Life-Cycle

Một ứng dụng gồm một hoặc nhiều thành phần được định nghĩa trong file manifest. Một thành phần có là một trong các phần sau:

1. Một Activity

Một Activity hiện hữu cho một giao diện của ứng dụng. Mỗi Activity có một giao diện riêng. Activity được đánh dấu là Activity đầu tiên sẽ được chạy khi chương trình chạy. Việc chuyển từ activity này đến activity khác bằng việc gọi **intents**.

2. Một Service

Một service không có giao diện, nhưng chạy ở bên dưới chúng ta không biết được thời gian của nó. Nó có thể kết nối tới một service đang chạy (và khởi động service nếu service đó không chạy). Trong quá trình kết nối, bạn có thể giao tiếp với service qua một giao diện đó là serviec exposes.

3. Một broadcast receiver

Broadcast receiver là một thành phần mà không có gì nhưng nhận và phản ứng lại những thông báo được broadcast. Nhiều broadcast gốc trong mã hệ thống (ví dụ: "you got mail") và các ứng dụng cũng có thể khởi tạo broadcast. Broadcast receivers không hiển thị giao diện người dùng. Tuy nhiên, họ có thể khởi tạo một Activity phản hồi lại những thông tin được nhận. Họ có thể dùng notification manager để cảnh giác đến người dùng.

4. Một content provider

Một content provider cung cấp một tập chi tiết dữ liệu ứng dụng đến các ứng dụng khác. Dữ liệu thường được lưu trữ ở file hệ thống, hoặc trong một SQLite database. Content Provider hiện thực một tập phương thức chuẩn mà các ứng dụng khác có thể truy xuất và lưu trữ dữ liệu của loại nó điều khiển.

Tuy nhiên, những ứng dụng không thể gọi các phương thức trực tiếp. Hơn thế chúng dùng content resolver và gọi những phương thức đó. Một content resolver có thể giao tiếp đến nhiều content provider; nó cộng tác với các provider để quản lý bất kỳ giao tiếp bên trong liên quan.

Mỗi ứng dụng Android chạy trên process của nó. Bất cứ lúc nào nó cũng quản lý được các thành phần của nó:

- Android phải đảm bảo các process thành phẩn phải đang chạy
- Khởi tạo nếu cần thiết

- Một trường hợp tương thích của cái thành phần là có giá trị, đang khởi tạo trường hợp nếu cần thiết.

Chu kỳ ứng dụng

Một tiến trình Linux gói gọn một ứng dụng Android đã được tạo ra cho ứng dụng khi codes cần được run và sẽ còn chạy cho đến khi

1. nó không phụ thuộc
2. hệ thống cần lấy lại bộ nhớ mà nó chiếm giữ cho các ứng dụng khác

Một sự khác thường và đặc tính cơ bản của Android là thời gian sống của tiến trình ứng dụng không được điều khiển trực tiếp bởi chính nó. Thay vào đó, nó được xác định bởi hệ thống qua một kết hợp của

1. những phần của ứng dụng mà hệ thống biết đáng chạy
2. những phần quan trong như thế nào đối với người dùng
3. bao nhiêu vùng nhớ chiếm lĩnh trong hệ thống

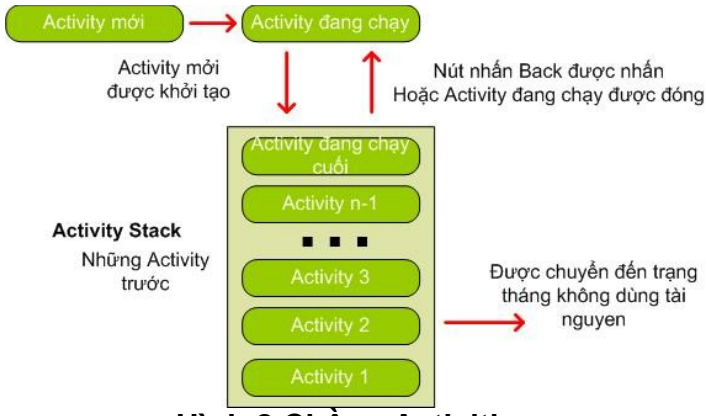
Chu kỳ sống thành phần

Các thành phần ứng dụng có một chu kỳ sống, tức là mỗi thành phần từ lúc bắt đầu khởi tạo và đến thời điểm kết thúc. Giữa đó, đôi lúc chúng có thể là active hoặc inactive, hoặc là trong trường hợp activies nó có thể visible hoặc invisible

Hình 1 Chu kỳ sống của ứng dụng Android

Activity Stack

Bên trong hệ thống các activity được quản lý như một activity stack. Khi một Activity mới được start, nó được đặt ở đỉnh của stack và trở thành activity đang chạy -- activity trước sẽ ở bên dưới activity mới và sẽ không thấy trong suốt quá trình activity mới tồn tại. Nếu người dùng nhấn nút Back thì activity kết tiếp của stack sẽ di duyển lên và trở thành active.



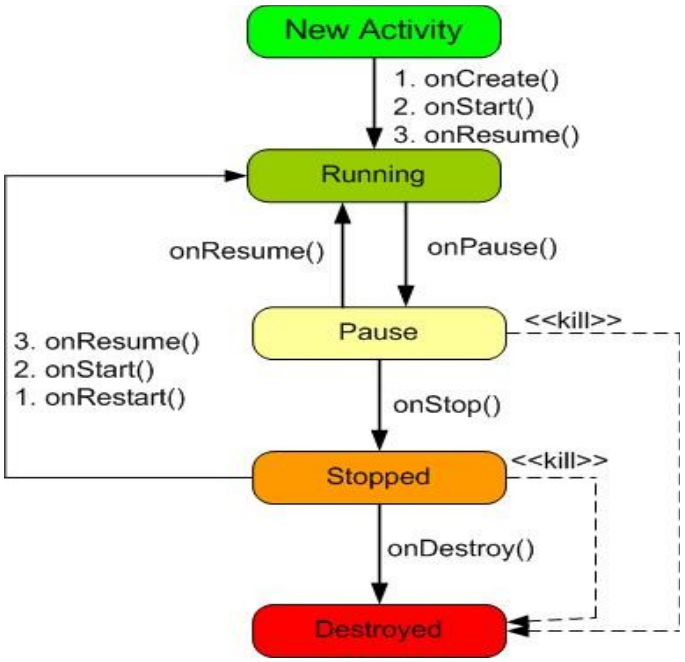
Hình 2 Chồng Activities

Các trạng thái của chu kỳ

Một Activity chủ yế có 3 chu kỳ chính sau:

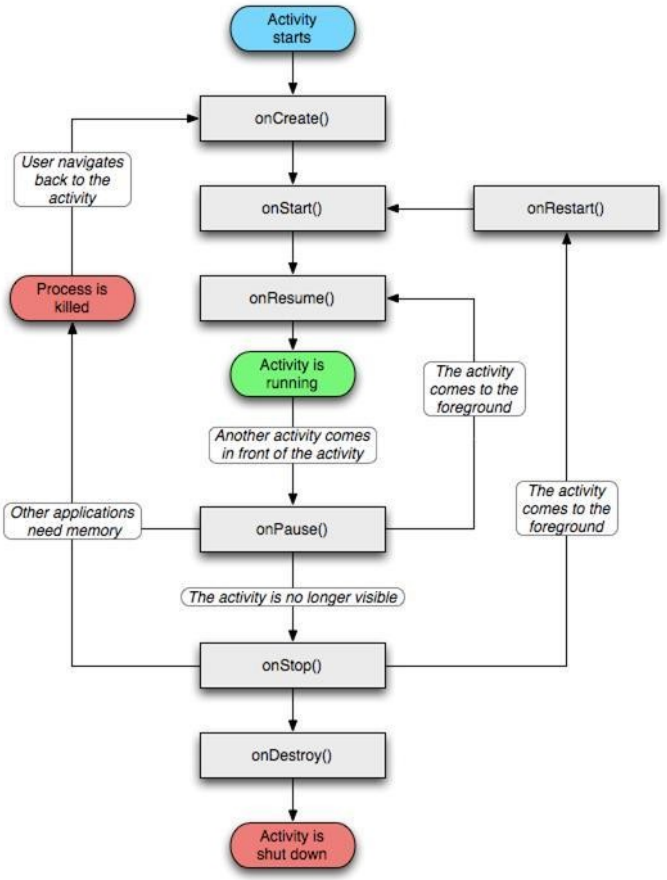
1. Active hoặc running: Khi Activity là được chạy trên màn hình. Activity này tập trung vào những thao tác của người dùng trên ứng dụng.
2. Paused: Activity là được tạm dừng (paused) khi mất focus nhưng người dùng vẫn trông thấy. Có nghĩa là một Activity mới ở trên nó nhưng không bao phủ đầy màn hình. Một Activity tạm dừng là còn sống nhưng có thể bị kết thúc bởi hệ thống trong trường hợp thiếu vùng nhớ.

3. Stopped: Nếu nó hoàn toàn bao phủ bởi Activity khác. Nó vẫn còn trạng thái và thông tin thành viên tron nó. Người dùng không thấy nó và thường bị loại bỏ trong trường hợp hệ thống cần vùng nhớ cho tác vụ khác.



Hình 3 Các trạng thái của chu kỳ một Activity

Chu kỳ của ứng dụng



Hình 4 Chu kỳ ứng dụng Android

Các sự kiện trong chu kỳ sống

Nếu một Activity được tạm dừng hoặc dừng hẳn, hệ thống có thể bỏ thông tin khác của nó từ vùng nhớ bởi việc finish() (gọi hàm finish() của nó), hoặc đơn giản giết tiến trình của nó.

Khi nó được hiển thị lần nữa với người dùng, nó phải được hoàn toàn restart và phục hồi lại trạng thái trước.

Khi một Activity chuyển qua chuyển lại giữa các trạng thái, nó phải báo việc chuyển của nó bằng việc gọi hàm transition.

Hình 5 Các phương thức trong 1 chu kỳ của Activity

Tất cả các phương thức là những móc nối mà bạn có thể override để làm tương thich công việc trong ứng dụng khi thay đổi trạng thái. Tất cả các Activity bắt buộc phải có

onCreate() để khởi tạo ứng dụng. Nhiều Activity sẽ cũng hiện thực onPause() để xác nhận việc thay đổi dữ liệu và mặt khác chuẩn bị dừng hoạt động với người dùng.

Thời gian sống của ứng dụng

7 phương thức chuyển tiệp định nghĩa trong chu kỳ sông của một Activity. Thời gian sống của một Activity diễn ra giữa lần đầu tiên gọi onCreate() đến trạng thái cuối cùng gọi onDestroy(). Một Activity khởi tạo toàn bộ trạng thái toàn cục trong onCreate(), và giải phóng các tài nguyên đang tồn tại tron onDestroy().

Visible Lifetime

Visible lifetime của một activity diễn ra giữa lầ gọi một onStart() cho đến khi gọi onStop(). Trong suốt khoảng thời gian này người dùng có thể thấy activity trên màn hình, có nghĩa là nó không bị foreground hoặc đang tương tác với người dùng. Giữa 2 phương thức người dùng có thể duy trì tài nguyên để hiển thị activity đến người dùng.

Foreground LifeTime

Foregroudn lifetime của một activity diễn ra giữa 2 lần gọi onResume() và onPause()
Trong khoảng thời gian này, activity ở trước tất cả activity khác và là đang tương tác với người dùng.
Một Actifity thường xuyên chuyển trạng thái giữa resumed và paused.

- onPause() được gọi khi thiết bi đi vào trạng thái sleep hoặc khi một activity mới là được start.
- onResume() được gọi khi một Activity hoặc một intent mới được delivered

Các phương thức của chu kỳ sống

Phương thức: onCreate()

- Được gọi khi activity lần đầu tiên được tạo
- Ở đây bạn làm tất cả các cài đặt tĩnh -- tạo các view, kết nối dữ liệu đến list và .v.v
- Phương thức này gửi qua một đối tượng*Bundle* chứa đựng từ trạng thái trước của Activity
- Luôn theo sau bởi onStart()

Phương thức: onRestart()

- Được gọi sau khi activity đã được dừng, chỉ một khoảng đang khởi động lần nữa (stared again)
- Luôn theo sau bởi onStart()

Phương thức: onStart()

- Được gọi trước khi một activity visible với người dùng
- Theo sau bởi *onResume()* nếu activity đến trạng thái foreground hoặc *onStop()* nế nó trở nên ẩn.

Phương thức: onResume()

- Được gọi trước khi activity bắt đầu tương tác với người dùng
- Tại thời điểm này activity ở trên dỉnh của stack activity.
- Luôn theo sau bởi *onPause()*

Phương thức: onPause()

- Được gọi khi hệ thống đang resuming activity khác
- Phương thức này là điển hình việc giữ lại không đổi dữ liệu.
- Nó nên được diễn ra một cách nhanh chóng bởi vì activity kế tiếp sẽ không được resumed ngay cho đến khi nó trở lại.
- Theo sau bởi *onResume* nếu activity trở về từ ở trước, hoặc bởi *onStop* nếu nó trở nên visible với người dùng.
- Trạng thái của activity có thể bị giết bởi hệ thống

Phương thức: onStop()

- Được gọi khi activity không thuộc tầm nhìn của người dùng.
- Nó có thể diễn ra bởi vì nó đang bị hủy, hoặc bởi vì activity khác vữa được resumed và bao phủ nó.
- Được theo sau bởi *onRestart()* nếu activity đang đở lại để tương tác với người dùng, hoặc *onDestroy()* nếu activity đang bỏ.
- Trạng thái của activity có thể bị giết bởi hệ thống

Phương thức: onDestroy()

- Được gọi trước khi activity bị hủy.
- Đó là lần gọi cuối cùng mà activity này được nhận.
- Nó được gọi khác bởi vì activity đang hoàn thành, hoặc bởi vì hệ thống tạm thời bị hủy diệt để tiết kiệm vùng nhớ.
- Bạn có thể phân bietj giữa 2 kịch bản với phương *isFinshing()*
- Trạng thái của activity có thể được giết bởi hệ thống.

Killable States

Activity đang ở trạng thái này có thể bị kết thúc bởi hệ thống bất kỳ lúc nào sau khi phương thức trả về.

3 phương thức (*onPause(), onStop(), và onDestroy()* là killable

onPause() phải được gọi trước khi tiến trình bị giết -- *onStop* và *onDestroy* có thể là không.

Vì vậy chúng ta dung *onPause* viết bất kỳ dữ liệu chắc để lưu trữ.

Ví dụ: Demo Life Cycle

Mục đích của ứng dụng này sẽ gọi các hàm trong Activity Life Cycle

```
//protected void onCreate(Bundle savedInstanceSta te);
//protected void onStart();
//protected void onRestart();
//protected void onResume();
//protected void onPause();
//protected void onStop();
//protected void onDestroy();
```

Tạo một ứng dụng mới trên Android với tên tùy thích với file Activity VietAndLifeCycle.java java:

```
1.
2. package com.vietanddev.vonguyen;

3.

4. import android.app.Activity;

5. import android.content.SharedPreferences;
```

```
6. import android.os.Bundle;
7. import android.view.View;
8. import android.widget.Button;
9. import android.widget.EditText;
10. import android.widget.TextView;
11. import android.widget.Toast;
12.
13. // Muc dich cua vi du nay la show cac su kien cua chu ky song
14.
15. //protected void onCreate(Bundle savedInstanceSta  te);
16. //protected void onStart();
17. //protected void onRestart();
18. //protected void onResume();
19. //protected void onPause();
20. //protected void onStop();
21. //protected void onDestroy();
22.
23. public class VietAndLifeCycle extends Activity {
24.
25.     public static final String TAGID = "VietAnddev:LifeCycle";
26.     public static final int acMode = Activity.MODE_PRIVATE;
27.
28.     EditText txtMsg;
29.     Button btnFinish;
30.     TextView txtTodo;
31.
32.     /** Called when the activity is first created. */
33.     @Override
34.     public void onCreate(Bundle savedInstanceState) {
```

```
35.         super.onCreate(savedInstanceState);
36.         setContentView(R.layout.main);
37.
38.         txtMsg = (EditText) findViewById(R.id.txtMsg);
39.         updateFromSavedState_IfNeeded();
40.         txtTodo = (TextView) findViewById(R.id.txtTodo);
41.         String msg = "Instructions:                 \n "
42.                 + "0. New instance (onCreate, onStart, onResume
    )   \n "
43.                 + "1. Back Arrow   (onPause, onStop, onDestro  y)
    \n "
44.                 + "2. Finish       (onPause, onStop, onDes  troy)
    \n "
45.                 + "3. Home (onPause, onStop) \n "
46.                 + "4. After 3> App Tab> re-execute current app \n
    "
47.                 + "  (onRestart, onStart, onResume) \n "
48.                 + "5. Run DDMS> Receive a phone call or   SMS \n
    "
49.                 + "  (onRestart, onStart, onResume) \n "
50.                 + "6. Enter some data - repeat steps 1-5    \n ";

51.         txtTodo.setText(msg);
52.
53.         btnFinish = (Button) findViewById(R.id.btnFinish);
54.         btnFinish.setOnClickListener(new Button.OnClickListener()
    {
55.
56.             @Override
57.             public void onClick(View v) {
58.                 // TODO Auto-generated method stub
59.                 finish();
```

```
60.             }
61.
62.         });
63.
64.         // xem message nay khi onCreate()duoc goi
65.         Toast.makeText(getBaseContext(), "onCreate ...", Toast.LE
   NGTH_LONG)
66.                 .show();
67.
68.     }
69.
70.     @Override
71.     protected void onDestroy() {
72.         super.onDestroy();
73.         clearMyPrefs();
74.         Toast.makeText(getBaseContext(), "onDestroy ...", Toast.L
   ENGTH_LONG)
75.                 .show();
76.
77.     }
78.
79.     @Override
80.     protected void onPause() {
81.         super.onPause();
82.         saveDataFromCurrentState();
83.         Toast.makeText(getBaseContext(), "onPause ...", Toast.LEN
   GTH_LONG)
84.                 .show();
85.     }
86.
```

```
87.     @Override
88.     protected void onRestart() {
89.         super.onRestart();
90.         Toast.makeText(getBaseContext(), "onRestart ...", Toast.L
   ENGTH_LONG)
91.                 .show();
92.     }
93.
94.     @Override
95.     protected void onResume() {
96.         super.onResume();
97.         Toast.makeText(getBaseContext(), "onResume...", Toast.LEN
   GTH_LONG)
98.                 .show();
99.     }
100.
101.         @Override
102.         protected void onStart() {
103.             super.onStart();
104.             Toast.makeText(getBaseContext(), "onStart ...", Toa
   st.LENGTH_LONG)
105.                     .show();
106.         }
107.
108.         @Override
109.         protected void onRestoreInstanceState(Bundle savedInsta
   nceState) {
110.             super.onRestoreInstanceState(savedInstanceState);
111.             Toast.makeText(getBaseContext(), "onRestoreInstance
   State ...BUNDLING",
112.                     Toast.LENGTH_LONG).show();
```

```
113.         }
114.
115.         @Override
116.         protected void onStop() {
117.             super.onStop();
118.             Toast.makeText(getBaseContext(), "onStop ...", Toas
   t.LENGTH_LONG)
119.                     .show();
120.         }
121.
122.         private void saveDataFromCurrentState() {
123.             // TODO Auto-generated method stub
124.             SharedPreferences myPrefs = getSharedPreferences(TA
   GID, acMode);
125.             SharedPreferences.Editor myEditor = myPrefs.edit();
126.             myEditor.putString("txtMsg", txtMsg.getText().toStr
   ing());
127.             myEditor.commit();
128.         }
129.
130.
131.         private void clearMyPrefs() {
132.             // TODO Auto-generated method stub
133.             SharedPreferences myPrefs = getSharedPreferences(TA
   GID, acMode);
134.             SharedPreferences.Editor myEditor = myPrefs.edit();
135.             myEditor.clear();
136.             myEditor.commit();
137.         }
138.
139.         private void updateFromSavedState_IfNeeded() {
```

```
            // TODO Auto-generated method stub
            SharedPreferences myPrefs = getSharedPreferences(TA
   GID, acMode);
            if ((myPrefs != null) && (myPrefs.contains("txtMsg"
   ))) {
                String myData = myPrefs.getString("txtMsg", "");

                txtMsg.setText(myData);
            }
        }

        @Override
        protected void onSaveInstanceState(Bundle outState){
            super.onSaveInstanceState(outState);
        }
    }
```

Layout code:

java:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/widget0"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ffffcccc"
android<img src="images/smilies/redface.gif" alt="" title="Embarrassmen
   t" class="inlineimg" border="0">rientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
     >
```

```
11.      <EditText
12.      android:id="@+id/txtMsg"
13.      android:layout_width="fill_parent"
14.      android:layout_height="95px"
15.      android:background="#ffffff99"
16.      android:text="Enter data here"
17.      android:textSize="18sp"
18.      android:layout_x="0px"
19.      android:layout_y="17px"
20.      >
21.      </EditText>
22.      <Button
23.      android:id="@+id/btnFinish"
24.      android:layout_width="117px"
25.      android:layout_height="49px"
26.      android:background="#ffffcc99"
27.      android:text="Finish"
28.      android:textStyle="bold"
29.      android:layout_x="82px"
30.      android:layout_y="115px"
31.      android:gravity="center"
32.      >
33.      </Button>
34.      <EditText
35.      android:id="@+id/txtTodo"
36.      android:layout_width="fill_parent"
37.      android:layout_height="251px"
38.      android:background="#ffff9999"
39.      android:text="Todo"
40.      android:textSize="18sp"
```

```
41.       android:layout_x="0px"
42.       android:layout_y="181px"
43.       >
44.       </EditText>
45.       </LinearLayout>
46.
47.
```

Kết quả demo!

+ YouTube Video

Câu hỏi:

Sau bài hướng dẫn này nếu các bạn có vấn đề gì xin liên hệ với tác giả VoNguyen, email: vonguyenpt@gmail.com