I. Các thành phần của một ứng dụng Android

Tìm hiểu cấu tạo cơ bản một ứng dụng Android gồm các thành phần nào cũng như những loại ứng dụng Android như Activity, Intent, Manifest, Context

Phần này tìm hiểu các thành phần cơ bản nhất tạo nên một ứng dụng Android. Hiểu về các thành phần này giúp cho có một cái nhìn tổng quan về cấu trúc ứng dụng.

1.Các Activity trong Android

Các ứng dụng được tạo ra bằng sự hòa hợp với một hay nhiều thành phần được gọi là **Activity**. Một Activity có thể là một module, thành phần chức năng độc lập của ứng dụng mà mỗi Activity thường tương ứng với một giao diện người dùng (UI) và các chức năng đáp lại sự tương tác với người dùng. Một ứng dụng có thể có một hay nhiều Activity

Các Activity được xây dựng để có thể sử dụng lại và tương tác, chia sẻ giữa các ứng dụng. Ví dụ có một ứng dụng với Activity để hiện thị hình ảnh, thì trong ứng dụng khác thay vì xây dựng mới Activity có chức năng này thì có thể gọi từ ứng dụng khác.

Một Activity được xây dựng bằng cách kế thừa lớp Activity, một Activity không thể gọi trực tiếp các phương thức hay truy cập vào dữ liệu của một Activity khác. Thay vào đó, phải dùng tới thành phần gọi là Intent từ Activity này kích hoạt Activity khác và Content Provider. Mặc định thì một Activity không trả về kết quả cho một Activity khác gọi đến nó

2.Các Intent trong Android

Các **Intent** là cơ chế để một Activity có thể khởi chạy Activity khác trong ứng dụng. Trong **Intent** có thông tin về nhiệm vụ, các tùy chọn, dữ liệu để Activity thi hành được. Intent có thể là dạng tường minh, nghĩa là chúng yêu cầu chạy một Activity cụ thể chỉ ra bởi tên lớp của Activity đó. Intent cũng có thể ở dạng không tường mình, bằng cách chỉ cung cấp hoạt động mong muốn, hoặc cung cấp dữ liệu mà căn cứ vào dữ liệu đó tương ứng với hành động, hệ thống Android ở thời điểm chạy sẽ chọn Activity để kích hoạt.

3.Broadcast Intent

Đây là một Intent đặc biệt, Broadcast Intent sẽ gửi thông tin ra bên ngoài, gửi đến tất cả các ứng dụng (ứng dụng có đăng ký nhận bằng Broadcast Receiver). Ví dụ, hệ thống Android gửi Broadcast Intent để thông báo trạng thái thiết bị như khởi động hệ thống, thay đổi kết nối ... ứng dụng của bạn muốn nhận thông tin nào thì phải đăng ký trước.

4.Broadcast Receiver

Đây là cơ chế để ứng dụng phản ứng lại với các Broadcast Intent, nó cần được đăng ký bởi ứng dụng và cấu hình với Intent Filter để biết được loại Intent nào muốn nhận. Nếu Intent Broadcast từ đâu đó trong hệ thống phát ra, phù hợp với loại ứng dụng đăng ký thì nó sẽ nhận được khi chạy. Khi nhận được thì Receiver có 5 giây để hoàn thành tác vụ (như tạo notification chẳng hạn). Broadcast Receiver thi hành ở background (nền) và không có giao diện người dùng.

5.Broadcast Services

Android Service là process (tiến trình) chạy ngầm (background) và không có giao diện người dùng. Nó có thể khởi chạy và quản lý bởi các Activity, Broadcast Receiver, hay bởi Service khác. Android Service dùng trong tình huống một ứng dụng cần tiếp tục thi hành các tác vụ nhưng không cần tới sự xuất hiện của giao diện người dùng.

Cho dù Service thiếu đi giao diện người dùng, nó vẫn có thể thông báo cho người dùng các tự kiện bằng sử dụng Notification hoặc Toast. Service có mức độ ưu tiên cao hơn khi chạy so với các loại tiến trình khác, cần giải phóng tài nguyên nó được xem xét kết thúc sau nhất và tất nhiên trong quá trình này Service sẽ tự động khởi động lại ngay khi có đủ tài nguyên hệ thống. Một Service có thể giảm đi nguy cơ kết thúc bằng cách khai báo bản thân nó chạy ở nền, bằng cách gọi hàm startForeground()

6.Content Provider

Content Provider là cơ chế cho phép chia sẻ dữ liệu giữa các ứng dụng với nhau. Bất kỳ ứng dụng nào đều có thể cung cấp cho ứng dụng khác khả năng truy cập dữ liệu của mình thông qua Content Provider với các chức năng thêm, bớt, truy vấn dữ liệu. Việc truy cập này được cung cấp thông qua URI định nghĩa và cung cấp bởi Content Provicder

Dữ liệu chia sẻ ở dạng một file hoặc CSDL SQLite. Các ứng dụng gốc Android cung cấp sẵn nhiều Content Provider cho phép ứng dụng truy cập dữ liệu như trình danh bạ, quản lý file media. Hiện nay Content Provider tồn tại trong hệ thống Android với Content Resolver.

7.Manifest

Manifest của ứng dụng định nghĩa trong file định dạng XML, đây là file chứa các cấu hình cho ứng dụng. Những thông tin này cần thiết để hệ thống Android chạy được ứng dụng như mong muốn.

8.Resource

Chứa toàn bộ các hợp file tài nguyên của dự án (Resource) như chuỗi ký tự, hình ảnh, font chữ, màu sắc ... những thành phần xuất hiện trong giao diện người dùng và được trình bày với file XML. Mặc định những file này lưu trữ bên trong thư mục /res

9.Context ứng dụng

Khi ứng dụng biên dịch, một lớp có tên là **R** được tự động tạo ra, nó chứa các tham khảo trỏ đến tài nguyên của ứng dụng. Các file manifest và tài nguyên sẽ được kết hợp lại với nhau được hiểu là Context ứng dụng. Context trong ứng dụng được biểu diễn bằng lớp Context, được sử dụng trong ứng dụng để thông qua nó truy cập tới các loại tài nguyên khi ứng dụng đang chạy. Ngoài ra, có nhiều phương thức dựa vào context để lấy các thông tin về môi trường ứng dụng hoạt động khi chạy.

II. Activity và vòng đời Activity

Khái niệm cơ bản nhất về Activity trong Android, tìm hiểu chi tiết về vòng đời hoạt động của một Activity từ khỉ kích hoạt đến khi kết thúc

1.Giới thiệu về Activity trong Android

Lớp **Activity** là thành phần quan trọng nhất của ứng dụng Android, cách mà chúng hoạt động tạo thành nền tảng cơ bản của mô hình lập trình ứng dụng. Android khởi chạy một ứng dụng thông thường bằng kích hoạt một Activity tương ứng với vòng đời cụ thể của nó trong quá trình hoạt động.

Thường một Activity cung cấp một của sổ, ở đó ứng dụng sẽ dựng các thành phần UI (User Interface - giao diện người dùng). Mặc định cửa sổ này là đầy màn hình thiết bị, nhưng có một vài trường hợp riêng sẽ nhỏ hơn và nổi phía trên cửa sổ khác.

Hầu hết các ứng dụng đều sử dụng nhiều màn hình khác nhau, có nghĩa nó sẽ phải có nhiều Activity khác nhau. Khi một Activity chỉ định là Activity chính, nó sẽ là màn hình đầu tiên khi khởi chạy ứng dụng. Một Activity này lại có thể gọi và kích hoạt một Activity khác. Ví dụ một Activity hiện thị danh sách các ghi chú, nó gọi một Activity khác để xem nội dung chi tiết của ghi chú.

2.Vòng đời của Activity

Để tạo một Activity thì bạn phải tạo ra một lớp kế thừa lớp Activity, sau đó triển khai tối thiểu phương thức **onCreate(Bundle savedInstanceState)**, sau đó tùy ngữ cảnh mà khi Activity hoạt động vòng đời của nó diễn ra như mô tả ở hình sau:



Nhìn vào hình, ta thấy có các mốc chính trong quá trình hoạt động của Activity tương ứng với các phương thức mà bạn có thể quá tải là:

* **onCreate(Bundle savedInstanceState)**: Được gọi khi hoạt động mới được tạo, tại đây khởi tạo các biến, nạp giao diện layout ..., phương thức này cũng nhận dữ liệu lưu lại trạng thái hoạt động trước đó (với mục đích để phục hồi - savedInstanceState). Sau sự kiện này bao giờ cũng gọi ngay lập tức **onStart()**.
* **onStart()**: Được gọi ngay trước khi Activity hiển thị trên màn hình.
* **onResume()**: Được gọi ngay khi Activity bắt đầu có thể tương tác với người dùng, và Activity nằm trên cùng trong danh sách các Activity của hệ thống. Sau phương thức này là các chức năng của Activity hoạt động dựa trên tương tác của người dùng ..., cho đến khi có một nguyên nhân nào đó mà phương thức OnPause được gọi.
* **onPause()**: Được gọi khi hệ thống sắp kích hoạt một Activity khác, nếu bạn quá tải phương thức này, thường để lưu lại dự liệu thật nhanh để hệ thống còn kích hoạt Activity khác. Ngay sau phương thức này nó sẽ gọi **onResume()** nếu Activity được kích hoạt lại ngay, hoặc gọi **OnStop()** nếu Activity bị ẩn đi.
* **onStop()**: Được gọi khi nó bị ẩn đi. Sau phương thức này, Activity có thể gọi **onRestart()** nếu nó được người dùng kích hoạt lại hoặc gọi **onDestroy()** để hết thúc.
* **onDestroy()**: gọi khi Activity bị hủy hoàn toàn (ví dụ gọi finish(), hoặc người dùng kill Activity)

# III. View, ViewGroup, Layout trong Android

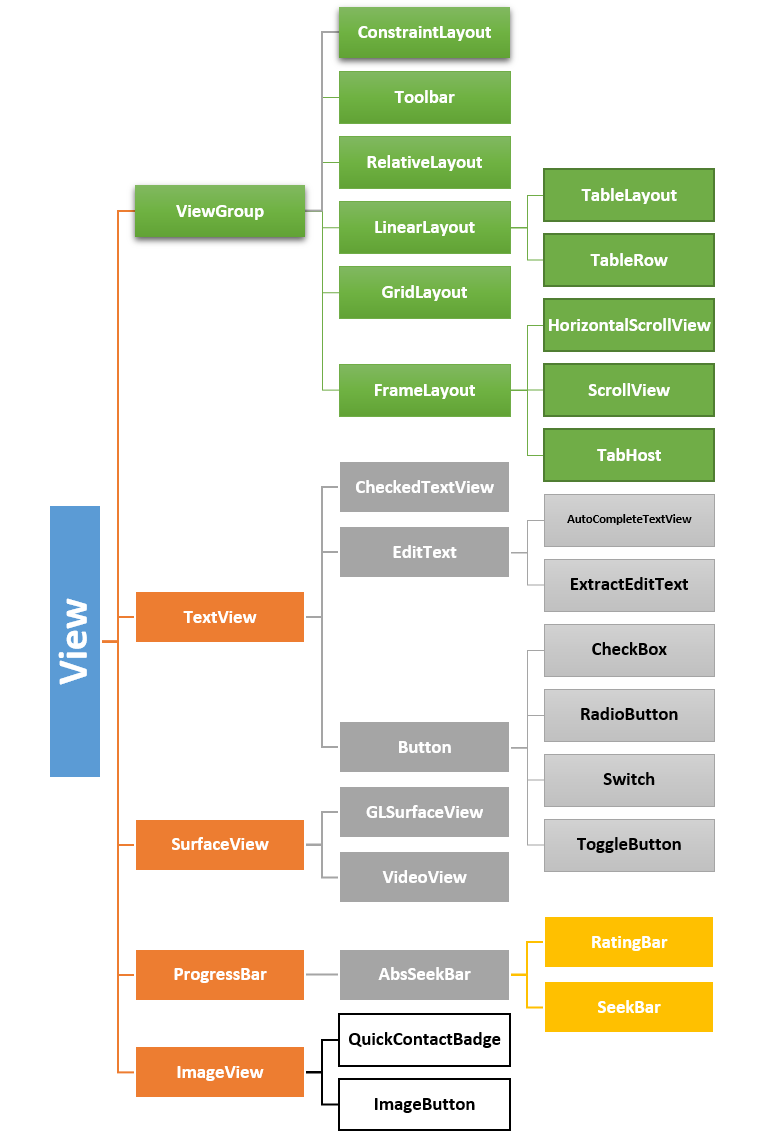
Các thành phần cơ bản để xây dựng giao diện người dùng (UI) trong Android là View, ViewGroup và Layout. Từ các thành phần cơ sở đó kết hợp chúng với nhau tạo ra các loại UI phức tạp cho ứng dụng Android

Để tạo ra một giao diện người dùng trong ứng dụng Android, chúng ta xây dựng từ sự kết hợp các thành phần có tên là **View**, **ViewGroup**, **Layout**.

## 1.View

Các thành phần giao diện xây dựng từ lớp cơ sở **View** ([android.view.View](https://developer.android.com/reference/android/view/View.html)) của Android, các thành phần này cung cấp sẵn khá đa dạng như Button, TextView, CheckBox ... tất cả chúng ta gọi nó là **View**. Sơ đồ các View được mộ tả theo sơ đồ như hình dưới.

**View** biểu diễn một hình chữ nhật, trong đó nó hiện thị thông tin nào đó cho người dùng, và người dùng có thể tương tác với View. Nhưng loại View cơ bản cần tìm hiểu trước tiên đó là: TextView, ImageView, Button, ImageButton, EditText

[](https://4.bp.blogspot.com/-r7fkm4ahE1Y/WrCl9dFfGsI/AAAAAAAAQ-8/7sbIuWZ69TY-dn_Y4zYkDresu_XRLr7uACLcBGAs/s1600/cautrucview.png)

## 2.ViewGroup

Trong sơ đồ trên, có một lớp abstract kế thừa từ View là **ViewGroup**, nó cũng chính là một View nhưng có khả năng chứa các View khác bên trong (kể cả ViewGroup khác). Nó là lớp cơ sở để xây dựng nên các layout như trên sơ đồ ta thấy có: **ConstraintLayout, RelativeLayout, LinearLayout, GridLayout, FrameLayout**

Các ViewGroup cũng như một phần tử chứa các View con, và ViewGroup cũng được kế thừa trở thành phần gốc để xây dựng UI gọi là **layout**

## 3.Layout

Các layout chính là các View (cụ thể nó kế thừa thừa ViewGroup) được thiết kế với mục đích chứa các View con và điều khiển, sắp xếp vị trí các View con đó trên màn hình, mỗi layout có cơ chế điều khiển vị trí View con riêng của mình. Có một số layout mà bạn tham khảo trước tiên như:

* **FrameLayout** đơn giản chỉ cung cấp một vùng màn hình, thường dùng nó để hiện thị một View con duy nhất. Nếu đặt vào nó nhiều view con, thì mặc định các view con này sẽ xếp chồng lên nhau. Tuy vậy, vị trí các view con trong nó cũng có thể điều chỉnh thông qua giá trị tham số gravity, ví dụ trong view con thiết lập *android:layout\_gravity="bottom|right"* thì view con nằm về phía trái, dưới của FrameLayout
* **ConstraintLayout** (giới thiệu trong Android 7), sử dụng layout này được khuyến khích cho hầu hết trường hợp. **ConstraintLayout** cho phép điều khiển vị trí và ứng sử của các view con trong layout bằng cách gán dàng buộc đơn giản vảo mỗi view con. Từ đó mà một bố cục phức tạp có thể dễ dàng được tạo ra mà sử dụng ít nhất sự lồng nhau trong layout (layout này nằm trong layout khác) giúp cho cải thiện tốc độ. ConstraintLayout cũng tích hợp sẵn vào Android Studio Layout Editor nên bạn có thể điều chỉnh một cách trực quan các View con trong layout này.
* **LinearLayout** với layout này thì các view con được xếp nối tiếp nhau (linear) thành một hàng (**vertical**) hay một cột (**horizontal**) (tùy vào lúc thiết kế thiết lập hướng xếp). Có một giá trị weight có thể gán vào mỗi View con để cho biết View con đó chiếm bao nhiêu không gian trong một tỷ lệ tương quan với các View con khác.
* **RelativeLayout** cho phép các view con định vị căn vào liên hệ với các view con khác đồng thời liên hệ với view cha thông qua các tham số align và margin.
* **GridLayout** chia ra thành lưới gồm một số hàng và một số cột để chứa các view con.
* **TableLayout** cung cấp khả năng bố trí các view con thành một lưới dạng bảng (gồm có hàng và cột). Một dòng của bảng biểu diễn bàng đối tượng view con **TableRow**, trong nó chứa có phần tử View con hiểu như các ô bảng.
* **CoordinatorLayout** nó được thiết kế nhằm mục đích có sự tương tác của các View con trong nó, đặc biệt sử dụng với ActionBar, FloatingActionButton, Snackbar ...

## 4.Một số thuộc tính XML trong tất cả các View

|  |  |
| --- | --- |
| **Thuộc tính** | **Diễn tả** |
| android:alpha | Nhận giá trị từ 0 - 1 là kênh alpha (độ trọng suốt) của View. Nhận 0 trong suốt hoàn toàn. |
| android:background | Thiết lập nền bằng màu sắc, drawable ... cho View |
| android:backgroundTint | Thiết lập màu sẽ nhuộm vào nền |
| android:clickable | Thiết lập true thì View có khả năng nhận sự kiện click |
| android:foreground | Gán một ảnh Drawable để vẽ vào phần nội dung View |
| android:foregroundGravity | Định vị foreground trong View |
| android:id | Gán một ID vào View, ví dụ android:id="@+id/imagebutton" |
| android:layout\_witdth | Xác định chiều rộng của View  "match\_parent": kích thước bằng View chứa nó  "wrap\_content": kích thước bằng nội dung của View |
| android:layout\_height | Xác định chiều cao của View  "match\_parent": kích thước bằng View chứa nó  "wrap\_content": kích thước bằng nội dung của View |
| android:layout\_gravity | Thiết lập vị trí của nội View trong khung chứa nó |
| android:padding | Thiết lập kích thước Padding cho cả 4 cạnh |
| android:paddingLeft | Thiết lập Padding cạnh trái |
| android:paddingRight | Thiết lập Padding cạnh phải |
| android:paddingTop | Thiết lập Padding cạnh trên |
| android:paddingBottom | Thiết lập Padding cạnh dưới |
| android:scaleX | Tỷ lệ thu/phong View theo chiều X |
| android:scaleY | Tỷ lệ thu/phong View theo chiều Y |
| android:tag | Gán một giá trị vào làm tag của View, sau này có thể đọc lại bằng getTag() và có thể tìm lại View bằng findViewWithTag() |
| android:theme | Thiết lập theme cho View |
| android:visibility | Gán các giá trị: gone (ẩn hoàn toàn), invisible (không hiện thị - nhưng chỗ nó chiếm vẫn còn), visible (hiện thị) |
| android:translationX | Dịch chuyển view theo chiều X |
| android:translationY | Dịch chuyển view theo chiều Y |
|  |  |

IV. LinearLayout trong Android

1.LinearLayout trong Android

LinearLayout trong Android là một layout mà nó sắp xếp các view con một cách liên tục theo một hàng (**vertical**) hay một cột (**horizontal**), có một giá trị weight có thể gán vào mỗi View con để cho biết View con đó chiếm bao nhiêu không gian trong một tỷ lệ tương quan với các View con khác.

2.Các thuộc tính của LinearLayout

**android:orientation :** để thiết lập cách sắp xếp phần tử.

android:orientation="horizontal" xếp theo chiều ngang.

android:orientation="vertical" xếp theo chiều đứng.

**android:gravity :** để căn chỉnh các View nằm ở vị trí nào trong LinearLayout, nó nhận các giá trị (có thể tổ hợp lại với ký hiệu |)

|  |  |
| --- | --- |
| **Giá trị** | **Ý nghĩa** |
| center | Căn ở giữa |
| top | Ở phần trên |
| bottom | Phần dưới |
| center\_horizontal | Ở giữa theo chiều ngang |
| center\_vertical | Ở giữa theo chiều đứng |
| left | Theo cạnh trái |
| right | Theo cạnh phải |
| bottom | Cạnh dưới |

**android:layout\_weight :** Gán trọng số cho View con trong LinearLayout. Nếu View không gán giá trị này coi như nó có trọng số bằng không.

Giá trị trọng số này sẽ được LinearLayout sử dụng để điều chỉnh kích thước View con có trọng số (*điều chỉnh chiều cao nếu là loại LinearLayout đứng và điều chỉnh chiều rộng nếu là loại ngang*)

Xét sự ảnh hưởng của trọng số với hai trường hợp sau:

**Trường hợp 1**

Trường hợp LinearLayout không sử dụng đến thuộc tính **android:weightSum**

**1** Đầu tiên các View không gán trọng số android:layout\_weight sẽ có kích thước (*rộng hay là cao tùy vào LinearLayout là ngang hay đứng*)sẽ theo đúng như thuộc tính *android:layout\_width, android:layout\_height* của nó.

**2** Không gian (kích thước) còn lại của LinearLayout sẽ chia cho các View có trọng số. Cái nào trọng số lớn hơn sẽ có kích thước lớn hơn. Nếu gọi kích thước còn lại là Size, gọi tổng các trọng số của các View là SUM, thì một view có trọng số là weight sẽ có kích thước là:  
**weight\*(Size/SUM)**

Lưu ý các kích thước mà gán match\_parent không bị ảnh hưởng của trọng số

**Trường hợp 2**

Trường hợp này trong LinearLayout thiết lập thuộc tính **weightSum** ví dụ như *android:weightSum="10"*, thì lúc này giá trị **SUM** trong công thức phân bổ kích thước ở trên không phải là lấy từ tổng weight các View con mà lấy bằng đúng android:weightSum của LinearLayout, sau đó vẫn dùng công thức phân bổ kích thước như trên: **weight\*(Size/SUM)**

# V. RelativeLayout trong Android

## 1.RelativeLayout trong Android

**RelativeLayout** là một dạng View Group quy định vị trí hiển thị của các View con. Vị trí của View con sẽ được hiển thị thông qua quan hệ của View đó với View khác hoặc so với View Group cha của nó

**RelativeLayout** là một layout hết sức mạnh mẽ về độ tiện lợi và hiệu quả, nếu giao diện không ở mức quá phức tạp việc chọn RelativeLayout mạng lại hiệu suất còn tốt hơn [ConstraintLayout](https://xuanthulab.net/su-dung-constraintlayout-trong-android.html). RelativeLayout dùng khi đơn giản, [ConstraintLayout](https://xuanthulab.net/su-dung-constraintlayout-trong-android.html) khi giao diện phức tạp.

Khi các View con đưa vào **RelativeLayout** nếu chưa có thiết lập mối liên hệ qua lại nào với phần tử cha hay với phần tử View con khác thì nó sẽ được định vị ở góc trên - trái của **RelativeLayout**. Như trường hợp dưới đây cả 3 View con không có thiết lập mối liên hệ nào, nên nó đều định vị ở góc trên / trái và vẽ chồng nên nhau, View con nào xếp sau sẽ ở lớp trên của màn hình.

## 2.Các thuộc tính của RelativeLayout

**android:gravity : Xác định vị trí của các View con trong View cha**, nó nhận các giá trị (có thể tổ hợp lại với ký hiệu | giống với [LinearLayout](https://xuanthulab.net/su-dung-linearlayout-trong-android.html))

|  |  |
| --- | --- |
| **Giá trị** | **Ý nghĩa** |
| center | Căn ở giữa |
| top | Ở phần trên |
| bottom | Phần dưới |
| center\_horizontal | Ở giữa theo chiều ngang |
| center\_vertical | Ở giữa theo chiều đứng |
| left | Theo cạnh trái |
| right | Theo cạnh phải |
| bottom | Cạnh dưới |

**android:ignoreGravity="id-view-con" : C**hỉ ra một View con tách khỏi khối biên chữ nhật chứa các View con để phần tử đó không bị ảnh hưởng bởi **gravity**

|  |  |
| --- | --- |
| **Thuộc tính** | **Ý nghĩa** |
| android:layout\_alignParentBottom | true căn thẳng cạnh dưới view con với cạnh dưới View cha |
| android:layout\_alignParentLeft | true căn thẳng cạnh trái view con với cạnh trái View cha |
| android:layout\_alignParentRight | true căn thẳng cạnh phải view con với cạnh phải View cha |
| android:layout\_alignParentTop | true căn thẳng cạnh trên view con với cạnh trên View cha |
| android:layout\_centerInParent | true căn view con vào giữa View cha |
| android:layout\_centerHorizontal | true căn view con vào giữa View cha theo chiều ngang |
| android:layout\_centerVertical | true căn view con vào giữa View cha theo chiều đứng |

|  |  |
| --- | --- |
| **Thuộc tính** | **Ý nghĩa** |
| android:layout\_below | Nằm phía dưới View có ID được chỉ định |
| android:layout\_above | Nằm phía trên View có ID được chỉ định |
| android:layout\_toLeftOf | Nằm phía trái View có ID được chỉ định |
| android:layout\_toRightOf | Nằm phía phải View có ID được chỉ định |
| android:layout\_alignBottom | Căn thẳng cạnh dưới với cạnh dưới của View có ID được chỉ định |
| android:layout\_alignLeft | Căn thẳng cạnh trái với cạnh trái của View có ID được chỉ định |
| android:layout\_alignRight | Căn thẳng cạnh phải với cạnh phải của View có ID được chỉ định |
| android:layout\_alignTop | Căn thẳng cạnh trên với cạnh trên của View có ID được chỉ định |

VI. FrameLayout trong Android

1. FrameLayout trong Android

**FrameLayout** là loại View cơ sở, nó là loại Layout đơn giản nhất. Mặc dù nó có thể chứa nhiều View con bên trong, nhưng mục đích chính thiết kế ra nó để chứa một View, từ đó nó trở thành cơ sở để tạo ra các View khác phức tạp hơn. Khi thiết kế Layout chứa nhiều View thì không nên sử dụng layout này, vì nó quá đơn giản việc bố cục các View con trong nó rất khó khăn (nó không có các tính năng điều khiển vị trí View con sao cho việc độc lập về màn hình được đảm bảo).

Nếu bạn vẫn sử dụng **FrameLayout** để thiết kế layout, thì cần lưu ý: Các View con đặt vào FrameLayout nằm chồng nên nhau theo thứ tự cái nào đưa vào sau thì hiện thị ở lớp trước, mỗi View con chỉ có thể điều chỉnh vị trí nó thông qua thuộc tính **android:layout\_gravity** gán cho View con

2. Các thuộc tính của FrameLayout

**android:layout\_gravity :** xác định vị trí của View con so với View cha

Các giá trị có thể kết hợp bằng ký hiệu |

|  |  |
| --- | --- |
| **giá trị** | **Vị trí của View con** |
| bottom | Nằm dưới FrameLayout |
| center | Nằm giữa FrameLayout |
| center\_horizontal | Giữa theo chiều ngang |
| center\_vertical | Giữa theo chiều đứng |
| end | Cuối FrameLayout |
| left | Bên trái |
| right | Bên phải |
| start | Bắt đầu FrameLayout |
| top | Trên FrameLayout |

# VII. TextView trong Android

## 1. TextView trong Android

**TextView** là một View cho phép hiện thị các dòng chữ (text) trên màn hình, nó có nhiều thuộc tính tùy mục đích sử dụng mà áp dụng, như thiết lập cỡ chữ, font chữ, màu chữ ...

## 2. Các thuộc tính của TextView

|  |  |
| --- | --- |
| **#** | **Thuộc tính** |
| 1 | **ID của TextView**  android:id để thiết lập id của TextView, ví dụ *android:id = "@+id/mytextview"* |
| 2 | **Kích thước TextView**  android:layout\_width, android:layout\_height thiết lập chiều rộng, chiều cao của TextView, gán giá trị theo các đơn vị như dp, px, mm ... ví dụ: *android:layout\_margin="5mm"*, *android:layout\_margin="10dp"* ... hoặc gán chúng bằng hằng số wrap\_content để kích thước vừa nội dung, hằng số match\_parent để căn vừa phần tử cha |
| 3 | **Vị trí của chữ trong TextView**  android:gravity để thiết lập vị trí tương đối của các dòng chữ bên trong TextView, giá trị nhận phối hợp từ các hằng số : top bottom left right clip\_verticalclip\_horizontal start. Ví dụ để các dòng chữ nằm về phía bên trái, phía dưới của TextView android:gravity="right|bottom" |
| 4 | **Gán text vào TextView**  android:text để gán dòng chữ. Ví dụ *android:text="TextView Demo"*, một cách dùng phổ biến nữa là nội dung text định nghĩa trong strings.xml ví dụ:  Gán dòng text có tên *textviewcontent* vào TextView: android:text="@string/textviewcontent" |
| 5 | **Thiết lập màu chữ, màu nền**  android:textColor để thiết lập màu chữ, gán giá trị màu theo số HEX của màu như *#b96e6e*, *#111* ... hoặc một số màu hệ thống định nghĩa trước *@android:color/holo\_purple*, *@android:color/black* ...  Ví dụ thiết lập màu chữ thành màu đỏ: *android:textColor="@android:color/holo\_red\_dark"*  bất kỳ  **Gán Drawable vào các biên TextView**  Dùng các thuộc tính **android:drawableLeft, android:drawableRight, android:drawableTop, android:drawableBottom** để gán các ảnh Drawable vào biên trái, phải, trên, dưới của TextView.  Nếu muốn thiết lập khoảng cách các ảnh Drawable đến vùng nội dung dùng thuộc tính **android:drawablePadding** |
| 6 | **Cỡ chữ**  android:textSize để thiết lập cỡ chữ, gán các giá trị theo đơn vị *sp, dp, mm*, đối với cỡ chữ để độc lập thiết bị tốt nhất dùng đơn vị **sp**. Ví dụ: *android:textSize="12dp"* |
| 7 | **Kiểu chữ**  android:textStyle để thiết lập định dạng chữ, có thể nhận phối hợp các giá trị *bold*(đậm), *italic* (nghiêng), *normal* (bình thường). Ví dụ thiết lập chữ đậm, nghiêng:android:textStyle="bold|italic" |
| 8 | **Padding trong TextView**  Padding để thiết lập vùng trình bày nội dung trong TextView, cách các cạnh của TextView bao nhiêu. Nếu dùng android:padding thì thiết lập cách đều 4 cạnh một khoảng giá trị chỉ ra, ví dụ *android:padding="8dp"* thì phần nội dung cách cách cạnh 8dp  Cũng có thể thiết lập khoảng cách này theo từng cạnh tương ứng với thuộc tính:android:paddingLef, android:paddingTop, android:paddingRight, android:paddingBottom |
| 9 | **Thiết lập Font chữ**  android:fontFamily dùng để thiết lập font chữ cho TextView, tùy vào hệ thống Android phiên bản nào, mà có một số giá trị khác nhau. Ví dụ có thể thiết lập một số font chữ mặc định như **"monospace", "serif-monospace"** ... Ví dụ, thiết lập font chữ serif-monospace: android:text="serif-monospace" |
| 10 | **Một số thuộc tính khác trong TextView**   * android:hint để gán dòng chữ sẽ xuất hiện nếu android:text rỗng. * android:textColorHint gán màu cho Hint * android:maxHeight : thiết lập kích thước tối đa cho chiều cao * android:maxWidth : thiết lập kích thước tối đa cho chiều rộng * android:minHeight : thiết lập kích thước tối thiểu cho chiều cao * android:minWidth : thiết lập kích thước tối thiểu cho chiều rộng.   Tuy nhiện các 4 thuộc tính trên chỉ tác động khi android:layout\_width, android:layout\_height thiết lập giá trị wrap\_content   * android:textAllCaps="true" tất cả các chữ chuyển thành chữ in hoa (gán false để tắt chế độ này ) |

# VIII. EditText trong Android

## 1. EditText trong Android

**EditText** là loại View hiện thị một hộp (chữ nhật) cho phép người dùng nhập dữ liệu (chữ, số ... có thể khống chế nhập liệu là text, số, phone, ngày tháng ...).

## 2. Các thuộc tính của EditText

Do **EditText** mở rộng chức năng từ **TextView**, nến các tùy chọn thiết lập trình bày ở TextView vẫn đúng cho EditText như: *màu chữ, font chữ, màu nền, hint, gán drawable vào các biên ...* Nên các đặc tính đó không trình bày chi tiết

|  |  |
| --- | --- |
| **#** | **Thuộc tính** |
| 1 | **Thiết lập kiểu nhập liệu**  **android:inputType :** dùng để thiết lập kiểu nhập liệu, hiện thị. Tương ứng với mỗi kiểu này EditText sẽ có cách hiện thị cũng như liên kết với loại bàn phím tương ứng. Ví dụ như thiết lập nhập password thì dữ liệu nhập vào sẽ hiện thị bằng ký hiệu \*, hay chọn dữ liệu số thì bàn phím xuất hiện là loại bàn phím số  Các hằng số gán vào **android:inputType** có thể kết hợp nhiều loại với nhau bằng phép toán |, tham khảo đầy đủ tại: [inputType](https://developer.android.com/reference/android/R.styleable.html#TextView_inputType), ở đây là một số giá trị hay dùng:   |  |  | | --- | --- | | Hằng XML | Ý nghĩa | | date | Nhập ngày tháng | | datetime | Nhập ngày tháng, giờ | | number | Nhập số | | numberDecimal | Nhập số thập phân | | numberSigned | Nhập số nguyên không dấu | | phone | Nhập số diện thoại | | text | Nhập văn bản | | textEmailAddress | Địa chỉ Email | | textMultiLine | Chữ trên nhiều dòng | | textPassword | Nhập password | | textUri | Địa chỉ URL | | time | Thời gian | |
| 2 | **Giới hạn dòng chữ nhập trên một dòng**  **android:maxlines** |

IX. ImageView trong Android

1. ImageView

**ImageView** là loại View dùng để hiện thị tài nguyên hình ảnh như các ảnh Bitmap, các ảnh Drawable. Nó cũng cung cấp các chức năng tùy biến khác nhau như đổ màu nhuộm (tint) vào ảnh, co kéo/cắt ảnh khi hiện thị trên View.

2. Thuộc tính của ImageView

|  |  |
| --- | --- |
| **#** | **Thuộc tính** |
| 1 | **Thiết lập thu phóng ảnh**  **android:scaleType :** dùng để thiết lập thu phóng ảnh, nhận các giá trị như: *fitXY, center, fitXY* ..., ví dụ như **android:scaleType="center"**   |  |  | | --- | --- | | **Giá trị** | **Sử dụng** | | center | ImageView.ScaleType.CENTER : Đặt ảnh vào giữa ImageView, không có thay đổi tỷ lệ ảnh. | | centerCrop | ImageView.ScaleType.CENTER\_CROP: Đặt ảnh vào giữa ImageView, có thu phóng ảnh (nhưng giữ nguyên tỉ lệ cao / rộng) sao cho ảnh phủ kín hết cả ImageView (phần thừa bị cắt) | | centerInside | ImageView.ScaleType.CENTER\_INSIDE: Đặt ảnh vào giữa ImageView, có thu phóng ảnh (nhưng giữ nguyên tỉ lệ cao / rộng) sao cho toàn bộ các phần của ảnh hiện thị trên ImageView. | | fitCenter | ImageView.ScaleType.FIT\_CENTER: Đặt ảnh vào giữa ImageView, có thu phóng ảnh (nhưng giữ nguyên tỉ lệ cao / rộng) sao cho toàn bộ các phần của ảnh hiện thị trên ImageView. | | fitEndfitStart | ImageView.ScaleType.FIT\_END, ImageView.ScaleType.FIT\_START: Co ảnh vừa View, vị trí ảnh ở cuối (ở đầu) ImageView | | fitXY | ImageView.ScaleType.FIT\_XY: Co ảnh vừa khít cả chiều rộng và cao. | |
| 2 | **Tự động co giãn vừa với ảnh**  **android:adjustViewBounds** nếu nhận giá trị true thì các ImageView tự động co biên vừa với ảnh. (cần có thiết lập chiều rộng hoặc cao là **wrap\_content**) |

## 