

Các Callback

Sau đây là tất cả các *callback* mà bạn nên biết. Bạn không nhất thiết phải hiện thực cho tất cả các *callback* này đâu nhé. Chỉ cần dùng đủ thôi, bài thực hành bên dưới sẽ phần nào giúp bạn giải đáp ý này.

onCreate()

Theo mình thì đây là *callback* tối thiểu trong một Activity mà bạn phải hiện thực, theo mình thôi nhé, bạn có thể bỏ qua nó mà không có báo lỗi gì.

*Callback* ***onCreate()*** này được gọi khá sớm, ngay khi Activity được kích hoạt, và thậm chí người dùng còn chưa nhìn thấy gì của Activity cả, thì *callback* này đã được gọi rồi. Ngoài ra thì bạn nên biết là *callback* này chỉ được gọi một lần duy nhất khi Activity được khởi tạo. Nó có thể được gọi lại nếu hệ thống xóa Activity này đi để lấy lại tài nguyên của hệ thống, nhưng rất hiếm khi xảy ra. Và nó còn có thể được gọi lại nếu bạn xoay màn hình ngang/dọc.

Do đặc tính được gọi khá sớm và chỉ được gọi một lần duy nhất trong vòng đời của nó như vậy, nên bạn sẽ tận dụng để load giao diện cho Activity ở giai đoạn này, thông qua phương thức ***setContentView()*** mà bạn đã biết từ những ngày đầu tiên. Ngoài giao diện ra, bạn có thể khởi tạo các logic nào đó chỉ chạy một lần ban đầu, như các lời gọi API, load database, tạo item list, tạo [*Navigation Drawer*](https://yellowcodebooks.com/2017/08/28/android-bai-25-xay-dung-navigation-drawer/), và nhiều logic khác mà bạn đã từng làm quen ở *callback* này từ các bài thực hành trước.

Tham số ***savedInstanceState*** được truyền vào phương thức này sẽ được mình nói đến ở bài viết về lưu trữ trong Android sau nhé.

onStart()

Sau khi gọi đến ***onCreate()***, hệ thống sẽ gọi đến ***onStart()***. Hoặc hệ thống cũng sẽ gọi lại ***onStart()*** sau khi gọi ***onRestart()*** nếu trước đó nó bị che khuất bởi Activity nào khác (một màn hình khác hoặc một ứng dụng khác) ***che hoàn toàn*** và rơi vào ***onStop()***.

Khi hệ thống gọi đến *callback* này thì Activity chuẩn bị (có nghĩa là chưa) được nhìn thấy bởi người dùng và tương tác với người dùng. Bởi đặc tính này mà ***onStart()*** ít được dùng đến.

onResume()

Khi hệ thống gọi đến *callback* này thì bạn yên tâm rằng người dùng đã nhìn thấy và đã tương tác được với giao diện.

***onResume()*** được gọi khi Activity được khởi tạo rồi và bước qua ***onStart()*** trên kia. Hoặc khi Activity bị một giao diện nào khác ***che đi một phần (hoặc toàn phần)***, rồi sau đó quay lại Activity hiện tại. Bạn có thể thấy rằng *callback* này được gọi rất nhiều lần trong một vòng đời của nó.

Chính đặc điểm này của ***onResume()*** mà bạn có thể tận dụng để quay lại tác vụ mà người dùng đang bị dang dở khi ***onPause()*** (được nói đến dưới đây) được gọi.

Chẳng hạn như bạn đang soạn nội dung cho TourNote, mà có cuộc gọi đến, bạn sẽ lưu tạm nội dung này khi *callback* ***onPause()***, để rồi khi ***onResume()*** được gọi lại sau đó khi người dùng kết thúc cuộc gọi và quay lại TourNote, bạn sẽ khôi phục nội dung đó để người dùng tiếp tục sử dụng TourNote như chưa có bất kỳ gián đoạn nào.

onPause()

Thông thường nếu có một thành phần nào đó che Activity hiện tại mà người dùng vẫn nhìn thấy Activity đó (nhìn thấy chứ không tương tác được). Chẳng hạn một popup hiện lên trên Activity. Thì ***onPause()*** của Activity sẽ được gọi. Sau này khi người dùng quay lại Activity thì ***onResume()*** sẽ được gọi.

Bạn có thể tưởng tượng rằng ***onPause()*** cũng sẽ được gọi khá nhiều lần trong một vòng đời Activity. Theo như Google thì ***onPause()*** được gọi đến khá nhanh, nếu bạn muốn lưu trữ dữ liệu như mình nói trên kia, thì nên lưu những gì nhanh gọn lẹ thôi. Nếu bạn muốn lưu trữ các dữ liệu nặng, hoặc gọi API kết nối server chỗ này, nhiều khả năng ứng dụng sẽ không kịp thực hiện. Do đó, thay vì làm các thao tác nặng nề ở ***onPause()***, bạn có thể cân nhắc gọi chúng ở ***onStop()***.

onStop()

Như mình có nói. ***onStop()*** được gọi khi Activity không còn được nhìn thấy nữa, có thể một màn hình nào khác che lên hoàn toàn, có thể một ứng dụng nào đó vào foreground, hoặc người dùng nhấn nút *Home* để về màn hình chính.

Bạn có thể tận dụng ***onStop()*** để lưu trữ dữ liệu ứng dụng. Hoặc để giải phóng các tài nguyên đang dùng. Ngưng các API còn đang gọi dang dở.

Tuy nhiên khi ***onStop()*** được gọi không phải là lúc chúng ta cũng nói lời tạm biệt Activity. Như mình đã nói, người dùng hoàn toàn có thể quay lại sử dụng Activity sau đó mà không cần phải khởi động lại Activity, khi này thì phương thức ***onRestart()*** và ***onStart()*** được gọi kế tiếp nhau.

onDestroy()

Trước khi Activity *“chết”*, hệ thống cho nó cơ hội để nói lời trăn trối, nó sẽ giúp gọi đến *callback* ***onDestroy()*** này của Activity.

Bạn có thể tận dụng *callback* này để giải phóng các tài nguyên hệ thống mà ở ***onStop()*** bạn chưa gọi đến.

Vòng đời của một Activity kết thúc ở đây.

## Các Trạng Thái Chính Trong Vòng Đời Activity

### Hoạt Động (Active)

Khi Activity được kích hoạt, và được hệ thống để vào BackStack, nó sẽ bước vào trạng thái active. Với trạng thái active, người dùng hoàn toàn có thể nhìn thấy và tương tác với Activity của ứng dụng.

### Tạm Dừng (Pause)

Trạng thái này khá đặc biệt. Trạng thái tạm dừng. Như bạn đã làm quen trên kia, trạng thái này xảy ra khi mà Activity của bạn vẫn đang chạy, người dùng vẫn nhìn thấy, nhưng Activity khi này lại bị ***che một phần*** bởi một thành phần nào đó. Chẳng hạn như khi bị một dialog đè lên (bạn sẽ được kiểm chứng khi học qua bài về dialog sau).

Bạn nhớ nhé, cái sự che Activity này không phải hoàn toàn. Chính vì vậy mà Activity đó tuy được người dùng nhìn thấy nhưng không tương tác được.

### Dừng (Stop)

Trạng thái này khá giống với trạng thái tạm dừng trên kia. Nhưng khi này Activity bị ***che khuất hoàn toàn*** bởi một thành phần giao diện nào đó, hoặc bởi một ứng dụng khác. Và tất nhiên lúc này người dùng không thể nhìn thấy Activity của bạn được nữa.

Hành động mà khi người dùng nhấn nút Home ở System Bar để đưa ứng dụng của bạn về background, cũng khiến Activity đang hiển thị trong ứng dụng rơi vào trạng thái dừng này. Bạn sẽ biết khi đi đến bài thực hành dưới đây.

### Chết (Dead)

Nếu Activity được lấy ra khỏi BackStack, chúng sẽ bị hủy và vào trạng thái này. Trường hợp này xảy ra khi user nhấn nút Back ở System Bar để thoát một Activity. Hoặc [lời gọi hàm finish()](https://yellowcodebooks.com/2017/09/06/android-bai-26-lam-quen-voi-activity/" \l "Xay_Dung_Chuc_Nang_Back_Tren_ActionBar_Cua_ContactActivity" \t "_blank) từ một Activity để “giết chính nó”. Cũng có khi ứng dụng ở trạng thái background quá lâu, hệ thống có thể sẽ thu hồi tài nguyên bằng cách dừng hẳn các Activity trong ứng dụng, làm cho tất cả các Activity đều vào trạng thái này.

Khi vào trạng thái dead, Activity sẽ kết thúc vòng đời “trôi nổi” của nó.

Những ý trên giúp bạn nắm được tổng quan các trạng thái mà một Activity có thể trải qua. Chúng ta cùng đi gần đến code hơn bằng cách nói đến công dụng của từng callback.