1. 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.

1. Cấu hình AndroidManifest.xml

1. AndroidManifest là gì

**AndroidManifest** là file để bản kê khai trình bày những thông tin thiết yếu về ứng dụng của bạn với hệ thống Android, thông tin mà hệ thống phải có trước khi có thể chạy bất kỳ mã nào của ứng dụng.Hay nói cách khác đây là file dùng để config những thuộc tính cho ứng dụng của bạn mà khi ứng dụng khởi chạy hệ điều hành có thể hiểu được và xử lí.

**Những công việc mà tập tin manifest cần phải thực hiện :**

– Khai báo tên của package cho ứng dụng. Tên của package là định danh duy nhất của ứng dụng

– Mô tả các thành phần (component) của ứng dụng: activity, services, broadcast receivers, và content providers mà ứng dụng có. Khai báo tên các lớp mà cài đặt các thành phần này và các tác vụ mà các thành phần này có thể xử lý (ví dụ các thông điệp một Intent có thể xử lý). Những khai báo này cho phép hệ điều hành Android biết các thành phần này có thể có thể thực hiện được tác vụ gì và điều kiện để chúng có thể thực thi.

– Xác định tiến trình (process) sẽ chứa những thành phần (component) nào.

– Khai báo các quyền mà ứng dụng cần phải có để truy xuất các thành phần của API và giao tiếp với các ứng dụng khác.

– Khai báo các quyền mà yêu cầu ứng dụng khác cần phải có để có thể giao tiếp được với các component của ứng dụng này.

– Mô tả phiên bản API tối thiểu mà ứng dụng có thể chạy.

– Liệt kê các thư viện mà ứng dụng cần phải kết nối.

2. Cấu trúc của tệp tin Manifest

<?xml version="1.0" encoding="utf-8"?>

<manifest>

<uses-permission />

<permission />

<permission-tree />

<permission-group />

<instrumentation />

<uses-sdk />

<uses-configuration />

<uses-feature />

<supports-screens />

<compatible-screens />

<supports-gl-texture />

<application>

<activity>

<intent-filter>

<action />

<category />

<data />

</intent-filter>

<meta-data />

</activity>

<activity-alias>

<intent-filter> . . . </intent-filter>

<meta-data />

</activity-alias>

<service>

<intent-filter> . . . </intent-filter>

<meta-data/>

</service>

<receiver>

<intent-filter> . . . </intent-filter>

<meta-data />

</receiver>

<provider>

<grant-uri-permission />

<meta-data />

<path-permission />

</provider>

<uses-library />

</application>

</manifest>

**Thẻ manifest**

**<manifest> … </manifest>**: bao bọc file **Manifest** này chính là thẻ có tên **manifest**, thẻ này là thẻ gốc cho các thẻ con bên trong nó, nó chỉ xuất hiện 1 lần duy nhất và phải có chứ không sẽ lỗi ngay.

**– xmlns:android**: đây đơn giản là dòng khai báo **namespace android** mà thôi. Sau dòng khai báo này thì các thẻ khác bên trong thẻ manifest này mới có thể sử dụng được từ khóa android.

**– package:** chính là package name của project. Mặc định package này sẽ là giá trị giống như thuộc tính applicationId trong file build.gradle ở cấp độ module mà bạn đã làm quen ở bài trước.

Hầu như chúng ta sẽ chẳng cần đụng đến thẻ này, bạn chỉ cần biết thôi và để nó như mặc định.

**Thẻ uses-permission**

**<uses-permission />**: đây là cú pháp lệnh để bạn xin cấp một quyền gì đó khi ứng dụng bạn tương tác với dữ liệu nào đó mà google không cho phép bạn dùng tùy ý.Ví dụ như mình đã nói ở trên khi ứng dụng bạn muốn đọc danh bạ của một điện thoại nào đó thì bạn sẽ phải xin quyền để đọc chứ không thể tự ý đọc được, cú pháp xin quyền như sau:

<uses-permission android:name="android.permission.READ\_CONTACTS" />

<uses-permission android:name="android.permission.CALL\_EMERGENCY\_NUMBERS" />

<uses-permission android:name="android.permission.READ\_OWNER\_DATA" />

<uses-permission android:name="android.permission.SET\_WALLPAPER" />

<uses-permission android:name="android.permission.DEVICE\_POWER" />

<uses-permission android:name="android.permission.INTERNET" />

Sau khi khai báo xong bạn có thể tùy ý xem danh bạ của thiết bị đó (1), có rất nhiều quyền bạn muốn dùng thì phải khai báo giống như trên như: quyền kết nối internet, quyền đọc tin nhắn, quyền kiểm tra kết nối mạng..rất rất nhiều

**Thẻ permission**

**<permission />**: thẻ này tuy trông giống với uses-permission ở trên đây nhưng permission có công dụng hoàn toàn khác. Thẻ permission này có thể dùng để định nghĩa ra các quyền cho riêng ứng dụng của bạn. Vì chúng khá ít xài nên các thẻ permission, permission-tree, permission-group sẽ được mình nói đến cụ thể sau nhé.

**Thẻ supports-screens**

**<supports-screens />**: thẻ này cũng kha khá quan trọng, nó được dùng để định nghĩa các thể loại màn hình mà ứng dụng của bạn hỗ trợ.

**Thẻ application**

**<application>… …</application>**: thẻ này quan trọng lắm đây, thẻ này chứa một số thuộc tính liên quan đến cấu hình của ứng dụng (đúng như tên gọi của nó), mà mình sẽ liệt kê như bên dưới đây.

**– android:allowBackup** – Cho phép ứng dụng của bạn nằm trong chương trình tự động backup của hệ thống Android hay không. Thực sự mình cũng chưa kiểm chứng tính hiệu quả của chế độ backup này, nghe nói hệ thống sẽ giới hạn dữ liệu backup của ứng dụng với mỗi người dùng là **25MB**, kèm theo việc giới hạn các loại dữ liệu có thể backup. Dù sao thì tính năng này cũng rất tốt nên bạn cứ để giá trị của thuộc tính này là **true** như mặc định nhé.  
**– android:icon** – Thuộc tính này là nơi bạn thiết lập icon cho ứng dụng. Icon này sẽ xuất hiện trên màn hình chính của thiết bị. Hiện tại thì icon này đang được set mặc định là file **ic\_launcher.png** để trong thư mục **res/mipmap/**.   
**– android:label** – Thuộc tính này trỏ đến một giá trị **string** trong resource, hoàn toàn tương tự như cách bạn khai báo text cho **TextView**. String này chính là tên ứng dụng của chúng ta.  
**– android:supportsRtl** – Nếu bạn còn nhớ, từ ***RTL*** viết tắt của right-to-left, thuộc tính này cho phép ứng dụng của bạn có hỗ trợ cho các hệ ngôn ngữ viết từ phải-sang-trái hay không.   
**– android:theme** – Thuộc tính này khá hay, nó giúp chúng ta định nghĩa *“giao diện chủ đề”* cho ứng dụng. Và vì nó hay quá nên mình sẽ không nói ở đây, mình dành hẳn một bài để nói về chủ đề này để cho các bạn hiểu rõ nhất về nó.

Tất nhiên thẻ **application** còn nhiều thuộc tính lắm, bạn có thể vào [*link này*](https://developer.android.com/guide/topics/manifest/application-element.html) để xem thông tin đầy đủ về nó. Mình chỉ nói qua những thuộc tính dùng nhiều nhất, còn lại thì khi nào phát sinh thuộc tính mới mình sẽ nói thêm với các bạn.

Ngoài ra thì trong thẻ **application** này có các thẻ con quan trọng nữa, chúng ta lại tiếp tục nói đến các thẻ con của application ở bên dưới đây.

**Thẻ activity**

**<activity>… …</activity>**: thẻ này giúp bạn khai báo các **Activity** có trong ứng dụng. **Activity** là gì? Chúng được xem như các màn hình trong ứng dụng của chúng ta. Nếu ứng dụng của chúng ta sẽ xây dựng nhiều màn hình khác nhau, khi đó thẻ **activity** sẽ nhiều lên, vì mỗi thẻ **activity** giúp khai báo cho một màn hình duy nhất.

**Thẻ service**

**<service>…</service>**: Tương tự như thẻ **activity** trên đây. Mỗi thẻ **service** dùng để định nghĩa cho một loại **Service**. **Service** là một dịch vụ chạy ngầm trên ứng dụng của bạn kể cả khi ứng dụng đã tắt đi và khi sử dụng bạn phải khai  báo chúng trong AndroidMainifest.

**Thẻ receiver**

**<receiver>…</receiver>**: Cũng như hai thẻ **activity** và **service** trên đây. Mỗi thẻ **receiver** dùng để định nghĩa cho một loại **BroadCast Receiver**. **BroadCast Receiver** để bạn lắng nghe các sự kiện thay đổi của hệ thống như khi bất wifi, khi sạc pin…

**Thẻ provider**

**<provider>…</provider>**: Cũng như các thẻ **activity**, **service**, và **receiver** trên đây. Mỗi thẻ **provider** dùng để định nghĩa cho một loại **Content Provider**.

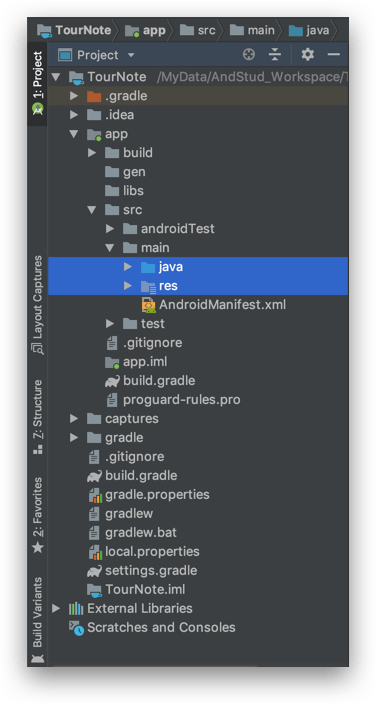
1. Android Resource

1. Resource là gì

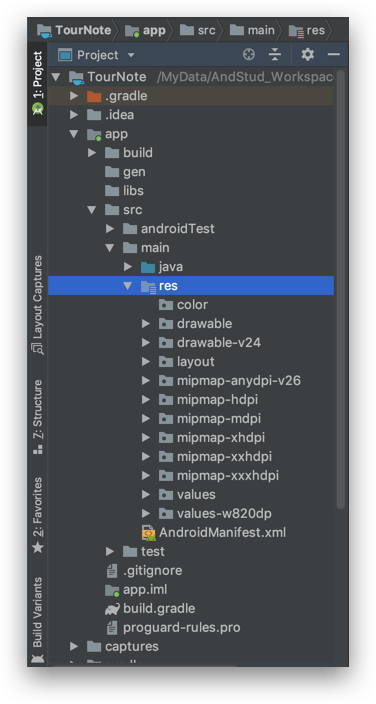
Trong lập trình Android, có một sự phân biệt rõ ràng giữa **resource** và **source code**. Nếu như source code là Java code, thì resource là những file còn lại không liên quan đến Java code đó, chẳng hạn như các file hình ảnh, âm thanh, video, các file text, màu sắc,…

2. Vị trí của resource trong Project

Nếu như *source code* được để trong thư mục***java/*** thì *resource* được để trong thư mục ***res/***. Bạn hãy nhìn vào hai thư mục mà mình vừa nhắc đến bên dưới đây.



Nếu bạn xổ thư mục ***res/*** này ra, bạn sẽ nhìn thấy rất nhiều thư mục con trong đó. Những thư mục con này được tạo sẵn khi bạn tạo mới một project, *số lượng thư mục tạo sẵn và tên các thư mục này phụ thuộc vào từng thời điểm của Android Platform*. Ngoài các thư mục được tạo sẵn này bạn có thể tạo thêm nhiều thư mục khác nữa (tất nhiên theo một quy luật đặt tên nhất định mà chúng ta sẽ nói rõ hơn ở các bài sau).



3. Cách sử dụng resource

**- Default Resource**

**Default resource** có thể dịch là *resource theo mặc định*. Các resource dạng này sẽ được hiển thị *tùy theo chỉ định mặc định của máy*.

Ví dụ như bạn để một [*ảnh Bitmap*](https://yellowcodebooks.com/2017/06/06/android-bai-20-su-dung-drawable-anh-bitmap/) vào trong *default resource*, thì ảnh của bạn sẽ được dùng cho tất cả các kích cỡ màn hình ở ngoài thị trường, nó có thể sẽ trông đẹp ở một số màn hình, nhưng có thể sẽ mờ, hay vỡ ở một số màn hình khác.

Thêm một ví dụ nữa, chẳng hạn như bạn đang xây dựng ứng dụng hỗ trợ song ngữ Anh-Việt, thì với file ngôn ngữ được để ở *default resource*, file này sẽ được hệ thống dùng đến mà không hề có sự chọn lựa rằng khi nào nên hiển thị giao diện ứng dụng với ngôn ngữ tiếng Anh, khi nào tiếng Việt.

Nói vậy không có nghĩa là default resource vô dụng, vậy khi nào thì dùng default resource được? Chúng ta đi tiếp phần dưới đây để cùng nhau sáng tỏ.

**- Cách sử dụng DefaultResource**

Nếu bạn có một ảnh, hay một file âm thanh, hay một text, mà bạn không cần phải phân biệt chúng giữa các thiết bị khác nhau trên thị trường, thì bạn có thể để các file này ở *default resource*.

Mình ví dụ như resource là một file âm thanh notification của app. Ồ file này thì dù cho máy có màn hình kiểu nào, LG hay Samsung, chạy ở quốc gia nào cũng được. Cho file này vào default resource thôi.

Hay bạn có một ảnh dạng Vector, ảnh dạng này sẽ không bị vỡ khi bạn zoom nó như là ảnh Bitmap, do đó nó không cần phân biệt theo các loại màn hình khác nhau. Cho file ảnh này vào default resource thôi.

Thêm một ví dụ nữa, đó là nếu bạn có các [*resource dạng màu sắc*](https://yellowcodebooks.com/2017/05/10/android-bai-17-su-dung-color/). Ồ màn hình nào cũng hiển thị màu giống nhau, không cần phân biệt. Cho resource này vào default resource thôi.

Nhiêu đây ví dụ chắc bạn cũng đã hiểu khi nào thì dùng resource dạng default resource. Vậy thì, cho resource vào default resource là thế nào? Dễ lắm, bạn chỉ cần để các resource mà bạn có vào các thư mục con của thư mục ***res/*** theo một quy tắc sau. Nên nhớ là các thư mục con này hoặc sẽ có sẵn khi bạn tạo mới project (như hình mình để ở trên kia), hoặc bạn tự tạo nhưng nhớ đúng với tên mình liệt kê ra đây (tất cả loại resource này sẽ được mình nói rõ ở từng bài học riêng biệt, bạn yên tâm nếu đọc sơ qua danh sách dưới đây mà không hiểu gì nhé).

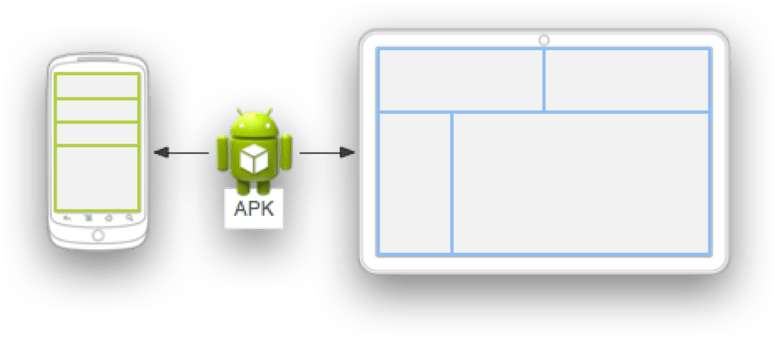
***– animator/***: Thư mục này dùng để chứa các animation theo dạng XML. Đây là các resource định nghĩa về chuyển động cho các thành phần UI của ứng dụng.  
***– anim/***: Thư mục này cũng dùng để chứa các animation theo dạng XML. Nhưng khác với animator, các resource theo dạng anim tạo chuyển động theo một số cấu trúc định sẵn, và vì vậy anim sẽ không linh hoạt bằng animator.  
***– color/***: Thư mục này dùng để chứa định nghĩa về các màu sắc được biến đổi theo trạng thái (state list of colors) cho ứng dụng theo dạng XML.  
***– drawable/***: Thư mục này dùng để chứa các ảnh bitmap (hỗ trợ các ảnh theo định dạng *.png*, *.9.png*, *.jpg*, *.gif*); Chứa các ảnh biến đổi theo trạng thái (state list); Chứa các hình khối (shape) do bạn tự vẽ bằng XML; Và còn nhiều loại khác nhưng ít phổ biến hơn các loại mình vừa nêu.  
***– mipmap/***: Thư mục này dùng để chứa các icon của ứng dụng.  
***– layout/***: Thư mục này dùng để chứa đựng tất cả các giao diện màn hình được code bằng XML. Bạn nhớ lại đi, bài trước chúng ta thử code giao diện ở file ***activity\_main.xml***, file lày có phải nằm trong thư mục ***layout/*** không nào?  
***– menu/***: Thư mục này dùng để chứa đựng tất cả các định nghĩa cho menu của ứng dụng, bao gồm *Options Menu*, *Context Menu*, hay *Sub Menu*.  
***– raw/***: Thư mục dùng để chứa đựng các file mà bạn được phép định nghĩa nội dung tùy thích, không bị gò bó phải là XML hay Java code gì cả.  
***– values/***: Thư mục này dùng để chứa đựng các file XML với các kiểu dữ liệu đơn giản, như *string*, *integer*, *color*. Chúng ta sẽ nói rõ hơn chút xíu về thư mục ***values/*** này ở phần tiếp theo của bài học hôm nay.  
***– xml/***: Thư mục này dùng để chứa đựng các file XML tùy ý, tuy tùy ý nhưng phải theo cấu trúc XML.

Vậy thôi đấy, có nhiều lắm không bạn. Bạn yên tâm chúng ta sẽ nói rất cụ thể về từng loại resource trong từng thư mục default resource này sau.

Và có một điều nên nhớ rằng, trong Android thư mục ***res/*** *chỉ chứa duy nhất một cấp thư mục con thôi*. Do đó bạn không thể tạo một thư mục con như này được ***res/values/my\_res/.***

**- Alternative resource**

**Alternative resource** có thể dịch là *resource có chọn lựa*. Ngược với default resource là các resource mà bạn không cần quan tâm chúng hiển thị thế nào cho từng loại thiết bị. Thì alternative resource là các resource mà bạn chính là người phải chỉ định cho hệ thống biết khi nào nên dùng resource nào, tức là có sự chọn lựa resource theo ý bạn. Bạn hãy nhìn vào hình minh họa sau.



Hình minh họa trên có nghĩa là, giả sử bạn có một layout theo chuẩn *alternative resource*, khi đó bạn có thể chỉ định rõ cho hệ thống rằng nếu ứng dụng chạy trên phone (như hình bên trái) thì hãy hiển thị layout đó dạng list, còn nếu ứng dụng chạy trên tablet (hình bên phải) thì hãy hiển thị layout dạng table.

Vậy cách sử dụng alternative resource có khác với default resource hay không? Mời bạn xem tiếp bước sau.

**- Cách sử dụng Alternative resource**

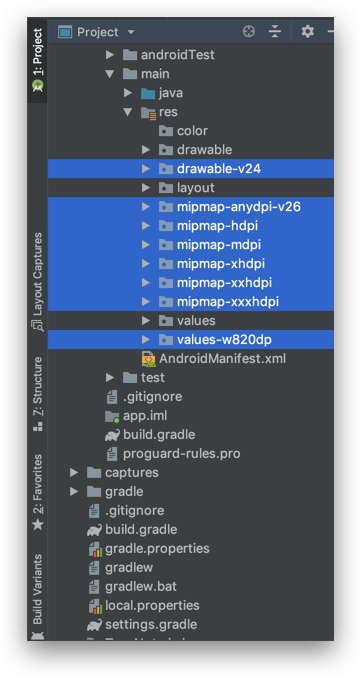
Cách sử dụng alternative resource đơn giản không kém cách sử dụng default resource, mình chắc chắn đó. Bởi vì bạn vẫn dựa vào cách thức tổ chức thư mục theo dạng default resource, rồi bạn chỉ cần thêm một điều kiện vào cuối tên của mỗi thư mục đó để hệ thống biết khi nào nên load resource nào mà thôi.

Công thức cụ thể cho alternative resource như sau.

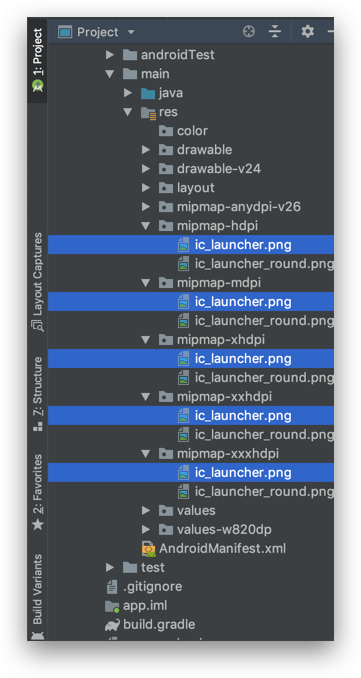
**<default\_resource>-<config\_qualifier>**

Trong đó ***default\_resource*** là các thư mục con của ***res/*** mà mình có nhắc đến ở trên như ***drawable/***, ***layout/***,… Còn ***config\_qualifier*** chính là phần chỉ định cho hệ thống biết khi nào nên sử dụng resource nào. Hai thành phần này được ngăn cách nhau bởi dấu ***“-“***.

Nhìn vào hình dưới đây, bạn sẽ thấy các thư mục được tô sáng chính là các thư mục alternative resource, vì chúng có chỉ định ***config\_qualifier***. Để biết rõ về các chỉ định ***config\_qualifier*** này thì mời bạn đọc [*bài viết này*](http://yellowcodebooks.com/2016/10/20/android-tat-tan-tat-ve-alternative-resource/) để hiểu rõ hơn nhé..



Bạn hãy nhìn vào ví dụ các file ảnh ***ic\_launcher.png***được để vào trong các alternative resource như hình sau.



Khi đó giả sử ở một thiết bị nào đó, khi hệ thống muốn load ảnh ***ic\_launcher.png*** này lên dùng, thì việc đầu tiên hệ thống sẽ phải xác định xem ***ic\_launcher.png*** ở thư mục nào là thích hợp nhất cho thiết bị đó. Nếu hệ thống tìm thấy một thư mục thích hợp, ví dụ ***mipmap-xxxhdpi/*** thì ***ic\_launcher.png*** trong thư mục đó sẽ được load lên dùng. Còn nếu hệ thống không tìm thấy một thư mục thích hợp, nó sẽ tìm đến thư mục default resource, tức là tìm đến thư mục với tên ***mipmap/***. Chính vì vậy bạn thường thấy có cả các default resource lẫn alternative resource được định nghĩa đồng thời trong cùng một ứng dụng.

1. Các Resource chuyên sâu Android – Style

1. Style là gì

***Style – Phong cách***. Như mình có trình bày ở trên, Phong Cách ở đây là muốn nói về cách mà bạn tạo ra một sự đồng nhất về giao diện cho ứng dụng. Như ví dụ về ***Button*** trên đây, bạn muốn tạo ra các ***Button*** giống nhau ở các màn hình, để cho nó có sự đồng nhất. Khi đó việc thiết kế cho ***Button*** sao cho chúng giống nhau như vậy khiến bạn cảm thấy nhàm chán vì mất khá nhiều thời gian. Lúc đó bạn sẽ cần đến một nơi định nghĩa hết tất tần tật những thuộc tính của ***Button*** trên, rồi đến với từng ***Button*** bạn chỉ việc mang định nghĩa đó ra dùng. Thì khi đó Style chính là kiểu resource mà bạn đang tìm

Đã nói về Style thì mình cũng muốn các bạn làm quen với khái niệm ***Theme*** luôn. Vì về cơ bản, trong Android *Theme không khác gì Style*. Chúng đều giúp tạo ra sự đồng nhất về giao diện. Nhưng thay vì Style giúp áp dụng sự đồng nhất cho các view sử dụng nó, thì Theme lại giúp áp dụng sự đồng nhất cho toàn bộ Activity (toàn bộ màn hình) hoặc toàn bộ ứng dụng. Ví dụ như bạn khai báo font cho Theme, rồi áp dụng Theme đó cho toàn bộ ứng dụng, thì tất cả các view trong màn hình đó đều được thay đổi font.

Nếu đến đây bạn vẫn còn mơi hồ về hai khái niệm Style và Theme, thì bạn cứ tưởng tượng chúng giống như khái niệm CSS (Cascading Style Sheets) trong lập trình Web vậy. CSS cũng giúp định hình các phong cách, để tạo nên sự đồng nhất cho các thẻ HTML bên trong trang Web đó.

Có một điều bạn cần phải nhớ là mặc dù Theme là một khái niệm để mang ra nói chung với Style, nhưng bạn không cần phải định nghĩa ra bất kỳ Theme nào cả. Bạn chỉ cần chọn một trong các Theme được xây dựng sẵn bởi hệ thống. Và vì kiến thức về Theme cũng kha khá nhiều và quan trọng, nên mình sẽ nói tiếp ở bài sau nhé, bài này sẽ tập trung cho Style.

Giả sử mình có một TextView như sau.

<?xml version="1.0" encoding="utf-8"?>

<manifest>

<uses-permission />

<permission />

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:textColor="#00FF00"

android:typeface="monospace"

android:text="@string/hello" />

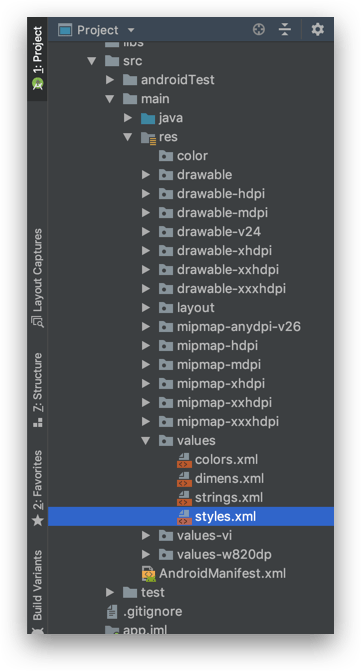
Mình muốn ***TextView*** này là một ***TextView*** chuẩn, để ở các màn hình khác mình cũng sẽ định nghĩa các ***TextView*** giống vậy. Và thay vì mình phải khai báo lại các thuộc tính như trên cho tất cả các ***TextView*** ở các màn hình khác, thì mình chỉ cần tạo ra một Style có tên là ***CodeFont***, rồi sau đó áp dụng Style này cho ***TextView*** này và các ***TextView*** về sau như sau.

|  |  |
| --- | --- |
| 1  2  3 | <TextView      style="@style/CodeFont"      android:text="@string/hello" /> |

Vậy đó. Cách để áp dụng Style cho ***TextView*** đơn giản vậy thôi. Cách thức khởi tạo chi tiết hơn một Style được mô tả tiếp ở bước kế tiếp.

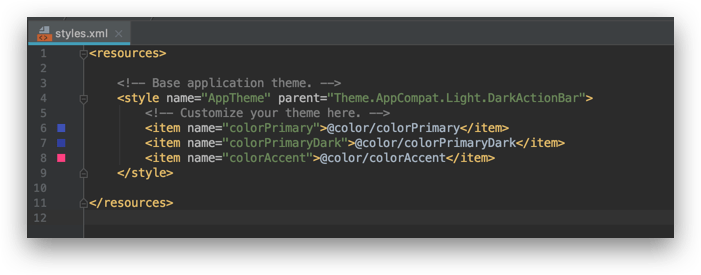
2. Cấu trúc file styles.xml

Như mình có nói thì Style là một dạng resource của Android, vậy chắc chắn file Style phải ở đâu đó trong thư mục ***res/*** rồi, và chính xác của thư mục chứa file Style này là ***res/values/***. Bạn hãy mở thư mục này theo đường dẫn như hình bên dưới. Khi bạn tạo mới một project thì hệ thống cũng tạo sẵn cho chúng ta một file Style và để ở thư mục này, tên của file Style này là ***styles.xml***.

[](https://i1.wp.com/yellowcodebooks.com/wp-content/uploads/2017/01/Hinh1-1.png?ssl=1)

Và dĩ nhiên vì là một loại resource của Android, nên file Style cũng theo quy luật [*Alternative Resource và Default Resource*](http://yellowcodebooks.com/2016/10/05/android-bai-8-tim-hieu-android-resource/). File ***styles.xml*** các bạn đang thấy trên hình thuộc về Default Resource, nhưng có thể project ở máy của bạn có chứa đựng một file ***styles.xml*** khác ở thư mục ***values-xxx/*** nào khác thì có nghĩa file Style đó thuộc về Alternative Resource.

Giờ thì chúng ta cùng mở file ***styles.xml*** ở thư mục Default Resource này lên nhé.

[](https://i0.wp.com/yellowcodebooks.com/wp-content/uploads/2017/01/Hinh2-2.png?ssl=1)

Bạn có thể thấy rằng file ***styles.xml*** cũng giống như file ***strings.xml*** mà bạn đã làm quen hay tất cả các file **.xml** khác trong thư mục ***values/*** mà bạn sẽ làm quen sau này cũng đều được để trong một thẻ gốc có tên ***resources***.

Hình trên đây đang có một Style, mỗi Style mà bạn tạo ra sẽ được nằm trong cặp đóng mở thẻ có tên ***style***. Và thẻ ***style*** duy nhất bạn thấy trong hình lại dùng như một Theme của ứng dụng, mà chúng ta sẽ nói về Theme này ở bài sau.

Còn bây giờ giả sử chúng ta cần tạo một Style có tên là ***CodeFont*** để dùng cho ví dụ về ***TextView*** ở trên, vậy mình sẽ thêm các dòng sau vào file ***styles.xml***.

<style name="CodeFont" parent="@android:style/TextAppearance.Medium">

<item name="android:layout\_width">match\_parent</item>

<item name="android:layout\_height">wrap\_content</item>

<item name="android:textColor">#00FF00</item>

<item name="android:typeface">monospace</item>

</style>

Khi đó ở mọi nơi mà mình muốn dùng đến Style này, mình sẽ gọi đến bằng thuộc tính ***style***như sau.

<TextView

style="@style/CodeFont"

android:text="@string/hello" />

Giờ thì chúng ta hãy làm quen với những gì có thể có bên trong thẻ style nhé, rồi sau đó cùng nhau thực hành tạo style cho TourNote ở bên dưới bài học này.

**Thuộc tính name**

Đây là tên của Style, như bạn thấy ở ví dụ trên tên này là ***CodeFont***, bạn phải đặt tên cho bất kỳ Style nào mà bạn đặt ra để có thể dùng đến sau này. Cách đặt tên cho Style cũng như cách đặt tên cho resource dạng String mà bạn đã thực hành ở [*Bài 8*](http://yellowcodebooks.com/2016/10/05/android-bai-8-tim-hieu-android-resource/).

**Các thẻ item**

Chà, như Style ví dụ trên đây thì sau thuộc tính ***name*** còn có thuộc tính ***parent***. Nhưng mình xin phép được nói đến các thẻ ***item*** trước vì nó quan trọng hơn.

Mỗi thẻ ***item*** như vậy định nghĩa ra một loại phong cách nào đó, trong đó tên của loại phong cách đó được miêu tả trong thuộc tính ***name*** của thẻ này, như bạn đã thấy trong ví dụ, các tên này như là “android:layout\_width”, “android:layout\_height”, “android:textColor”, “android:typeface” và còn nhiều tên nữa… Bạn có thể thấy các giá trị cho thuộc tính ***name*** này đều là các thuộc tính của các widget hay của layout chứa widget đó mà bạn đã làm quen trong việc thiết kế giao diện trước đây. Như vậy Style cũng chính là nơi bạn lấy các thuộc tính có sẵn ra, rồi định nghĩa cho chúng một giá trị nào đó, rồi áp dụng lại cho các widget, đó chính là khái niệm Phong Cách trong bài học hôm nay.

Theo sau các khai báo ***name*** trong thẻ này là các giá trị của nó, đó có thể là một kiểu String, một giá trị màu sắc, hay một độ lớn ***dp***, ***sp***,… tùy thuộc vào từng loại phong cách.

**Thuộc tính parent**

Thuộc tính này không bắt buộc. Nhưng nếu bạn muốn Style của mình được kế thừa từ một Style có sẵn (do bạn tạo ra trước đó, hay từ Style của một thư viện nào đó, hoặc Style của hệ thống), việc kế thừa này giúp bạn tận dụng lại những định nghĩa từ Style gốc, và tạo ra các định nghĩa mới bổ sung cho Style gốc còn thiếu.

Như ví dụ từ định nghĩa trên đây, bạn sẽ thấy Style dành cho ***TextView*** có kế thừa từ một Style của ***TextView*** có sẵn của hệ thống, đó là ***@android:style/TextAppearance.Medium***. Ngoài việc dùng lại tất cả những thuộc tính được khai báo sẵn của hệ thống, thì Style ***CodeFont*** trên lại định nghĩa mới các thuộc tính về **layout\_width**, **layout\_height**, **textColor**, và **typeface** cho riêng mình.

Còn nếu bạn muốn biết nhiều hơn về ví dụ cách kế thừa từ một Style do bạn tạo ra, thì mời bạn xem đoạn định nghĩa các Style bên dưới.

<style name="BaseTextViewStyle">

<item name="android:layout\_width">wrap\_content</item>

<item name="android:layout\_height">wrap\_content</item>

<item name="android:textColor">@color/cl\_default</item>

<item name="android:textSize">@dimen/text\_size\_normal</item>

</style>

<style name="LargeTextViewStyle" parent="BaseTextStyle">

<item name="android:textSize">@dimen/text\_size\_large</item>

</style>

<style name="SmallTextViewStyle" parent="BaseTextStyle">

<item name="android:textSize">@dimen/text\_size\_small</item>

</style>

Với ví dụ này thì mình định nghĩa ra một Style chung cho nhiều ***TextView*** có tên là ***BaseTextViewStyle***. Sau đó mình có thêm hai Style nữa là ***LargeTextViewStyle*** và ***SmallTextViewStyle*** đều có thuộc tính ***parent*** chỉ đến ***BaseTextViewStyle***, nhưng hai Style con lại có cách định nghĩa riêng giá trị **textSize** cho chúng.

Lưu ý là với việc kế thừa từ Style của bạn ở cùng một file, thì bạn có thể không cần đến khai báo ***parent*** nữa mà dùng dấu chấm như thế này cũng được.

<style name="BaseTextStyle.LargeTextViewStyle">

<item name="android:textSize">@dimen/text\_size\_large</item>

</style>

<style name="BaseTextStyle.SmallTextViewStyle">

<item name="android:textSize">@dimen/text\_size\_small</item>

</style>

1. 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:



Sơ đồ này là mô phỏng vòng đời Activity. Mỗi Activity trong một ứng dụng sẽ có vòng đời khác nhau nhưng chúng đều tuân theo chuẩn của sơ đồ này.

Các phương thức được mô tả trong sơ đồ này bao gồm ***onCreate(), onStart(), onRestart(), onResume(), onPause(), onStop() và onDestroy()***. Các phương thức này được gọi là ***các lời gọi về (callback).*** Các ***callback*** chính là các phương thức được bạn ***override*** lại từ các phương thức sẵn có của lớp cha, để khi mà hệ thống cần, hệ thống sẽ gọi đến các phương thức đó trong ứng dụng của bạn.

**- Mô tả sơ đồ :**

Sơ đồ bắt đầu từ khi ***Activity launched***, tức là khi Activity được kích hoạt, và được hệ thống để vào [BackStack](https://yellowcodebooks.com/2017/09/16/android-bai-27-tim-hieu-back-stack/). Sau khi kích hoạt, lần lượt các callback ***onCreate()***, ***onStart()***, ***onResume()*** sẽ được hệ thống gọi đến.

Sau khi gọi đến các callback trên, thì Activity mới chính thức được xem là đang chạy (***Activity running***).

Lúc này, nếu có bất kỳ Activity nào khác chiếm quyền hiển thị, thì Activity hiện tại sẽ rơi vào trạng thái ***onPause()***. Nếu cái sự hiển thị của Activity khác làm cho Activity mà chúng ta đang nói đến không còn nhìn thấy nữa thì ***onStop()*** sẽ được gọi tiếp theo nữa. Lát nữa khi đi vào các chi tiết dưới đây bạn sẽ hiểu sự khác nhau giữa hai callback này thôi.

Nếu Acvitity đã vào ***onPause()*** rồi, tức là đang bị Activity khác đè lên, mà người dùng sau đó quay về lại Activity cũ, thì ***onResume()*** được gọi. Còn nếu Activity đã vào ***onStop()*** rồi, mà người dùng quay về lại Activity cũ thì ***onRestart()*** được gọi.

Trong cả hai trường hợp Activity rơi vào ***onPause()*** hoặc ***onStop()***, nó ***sẽ rất dễ bị hệ thống thu hồi (tức là hủy luôn ) để giải phóng tài nguyên***, khi này nếu quay lại Activity cũ, ***onCreate()*** sẽ được gọi chứ không phải ***onResume()*** hay ***onRestart()*** đâu nhé.

Và cuối cùng, nếu một Activity bị hủy một cách có chủ đích, chẳng hạn như người dùng nhấn nút Back ở System Bar, hay hàm ***finish()*** được gọi,… thì ***onDestroy()*** sẽ được kích hoạt và Activity kết thúc vòng đời của nó.

**- Các trạng thái trong vòng đời của 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/#Xay_Dung_Chuc_Nang_Back_Tren_ActionBar_Cua_ContactActivity) 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.

**- Làm quen với các Callback**

* **onCreate(Bundle savedInstanceState)**:

***Callback*** này được gọi khi Activity được kích hoạt, thậm chí người dùng chưa thấy gì của Activity thì ***callback*** này đã được gọi rồi. ***Callback*** này chỉ được gọi một lần duy nhất khi ***Activity*** được khởi tạo. Sau ***callback*** này bao giờ cũng gọi ngay lập tức ***onStart().***

Ở phương thức này chúng ta thường sử dùng để ***setContentView()***, khởi tạo Adapter cho ***ListView***, ***RecyclerView***…

* **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 chúng ta chưa nhìn thấy được ***Activity*** và tương tác được với ***Activity***

* **onResume()**:

***Callback*** này được gọi ngay khi Activity bắt đầu có thể tương tác với người dùng, tức là sau khi ***onStart()*** được gọi, 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, thường để lưu lại trạng thái của các biến trước khi hệ thống 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 Activity 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)

# View, ViewGroup, Layout trong Android

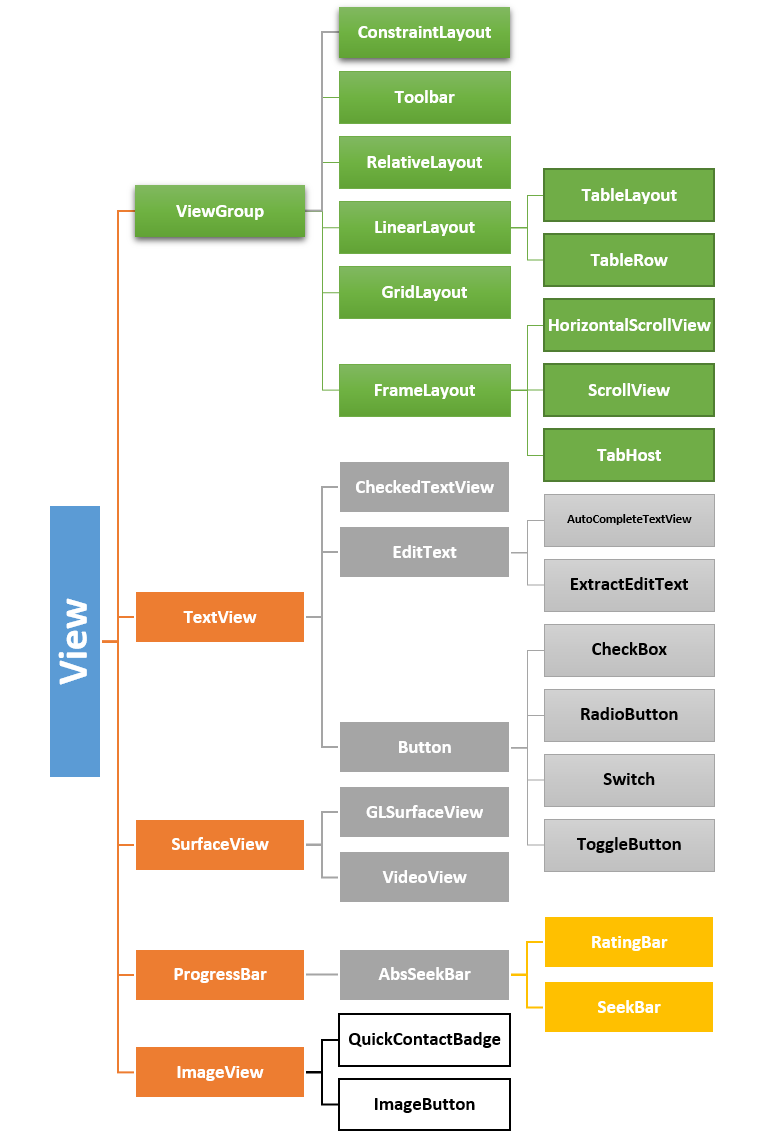
Các thành phần cơ bản để xây dựng giao diện người dùng (UI) trong Android là View, ViewGroup và Layout. Từ các thành phần cơ sở đó kết hợp chúng với nhau tạo ra các loại UI phức tạp cho ứng dụng Android

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 |
|  |  |

1. LinearLayout trong Android

1.LinearLayout trong Android

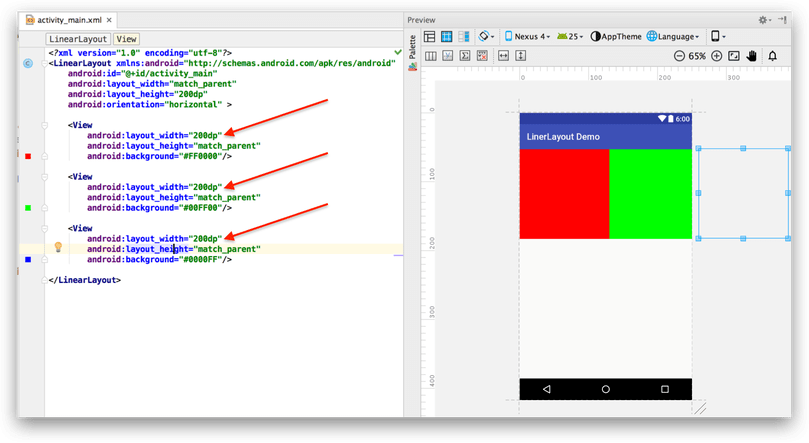
LinearLayout trong Android là một layout mà nó sắp xếp các view con một cách liên tục theo một hàng (**vertical**) hay một cột (**horizontal**), có một giá trị weight có thể gán vào mỗi View con để cho biết View con đó chiếm bao nhiêu không gian trong một tỷ lệ tương quan với các View con khác.

**Nguyên tắc của LinearLayout:**

Giả sử bạn có một ***LinearLayout*** mà bên trong nó có 3 view con (view ở đây nói chung là View hoặc ViewGroup) được xếp theo nguyên tắc ***horizontal***, tức xếp lớp theo chiều ngang. Giả sử có một view con có thuộc tính ***android:layout\_width=”match\_parent”***, tức là view con này có chiều rộng hết ***LinearLayout*** cha luôn rồi, thì các view còn lại sẽ hiển thị như thế nào?

Câu trả lời là ***LinearLayout*** sẽ dành không gian cho con kiểu “ưu tiên con cả trước”, tức là anh nào được đưa vào trước sẽ được ưu tiên không gian trước, anh nào vào sau sẽ… ra khỏi màn hình mà nằm nếu không đủ chỗ, tức là người dùng sẽ không thấy anh này đâu cả.

Bạn thấy như hình ví dụ sau, mình cố tình thiết lập cho cả 3 view con đều dành không gian rộng ***200dp***, như vậy ắt sẽ có view phải ra khỏi màn hình, đó là view có background màu xanh dương.



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.

Đối với ***LinearLayout***, Google đã đưa ra khái niệm ***weight*** trong việc phân bổ không gian cho view con.

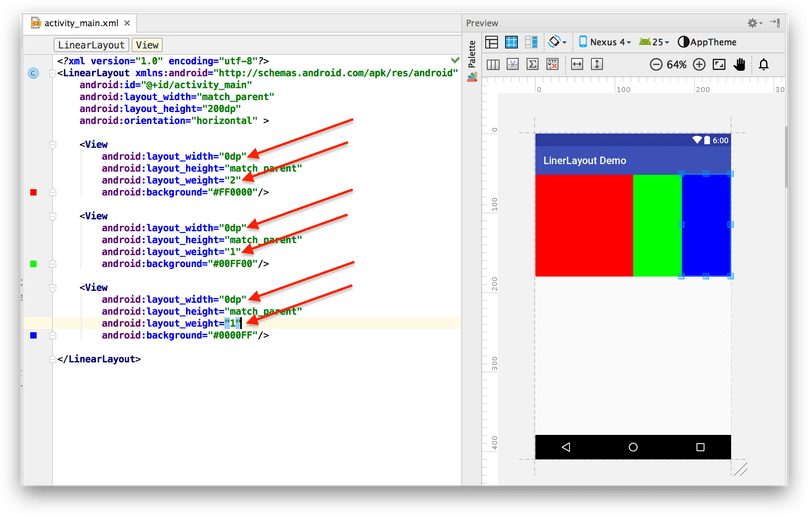
Nói một cách đầy đủ, ***weight*** sẽ giúp bạn thiết lập trọng số cho các view con để sử dụng phần không gian trống còn lại của một ***LinearLayout***, với mỗi trọng số khác ***0***, phần không gian trống còn lại của ***LinearLayout*** sẽ được chia ra cho các view con theo trọng số đó. Để sử dụng trọng số ***weight***, bạn không cần dùng đến ***layout\_width*** hay **layout\_height** cho view con tương ứng nữa.

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*)

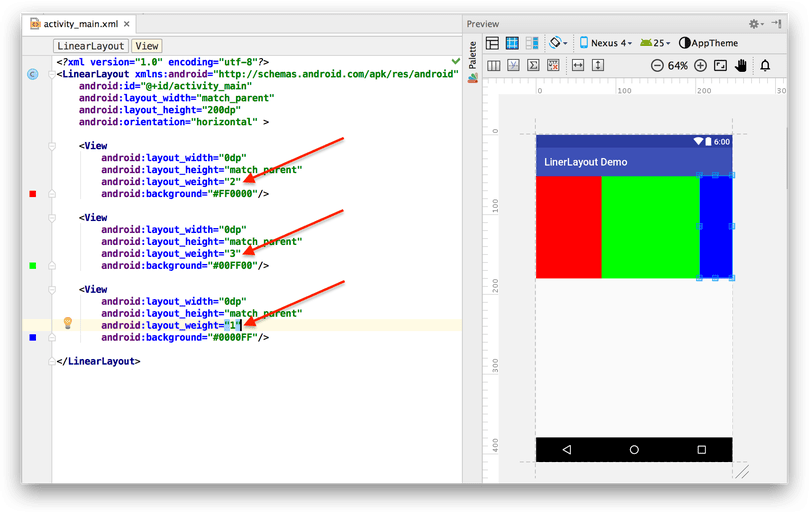
Nói một cách đầy đủ, ***weight*** sẽ giúp bạn thiết lập trọng số cho các view con để sử dụng phần không gian trống còn lại của một ***LinearLayout***, với mỗi trọng số khác ***0***, phần không gian trống còn lại của ***LinearLayout*** sẽ được chia ra cho các view con theo trọng số đó. Để sử dụng trọng số ***weight***, bạn không cần dùng đến ***layout\_width*** hay **layout\_height** cho view con tương ứng nữa, chúng ta xem qua cách sử dụng ***weight*** như sau trước khi xem ví dụ bên dưới.

– Với ***LinearLayout*** có **orientation** là ***horizontal***, bạn nên set ***android:layout\_width*** của view con về ***0dp***. Với ***LinearLayout*** có **orientation** là ***vertical***, bạn set ***android:layout\_height*** của view con về ***0dp***.  
– Khai báo ***android:layout\_weight*** cho view con đó và cho nó một con số (số nguyên hay số thực đều được, và không kèm theo ***dp*** hay ***dip*** vào giá trị này).

Bạn có thể xem ví dụ ở hình dưới, mình đã sửa lại bằng cách khai báo ***weight*** cho các view con lần lượt là ***2-1-1***. Khi đó view đầu tiên sẽ có chiều rộng sao cho bằng 2 lần các view còn lại.

[](https://i0.wp.com/yellowcodebooks.com/wp-content/uploads/2016/11/screen-shot-2016-11-14-at-14-57-18.png?ssl=1)

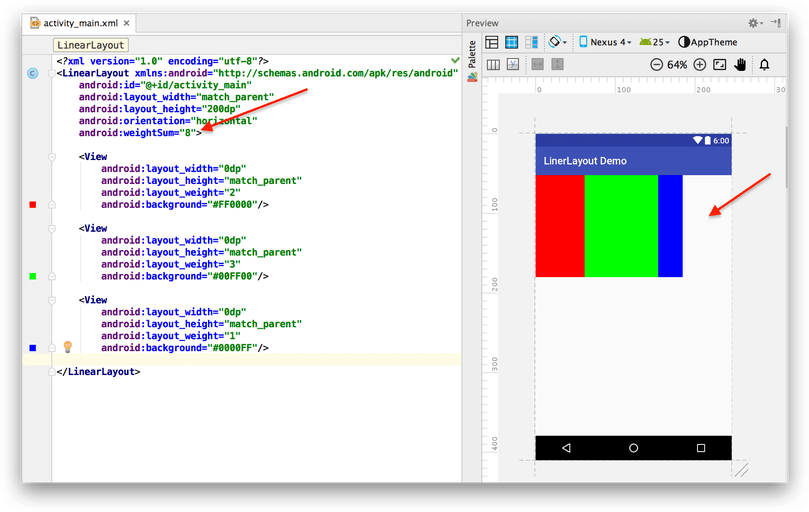
Để bạn dễ hiểu hơn, mình tiếp tục thử nghiệm bằng cách đổi lại ***weight*** cho chúng lần lượt là ***2-3-1***. Bạn sẽ thấy view màu đỏ với ***weight*** là ***2*** sẽ chiếm không gian rộng gấp 2 lần so với view xanh dương với ***weight*** là ***1***. Tương tự, view màu xanh lá với ***weight*** là ***3*** sẽ rộng gấp 3 lần so với view xanh dương với ***weight*** là ***1***. Bạn hiểu độ tương quan của các trọng số ***weight*** chưa nào?

[](https://i2.wp.com/yellowcodebooks.com/wp-content/uploads/2016/11/screen-shot-2016-11-14-at-16-11-25.png?ssl=1)

**android:weightSum :**

Với cách set trọng số ***weight*** như trên thì dù cho ***LinearLayout*** có không gian như thế nào thì view con cũng sẽ tìm cách lấp đầy không gian trống đó. Nhưng nếu bạn muốn chừa không gian trống cho mục đích nào đó, bạn có thể thiết lập một giá trị ***android:weightSum*** cho ***LinearLayout*** rồi set ***weight*** cho các view con của nó sao cho tổng giá trị ***weight*** nhỏ hơn ***weightSum*** như hình bên dưới đây.

Như hình thì bạn thấy mình set giá trị ***weightSum*** cho ***LinearLayout*** là ***8***, mà tổng ***weight*** của các view con chỉ là ***6*** thôi, như vậy sẽ còn dư 2 ***weight*** không có con nào được fill vào, nên nó mới trống như vậy.

[](https://i0.wp.com/yellowcodebooks.com/wp-content/uploads/2016/11/screen-shot-2016-11-14-at-16-24-45.png?ssl=1)

# 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 |
| android:layout\_alignBaseline | Căn thẳng đường baseline với View có ID được chỉ định |

1. 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 |

# ConstraintLayout trong Android

## 1. ConstraintLayout trong Android

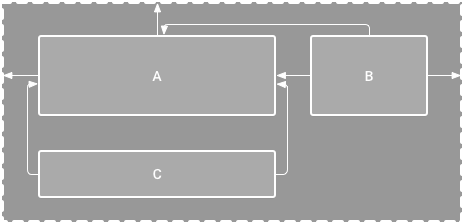
Dùng ConstraintLayout để xây dựng các layout trong lập trình Android, với tính năng ràng buộc các phần tử, thiết lập các xích đa dạng

**ConstraintLayout** là một layout mạnh, khuyến khích bạn dùng nếu có thể vì nó giúp tạo ra các giao diện phức tạp, mềm dẻo (hạn chế tối đa sử dụng các layout lồng nhau). Nó giúp định vị, sắp xếp các View con dựa trên sự ràng buộc liên hệ của các View con với View cha và sự liên hệ ràng buộc giữa các View con với nhau, với cơ chế tạo xích các View, gán trọng số hay sử dụng trợ giúp giao diện với Guideline.

ConstraintLayout thuộc Libaray Support nên để tích hợp vào dự án hãy thêm vào Gradle phiên bản muốn dùng, ví dụ:

implementation 'com.android.support:appcompat-v7:27.1.0'

implementation 'com.android.support.constraint:constraint-layout:1.0.2'



## 2. Sự ràng buộc

Mỗi view trong **ConstraintLayout** để định vị được chính xác cần tối thiếu 2 ràng buộc, một theo phương ngang (X) và một theo phương đứng (Y)

Khái niệm ràng buộc giữa các phần tử ở đây ám chỉ sự liên kết với nhau của các phần tử với nhau (kể cả với phần tử cha ConstraintLayout), sự căn chỉnh phần tử theo phần tử khác, hoặc với những đường thẳng ẩn thêm vào. Mỗi ràng buộc của phần tử View sẽ hoặc hưởng đến vị trí của nó theo trục X hoặc trục Y. Các View không có ràng buộc sẽ định vị ở góc trái - trên (tọa độ 0,0).

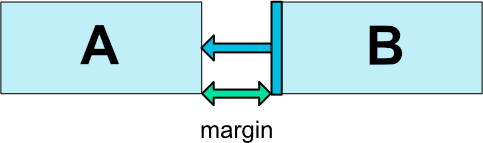
Trước tiên tham khảo bảng các thuộc tính về ràng buộc *layout\_constraint ...*, các thuộc tính ràng buộc sử dụng với namespace:app, giá trị nó gán vào là một ID của phần tử khác để kết nối ràng buộc hoặc là phần tử ch bằng hằng số "parent", ví dụ:

**app:layout\_constraintBottom\_toBottomOf="parent"**

|  |  |
| --- | --- |
| **Ràng buộc** | **Ý nghĩa ràng buộc** |
| layout\_constraintLeft\_toLeftOf | Rang buộc cạnh trái của phần tử tới phần tử chỉ ra trong giá trị (gán ID) |
| layout\_constraintLeft\_toRightOf | Bên trái với bên phải của phần tử chỉ ra |
| layout\_constraintRight\_toLeftOf | Bên phải với bên trái |
| layout\_constraintRight\_toRightOf | Phải với phải |
| layout\_constraintTop\_toTopOf | Cạnh trên với cạnh trên |
| layout\_constraintTop\_toBottomOf | Cạnh trên nối với cạnh dưới |
| layout\_constraintBottom\_toTopOf | Dưới với trên |
| layout\_constraintBottom\_toBottomOf | Dưới với dưới |
| layout\_constraintBaseline\_toBaselineOf | Trùng Baseline |
| layout\_constraintStart\_toEndOf | Bắt đầu - Kết húc |
| layout\_constraintStart\_toStartOf | Bắt đầu - Bắt đầu |
| layout\_constraintEnd\_toStartOf | Cuối với bắt đầu |
| layout\_constraintEnd\_toEndOf | Cuối với cuối |

## 3. Thuộc tính Margin trong các phần tử con

Cạnh nào của View con có ràng buộc thì có thể thiết lập thêm thuộc tính Margin để điều chỉnh thêm khoảng cách các cạnh tới điểm nối ràng buộc.



Các thuộc tính margin theo các cạnh: *android:layout\_marginStart, android:layout\_marginEnd, android:layout\_marginLeft, android:layout\_marginTop, android:layout\_marginRight, android:layout\_marginBottom*

Tương tự có *layout\_goneMarginStart, layout\_goneMarginEnd, layout\_goneMarginLeft, layout\_goneMarginTop, layout\_goneMarginRight, layout\_goneMarginBottom* có hiệu lực khi đối tượng ràng buộc đến gone

## 4. Phần tử Guideline

Ta có thể một đường kẻ ẩn trong ConstraintLayout nằm ngang hoặc đứng nó như là một View con để các View khác ràng buộc đến nếu muốn. Thêm vào bằng cách:

<android.support.constraint.Guideline

android:id="@+id/guideline\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical"

app:layout\_constraintGuide\_percent="0.3" />

Thiết lập đó là được kẻ ngang bằng thuộc tính: android:orientation="horizontal"đường kể đứng android:orientation="vertical"

**Vị trí phân chia ContraintLayout của GuideLine**

**1** Vị trị của Guideline có thể thiết lập nó cách cạnh trái (hoặc trên nếu là Guideline ngang) bằng thuộc tính app:layout\_constraintGuide\_percent giá trị là tỷ lệ phần trăm như 0.5 (50%), 0.2 (20%) độ rộng (cao) của ConstraintLayout. Ví dụ: app:layout\_constraintGuide\_percent="0.1"

**2** Nếu không muốn sử dụng layout\_constraintGuide\_percent thì có thể sử dụng cách thiết lập Guideline cách cạnh bắt đầu (trên với Guidleline ngang) bao nhiêu bằng thuộc tính: app:layout\_constraintGuide\_begin ví dụ:

Hoặc thiết lập cách cạnh cuối bao nhiêu với layout\_constraintGuide\_end

## 5. Bias

Khi hai cạnh đối diện nhau của View có ràng buộc, thì hai dàng buộc này ứng xử như một liên kết lò xo mặc định nó có độ mềm (độ cứng) bằng nhau dẫn đến View sẽ nằm giữa 2 điểm neo của ràng buộc. Nếu muốn điều chỉnh độ cứng này thì sử dụng thuộc tính:

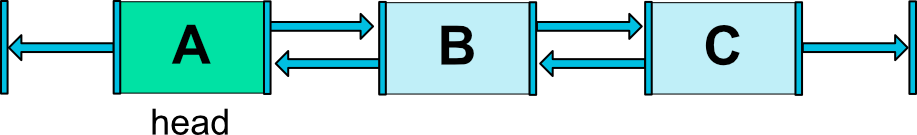
* app:layout\_constraintVertical\_bias thiết lập độ mềm của ràng buộc đầu (ngang). Với tổng độ mềm là 1 thì khi app:layout\_constraintVertical\_bias="0.1" thì độ mềm ràng buộc thứ hai sẽ là 0.9
* layout\_constraintHorizontal\_bias để thiết lập độ mềm hai ràng buộc theo phương đứng

## 6.Tỷ lệ các cạnh của View

Khi View con có thiết lập tối thiểu một kích thước là "0dp" thì kích thước đó có thể tự động điều chỉnh bằng cách lấy theo tỷ lệ với cạnh còn lại, thuộc tính app:layout\_constraintDimensionRatio cho phép gán tỷ lệ giữa chiều rộng và chiều cao, ví dụ: app:layout\_constraintDimensionRatio="2:1" chiều rộng gấp đôi chiều cao

## 7. Chain - xích các View lại

Các View ràng buộc qua lại các cạnh tiếp giáp nhau sẽ tạo thành một xích các View. Có hai loại xích các phần tử theo phương ngang và theo phương đứng. Lúc đó, phần tử đầu tiên có chức năng thiết lập chung một số thông số về xích.



Một xích theo phương ngang

Phần tử đầu của xích thiết lập kiểu xích bằng thuộc tính: app:layout\_constraintHorizontal\_chainStyle và app:layout\_constraintVertical\_chainStyle tùy theo xích đứng hay ngang, mặc định xích có kiểu spread

**Xích "spread"**:

*app:layout\_constraintHorizontal\_chainStyle="spread"* hoặc *app:layout\_constraintVertical\_chainStyle="spread"*

**Xích "packed"**:

*app:layout\_constraintHorizontal\_chainStyle="packed"* hoặc *app:layout\_constraintVertical\_chainStyle="packed"*

**Xích "spread\_inside"**:

*app:layout\_constraintHorizontal\_chainStyle="spread\_inside"*hoặc *app:layout\_constraintVertical\_chainStyle="spread\_inside"*

Với loại **spread\_inside** và **spread** theo phương của xích các phần tử có thể gán kích thước bằng 0 và điều chỉnh lại theo trọng số **weight** với ý nghĩa tương tự như [LinearLayout](https://xuanthulab.net/su-dung-linearlayout-trong-android.html), thuộc tính thiết lập trọng số là: app:layout\_constraintHorizontal\_weight và app:layout\_constraintVertical\_weight

# 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 so với vùng không gian của chính nó, 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 ) |

# 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** |

# Button trong Android

## 1.Button trong Android

**Button** là một loại View, nó hiện thị nút bấm để chờ người dùng bấm vào. Button kế thừa từ [**TextView**](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html) nên các thuộc tính, thiết lập cho TextView ở phần trước là có hiệu quả như đối với Button

## 2. Các thuộc tính của Button

Do **Button** mở rộng từ **TextView** nên nó có đầy đủ các thuộc tính của TextView. Ngoài các thuộc tính được kế thừa đó ra có một số thuộc tính cần lưu ý:

|  |  |
| --- | --- |
| **#** | **Thuộc tính** |
| 1 | **Gán Drawable vào các biên nút bấm**  **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   |  |  | | --- | --- | | Thuộc tính | Ý nghĩa | | android:drawableLeft | Gán ảnh vào biên trái button | | android:drawableRight | Gán ảnh vào biên phải button | | android:drawableTop | Gán ảnh vào biên trên button | | android:drawableBottom | Gán ảnh vào biên dưới button | | android:drawablePadding | Thiết lập khoảng cách ảnh Drawable đến nội dung dùng thuộc tính | |
| 2 | **Tùy biến nền Button**  **android:background:** gán màu nền cho Button  Cách thiết lập vào background giá trị màu theo HEX, theo giá trị màu như vậy thì nền đã thay đổi như mong muốn, tuy nhiên hiệu ứng khi nhấn, hiệu ứng khi Button đang focus bị mất.  Thuộc tính trên cũng dùng để gán một Drawable cho một Button |

# M. 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**) |
| 3 | **Hiển thị ảnh** |
|  | **android:src** – Hiển thị ảnh theo dạng content, ảnh này sẽ nằm đè lên ảnh background nếu xét thuộc tính android:background |

# N. CheckBox, Switch, ToggleButton, RadioButton và TextClock trong Android

## 1. CompoundButton

**CompoundButton** là một lớp cơ abstract mở rộng từ [TextView](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html) ([Button](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html)), từ lớp này nó được mở rộng để xây dựng các View là: **CheckBox, RadioButton, Switch, ToggleButton**

Các lớp mở rộng *CheckBox, RadioButton, Switch, ToggleButton* lưu lại các thuộc tính, ứng xử tương tự như TextView, Button nhưng có thêm hai trạng thái là checked và unchecked

Một số thuộc tính, phương thức chung cho *CheckBox, RadioButton, Switch, ToggleButton*

* android:checked thiết lập trạng thái checked, unchecked với giá trị "true" hay "false"
* android:button để gán Drawable vào View (vẽ trạng thái cho CheckBox, RadioButton ...)
* android:buttonTint để gán màu Tint

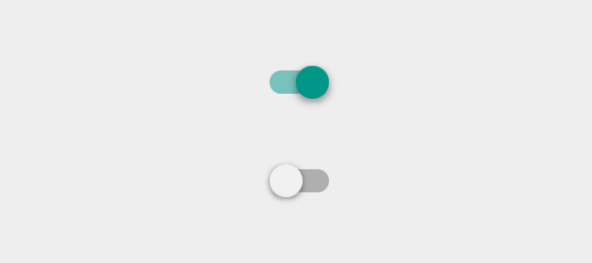
## 2. Checkbox

Lưu ý khi sử dụng Checkbox nó có các thuộc tính tương tự như TextView, Button

## 3. Swicth / SwitchCompat

**Switch** là loại View, mở rộng từ [CompoundButton](https://xuanthulab.net/su-dung-checkbox-trong-lap-trinh-android.html), nên nó có các thiết lập, thuộc tính, triển khai code và XML tương tự như [TextView](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html), [Button](https://xuanthulab.net/button-trong-androi.html), [CheckBox](https://xuanthulab.net/su-dung-checkbox-trong-lap-trinh-android.html) nên bạn có thể tham khảo các nội dung đó trước.

Loại View này cho phép bật tắt qua lại giữa hai trạng thái bằng cách bấm, hoặc kéo một cái công tắc gạt (thumb) trượt trên một đường ngắn (track). Có hai phiên bản **Switch** (android.widget.Switch) và **SwitchCompat**(android.support.v7.widget.SwitchCompat), để đảm bảo tính năng hiện đại tương thích trên nhiều thiết bị khi sử dụng Library Support nên sử dụng **SwitchCompat**

[](https://4.bp.blogspot.com/-OX2nm0mSh5U/Wrp8y-A4kkI/AAAAAAAAREg/pV-zBEIR0FIxyKatKve-VciuGLoZNmrDACLcBGAs/s1600/switch.png)

**Các thiết lập của Switch :**

* **Thiết lập dòng text: on / off**

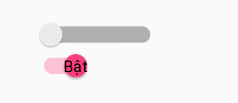
android:textOn="Bật"

android:textOff="Tắt"

app:showText="true"

* **Thay đổi độ rộng phần trượt**

Trong XML thiết lập: *app:switchMinWidth="100dp"*

[](https://3.bp.blogspot.com/-TeQaux42bmM/WrqG2aVlP2I/AAAAAAAARE8/Ql8LIkCzjFc4KpNzDXxOnoCF1VxmwwtqACLcBGAs/s1600/switch.png)

* **Màu của Thumb và Track trong Switch**

Trong XML sử dụng thuộc tính **app:thumbTint**, **app:trackTint** để thiết lập màu, có thể gán màu trực tiếp ví dụ như:

app:thumbTint="#0d47a1"

app:trackTint="#cc1e1e"

[https://1.bp.blogspot.com/-mOVKkbaSd6o/WrstQXmPKoI/AAAAAAAARFM/Lzi8AnqyudknPCtiu3S8ArrMMnngrWLHwCLcBGAs/s1600/switchcolor.PNG](https://1.bp.blogspot.com/-mOVKkbaSd6o/WrstQXmPKoI/AAAAAAAARFM/Lzi8AnqyudknPCtiu3S8ArrMMnngrWLHwCLcBGAs/s1600/switchcolor.PNG)

Việc gán màu trực tiếp thì làm cho màu của Thumb, Track không có sự thay đổi giữa trạng thái bật và tắt. Nếu muốn thiết lập màu cho cả ở trạng thái bật, tắt thì không gán giá trị màu trực tiếp mà gán giá trị Selector với cấu trúc như ví dụ:

Có một Selector drawable/switch\_thumb.xml

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

*<!--Màu ở trạng thái bật-->*

<item android:state\_checked="true" android:color="#38ae20" />

*<!--Màu ở trạng thái bình thường (tắt) -->*

<item android:color="#d9c9c9" />

</selector>

Tương tự tạo ra một Selector cho track: drawable/switch\_track.xml

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state\_checked="true" android:color="#80967b" />

<item android:color="#464141" />

</selector>

Gán vào Switch ở XML bằng cách

app:thumbTint="@drawable/switch\_thumb"

app:trackTint="@drawable/switch\_track"

* **Gán Drawable vào Thumb và Track**

Tùy chọn Drawable cho track dùng thuộc tính **app:track**, và gán Drawable cho thumb dùng thuộc tính **android:thumb**

Ví dụ có Drawable @drawable/switch\_mytrack.xml (vẽ một hình chữ nhật cao 30dp)

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item android:gravity="center\_vertical|fill\_horizontal"

android:start="4dp"

android:end="4dp">

<shape android:shape="rectangle" android:tint="#71c25b">

<corners android:radius="10dp" />

<solid android:color="#707c70" />

<size android:height="30dp" />

</shape>

</item>

</layer-list>

Có drawable drawable/switch\_mythumb.xml (vẽ một hình vuông 60x60dp)

<?xml version="1.0" encoding="utf-8"?>

<layer-list

xmlns:android="http://schemas.android.com/apk/res/android">

<item android:width="60dp" android:height="60dp">

<shape android:shape="rectangle">

<solid android:color="#963e3e" />

</shape>

</item>

</layer-list>

Gán nó vào Switch

app:track="@drawable/switch\_mytrack"

android:thumb="@drawable/switch\_mythumb"

## 4. ToggleButton

**ToggleButton** là loại View hiện thị hai trạng thái on, off (checked, unchecked) nó kế thừa từ CompoundButton nên cách sử dụng của nó giống với **CheckBox** chỉ là cách hiện thị thông tin khác nhau một chút (nên về chi tiết, cách triển khái ban đọc: [CheckBox Android](https://xuanthulab.net/su-dung-checkbox-trong-lap-trinh-android.html) trước)

**ToggleButton** mặc định hiện thị một dòng text (cho biết trạng thái on, off: ví dụ ON/OFF, BẬT/TẮT ... dòng text này do bạn thiết lập), đồng thời một dòng kẻ phía dưới

**Các thiết lập ToggleButton :**

* **Thiết lập style**

Có sẵn một số kiểu thể hiện ToggleButton có sẵn trong thư viện bạn có thể sử dụng như:



* + style="@android:style/Widget.Button.Toggle"
  + style="@android:style/Widget.Holo.Light.Button.Toggle"
  + style="@android:style/Widget.Material.Button.Toggle"
* **Tùy chọn hiện thị**

Tương tự CheckBox, RadioButton ... bạn có thể thiết lập android:backgroundgán một Selector trong đó chứa các Drawable cho các trạng thái checked/unchecked. Ví dụ có:

drawable/toggle\_state\_on.xml

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item>

<shape android:shape="rectangle">

<stroke android:color="#ef5400" android:width="5dp" />

</shape>

</item>

<item android:bottom="5dp">

<shape android:shape="rectangle">

<solid android:color="#6dd988"/>

</shape>

</item>

</layer-list>

drawable/toggle\_state\_off.xml

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item>

<shape android:shape="rectangle">

<stroke android:color="#97928f" android:width="5dp" />

</shape>

</item>

<item android:bottom="5dp">

<shape android:shape="rectangle">

<solid android:color="#a9b7c1"/>

</shape>

</item>

</layer-list>

drawable/toggle\_custom.xml

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:drawable="@drawable/toggle\_state\_on"

android:state\_checked="true" />

<item

android:drawable="@drawable/toggle\_state\_off"

android:state\_checked="false" />

</selector>

Sau đó thiết lập vào ToggleButton bằng: **android:background="@drawable/toggle\_custom"**

## 5. RadioButton và RadioGroup

Sử dụng RadioButton và RadioGroup để tạo ra giao diện tùy chọn một tronh các gợi ý đưa ra, ví dụ sử dụng RadioButton tạo ra câu hỏi trắc nghiệm một lựa chọn

**RadioButton** cũng là loại control biểu diễn trạng thái **checked/unchecked**. Có điểm khác với **CheckBox, Switch** đó là khi người dùng nhấn vào chọn nó, nó sẽ chuyển sang **checked** nếu đang là **unchecked**, nhưng chiều ngược lại nếu nó đang là **checked** thì không thể bấm vào nó để chuyển sang trạng thái **unchecked**(Tuy nhiên có thể thiết lập bằng code)

Thường **RadioButton** sẽ sử dụng cùng với **RadioGroup**, lúc đó sẽ có vài **RadioButton** bên trong **RadioGroup** và ở một thời điểm người dùng chỉ có thể chọn một.

Đã sử dụng RadioButton thì nên luôn sử dụng cùng RadioGroup

Các thuộc tính, phương thức đã thực hiện với [TextView](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html), [CheckBox](https://xuanthulab.net/su-dung-checkbox-trong-lap-trinh-android.html) hoàn toàn có thể áp dụng cho **RadioButton** mà không cần nhắc lại ở đây như:

* Thiết lập màu chữ, màu nền, nền Drawable ...
* Thiết lập hình ảnh trạng thái với thuộc tính: android:button, android:buttonTint
* Lấy trạng thái, thiết lập trạng thái checked/unchecked với: setCheck(), isChecked()
* Lắng nghe sự kiện thay đổi trạng thái với listener gán bằng: setOnCheckedChangeListener()

**Các thiết lập cho RadioButton và RadioGroup**

Trong một nhóm các RadioButton (bên trong RadioGroup) bạn thiết lập một RadioButton ở trạng thái checked mặc định bằng một trong hai cách

**1** phần tử RadioButton nào ở trạng thái checked mặc định thì cho nó thuộc tính: **android:checked="true"**

**2** nếu không sử dụng cách 1 thì trong RadioGroup cho thêm thuộc tính: **android:checkedButton="@id/idradio"**, với idradio là id của RadioButton sẽ checked mặc định

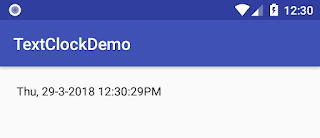
Nên nhớ nguyên tắc sử dụng RadioButton theo đúng mục đích thiết kế của nó: **chọn một phương án trong các phương án đưa ra**, nếu có nhu cầu chọn nhiều thì chuyển sang sử dụng CheckBox chứ không cố ép sử dụng RadioButton

Các tùy chọn khác như:

* Để thay đổi đổi layout RadioGroup theo hướng ngang hay đứng dùng thuộc tính **android:orientation** gán bằng vertical (đứng) hay horizontal(ngang)
* Màu sắc icon trạng thái của RadioButton thay đổi bằng thuộc tính **android:buttonTint**

## 6. TextClock

**TextClock** là loại View mở rộng từ **TextView** dùng để hiện thị ngày - giờ hệ thống hiện tại.



Giờ hệ thống thay đổi sẽ lập tức tự động cập nhật thông tin đó vào **TextClock**, cách dòng thông tin hiện thị bạn định dạng thông qua định dạng hiện thị **android:format24Hour** và **android:format12Hour**, nếu hệ thống Android người dùng thiết lập chọn hiện thị giờ theo 24H thì nó sẽ dùng chuỗi định dạng **android:format24Hour**

Chuỗi định dạng cũng có thể thiết lập bằng code như:

TextClock textClock = findViewById(R.id.textclock);

String formatdate = "E, d-M-yyyy k:m:sa";

textClock.setFormat12Hour(formatdate);

textClock.setFormat24Hour(formatdate);

Thiết lập màu sắc, kiểu chữ, font chữ ... tương tự TextView. Nếu muốn tùy chọn hiện thị màu sắc, từng thành phần cho dòng text hiện thị hãy dùng Spanned được trình bày tại: [Spanned trong TextView](https://xuanthulab.net/textview-trong-lap-trinh-android-hien-thi-text-html-spannablestring.html#span)

Thông tin hiện thị trên **TextClock** căn cứ vào chuỗi định dạng ngày tháng gán cho nó. Chuỗi ngày bạn tạo ra bằng cách sử dụng tổ hợp các ký tự mang ý nghĩa thông tin định dạng như sau:

|  |  |
| --- | --- |
| **Ký tự** | **Ý nghĩa** |
| **y** | Hiện thị năm ví dụ: yy (99), yyyy (1999) |
| **M** | Tháng - Ví dụ: M, MM -> 08, July |
| **d** | Ngày (1 ... 31) |
| **E** | Thứ - (E, EE) - Tue, Tuesday |
| **a** | Kỹ hiệu AM, PM |
| **H** | Giờ (0 - 23) |
| **k** | Giờ (1 - 24) |
| **K** | Giờ theo AM, PM (0 - 12) |
| **h** | Giờ theo AM, PM (1-12) |
| **m** | Phút (của giờ hiện tại) |
| **s** | Giây (của phút hiện tại) |
| **S** | Mini giây |

Ví dụ: một số chuỗi tổ hợp từ ký tự trên (các ký tự nếu nằm trong cặp '' sẽ không được phân tích)

* **"d-M-Y"** hiện thị dạng: 29-3-18
* **"d-M-yyyy"** hiện thị dạng: 29-3-2018
* **"'Ngày' d 'tháng' M 'năm' yyyy"** hiện thị: Ngày 29 tháng 3 năm 2018
* **"hh:mm a"** hiện thị 01:22 PM
* **"E, d/M/yyy hh:mm:ss a"** hiện thị: Thu, 29/3/2018 01:24:48 PM