений.

Класс Toast

Класс всплывающих сообщений. Имеет конструктор, получающий окружение, сообщение и длительность показа этого сообщения в выбранном контексте. Такие сообщения могут показываться только в окружении активности, управляющие элементы не имеют возможности показывать всплывающие окна.

```
Toast.makeText(Context context,String message,Int duration)
```

Если свойства подсказки не будут настраиваться, то можно создавать анонимный экземпляр и сразу же его выводить на экран:

```
Toast.makeText(this,”message”,Toast.LENGTH_SHORT).show()
```

По умолчанию объект центрируется по нижнему краю активности, но можно его переместить с помощью сеттера setGravity(). В таком случае создаётся переменная класса Toast, настраиваются её свойства, и затем производится показ сообщения на экране:

```
var t = Toast.makeText(this,”message”,Toast.LENGTH_SHORT)
t.setGravity(Gravity.CENTER_VERTICAL,0,0)
t.show()
```

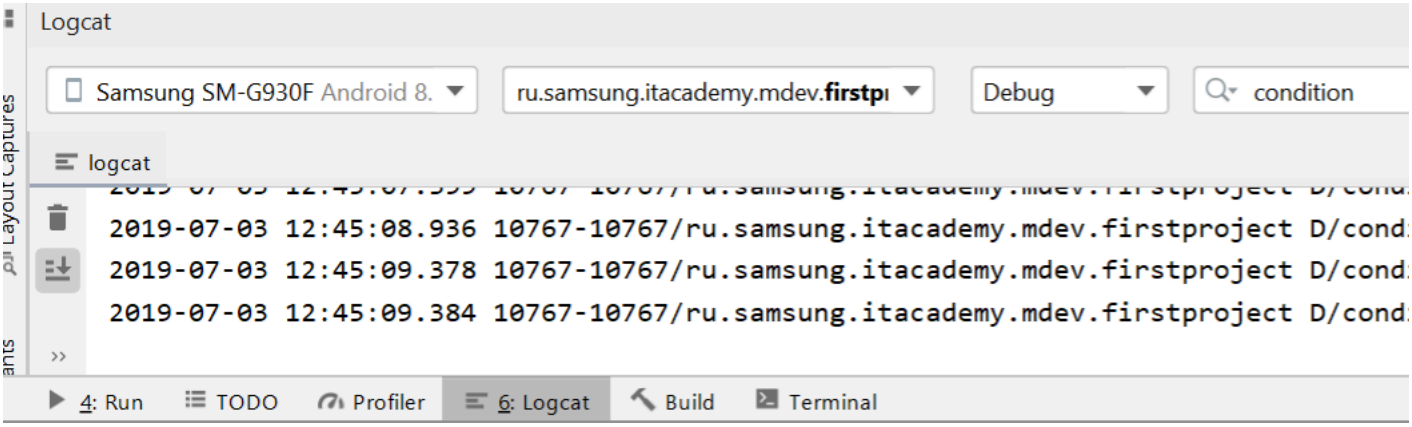
В тексте сообщения помещается выводимая информация.

Класс Log

Альтернативой всплывающим сообщениям являются консольные сообщения, не требующие экрана устройства для своего показа. Эти объекты используют окно сообщений LogCat для вывода. Класс Log является статическим, то есть сам выступает в качестве объекта, порождающего сообщение. В классе имеются методы для чтения системных сообщений различного уровня важности:

- Verbose - полный протокол системных действий.
- Debug - отладочная информация: параметры,порождаемые в процессе работы приложения.
- Info - информационные сообщения.
- Warn - предупреждения.
- Error - ошибки, возникающие в процессе работы приложения. В протоколе выделяются красно-коричневым цветом, чтобы их было сразу заметно.
- Assert - предложения, введённые в программу программистом для отслеживания состояний выполнения.

Понятно, что уровень Verbose выводит все сообщения, относящиеся и к остальным уровням. Таким образом, протокол получается очень подробный, но слишком большой, что приводит к сложности поиска нужной строки. По этому рекомендуется для отслеживания использовать уровень Debug. Сообщения можно генерировать из кода, для этого в классе Log предусмотрены методы по уровням важности от `Log.v(tag: String, message: String)` для уровня Verbose до `Log.a(tag: String, message: String)` для уровня Assert. Первый аргумент функции - это текстовая строка «имени сообщения» для его идентификации в протоколе, второй аргумент - строка сообщения. Имя сообщения позволяет осуществлять поиск в протоколе сообщений общего назначения. На рисунке 2.2.2 показан поиск сообщений с тегом «condition»



Упражнение 2.2.1 Отслеживание методов обратного вызова.

1. Откройте приложение My_app и в файле манифеста удалите принудительную ориентацию экрана активности: `android:screenOrientation="landscape"`. В классе MainActivity приложения создайте текстовую константу для именования сообщений логирования

```
private val TAG = "condition"
```

и напишите функцию, выводящую сообщения на экран в виде Toast и в журнал событий LogCat

```
fun showMessage(s: String){
    Toast.makeText(this,s,Toast.LENGTH_SHORT).show()
    Log.d(TAG, s)
}
```

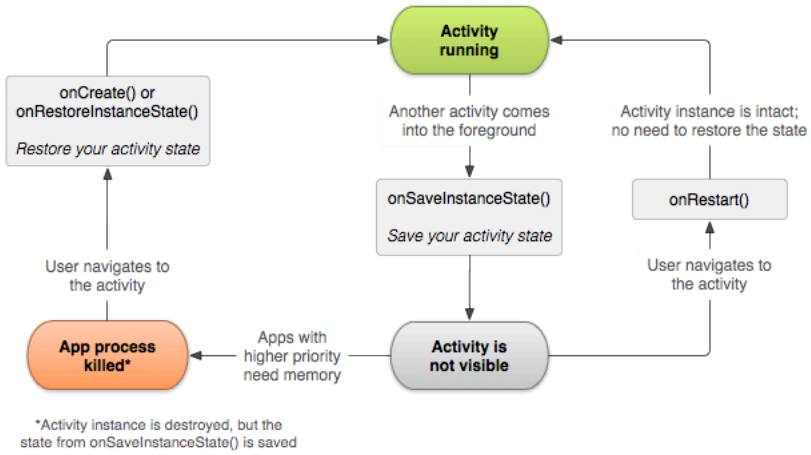
2. В методе onCreate() закомментируйте строку *setContentView(R.layout.activity_ main),* чтобы активность была пустой и вызовите функцию *showMessage()* с аргументом *“I created”,* сообщающем о создании активности.

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
       // setContentView(R.layout.activity_main)
        showMessage("I created")
}
```

3. Запустите приложение на устройстве и убедитесь в появлении всплывающего сообщения на экране.
4. Откройте вкладку LogCat окна консоли и введите в поле поиска тег имени сообщений *condition*. В окне должна остаться единственная строка с сообщением “I created”.
5. Переопределите аналогично методы onPause(), onStart(), onRestart() и onDestroy().
6. Определите последовательность работы методов при запуске приложения, при повороте экрана, при сворачивании приложения и при его закрытии. Сделайте вывод о применимости методов onStart() и onRestart().

2.2.3 Сохранение состояния активности. Класс Bundle.

Методы обратного вызова, останавливающие работу активности, но не разрушающие её, сохраняют параметры состояния закрываемого окна. По этому при возобновлении работы активность возвращается в то же состояние, в котором была остановлена. Но при срабатывании метода onDestroy(), когда происходит удаление объекта из памяти устройства, состояние не сохраняется и метод onCreate() запускает активность в её начальном состоянии. Это очень неудобно, поскольку пользователь не может указывать системе, какую активность нельзя уничтожать, а при повороте экрана система обязательно вызовет метод onDestroy() и перестроит активность заново методом onCreate(). Для сохранения состояния активности даже при её разрушении в коллекции инструментов у разработчика имеется класс Bundle, позволяющий сохранить текущее состояние активности. Запись параметров активности в Bundle производится в методе обратного вызова `onSaveInstanceState()`, получающего от системы в качестве параметра текущее состояние activity. Параметры записываются ассоциативным массивом в виде пар «имя свойства - значение свойства». Таким образом, до вызова метода onStop() система запустит onSaveInstanceState() и запишет через него состояние активности. Тогда при новом запуске этой активности в метод onCreate() будет передано последнее сохранённое состояние. Вместо onCreate() может быть выполнен метод обратного вызова `onRestoreInstanceState()`, тоже восстанавливающий последнее сохранённое состояние запускаемой активности. Следует заметить, что метод onSaveInstantState() не будет запускаться системой при закрытии активности пользователем, а так же он присутствует неявно у элементов оконного интерфейса приложения. Если у представления установлен атрибут `id`, то его состояние сохраняется системой.



Начать тур для пользователя на этой странице