Mục lục

[1 Lịch sử Android 3](#_Toc281926286)

[2 Khái niệm Android 3](#_Toc281926287)

[3 Kiến trúc Android 4](#_Toc281926288)

[4 Phát triển ứng dụng trên Android 8](#_Toc281926289)

[4.1 Ngôn ngữ lập trình 8](#_Toc281926290)

[4.2 Một số lưu ý 8](#_Toc281926291)

[4.2.1 Vấn đề phân mảnh phiên bản Android trên thị trường: 8](#_Toc281926292)

[4.2.2 Vấn đề kích thước màn hình 11](#_Toc281926293)

[4.3 Môi trường lập trình cho Android 12](#_Toc281926294)

[4.3.1 Các thành phần cơ bản của một project Android trên Eclipse 13](#_Toc281926295)

[4.3.2 File AndroidManifest.xml 14](#_Toc281926296)

[4.4 Các thành phần cơ bản của một ứng dụng Android 16](#_Toc281926297)

[4.4.1 Activity 16](#_Toc281926298)

[4.4.2 Intent 20](#_Toc281926299)

[4.4.3 Service 23](#_Toc281926300)

[4.4.4 Content Provider 26](#_Toc281926301)

[4.4.5 View 30](#_Toc281926302)

[4.4.6 Lưu trữ dữ liệu 35](#_Toc281926303)

# Lịch sử Android

Tháng 7/2005, Google mua lại công ty phát triển phần mềm điện thoại Android, nhưng chưa công bố sẽ sử dụng cho mục đích gì. Để trả lời cho câu hỏi đó, tháng 11/2007, sau 2 năm phát triển, Google công bố hệ điều hành điện thoại di động mã nguồn mở Android, cùng với sự thành lập “Liên minh di động mở” (Open Handset Alliance) bao gồm hơn 65 nhà sản xuất phần cứng điện thoại lớn trên thế giới như Intel, HTC, China Mobile, T-Mobile,....



Hình 1 - Các thành viên của "Liên minh di động mở"

Tứ năm 2007, hệ điều hành Android đã trải qua nhiều lần cập nhật, với phiên bản gần đây nhất là Gingerbread 2.3, ra ngày 06/12/2010. Tính đến tháng 6/2010, đã có khoảng 70 mẫu điện thoại di động sử dụng hệ điều hành Android, chưa kể các thiết bị điện tử khác như máy tính bảng (tablet computers), ebook readers, Google TV,...

# Khái niệm Android

Android là hệ điều hành điện thoại di động mở nguồn mở miễn phí do Google phát triển dựa trên nền tảng của Linux. Bất kỳ một hãng sản xuất phần cứng nào cũng đều có thể tự do sử dụng hệ điều hành Android cho thiết bị của mình, miễn là các thiết bị ấy đáp ứng được các tiêu chuẩn cơ bản do Google đặt ra (có cảm ứng chạm, GPS, 3G,...) (Xem thêm: [**Android Compatibility Definition Document**](http://source.android.com/compatibility/android-2.1-cdd.pdf)**)**

Các nhà sản xuất có thể tự do thay đổi phiên bản Android trên máy của mình một cách tự do mà không cần phải xin phép hay trả bất kì khoản phí nào nhưng phải đảm bảo tính tương thích ngược (backward compatibility) của phiên bản chế riêng đó.

Android là nền tảng cho thiết bị di động bao gồm một hệ điều hành, midware và một số ứng dụng chủ đạo. Bộ công cụ Android SDK cung cấp các công cụ và bộ thư viên các hàm API cần thiết để phát triển ứng dụng cho nền tảng Android sử dụng ngôn ngữ lập trình java.

Những tính năng mà nền tảng Android hổ trợ:

**Application framework:** Cho phép tái sử dụng và thay thế các thành phần sẳn có của Android.

**Dalvik virtual macine:** Máy ảo java được tối ưu hóa cho thiết bị di động.

**Intergrated browser:** Trình duyệt web tích hợp được xây dựng dựa trên [WebKit](http://webkit.org/) engine.

**Optimized graphics:** Hổ trợ bộ thư viện 2D và 3D dự vào đặc tả OpenGL ES 1.0.

**SQLite:** DBMS dùng để lưu trữ dữ liệu có cấu trúc.

Hổ trở các định dạng media phổ biến như: MPEG4, H.264, MP3, AAC, ARM, JPG, PNG, GIF.

Hổ trợ thoại trên nền tảng GSM (Phụ thuộc vài phần cứng thiết bị).

Bluetooth, EDGE, 3G và WiFi (Phụ thuộc vài phần cứng thiết bị).

Camera, GPS, la bàn và cảm biến (Phụ thuộc vài phần cứng thiết bị).

Bộ công cụ phát triển ứng dụng mạnh mẽ.

# Kiến trúc Android

Lược đồ sau thể hiện các thành phần của hệ điều hành Android:



Hình 2 - Mô hình kiến trúc nền tảng Android

Applications

Hệ điều hành Android tích hợp sẳn một số ứng dụng cơ bản như email client, SMS, lịch điện tử, bản đồ, trình duyệt web, sổ liên lạc và một số ứng dụng khác. Ngoài ra tầng này cũng chính là tầng chứa các ứng dụng được phát triển bằng ngôn ngữ Java.

Application Framwork

Tầng này của hệ điều hành Android cung cấp một nền tảng phát triển ứng dụng mở qua đó cho phép nhà phát triển ứng dụng có khả năng tạo ra các ứng dụng vô cùng sáng tạo và phong phú. Các nhà phát triển ứng dụng được tự do sử dụng các tính năng cao cấp của thiết bị phần cứng như: thông tin định vị địa lý, khả năng chạy dịch vụ dưới nền, thiết lập đồng hồ báo thức, thêm notification vào status bar của màn hình thiết bị…

Người phát triển ứng dụng được phép sử dụng đầy đủ bộ API được dùng trong các ứng dụng tích hợp sẳn của Android. Kiến trúc ứng dụng của Android được thiết kế nhằm mục đích đơn giản hóa việc tái sử dụng các component. Qua đó bất kì ứng dụng nào cũng có thể công bố các tính năng mà nó muốn chia sẻ cho các ứng dụng khác (VD: Ứng dụng email có muốn các ứng dụng khác có thể sử dụng tính năng gởi mail của nó). Phương pháp tương tự cho phép các thành phần có thể được thay thế bởi người sử dụng.

Tầng này bao gồm một tập các services và thành phần sau:

Một tập phong phú và có thể mở rộng bao gồm các đối tượng View được dùng để xây dựng ứng dụng như: list, grid, text box, button và thậm chí là một trình duyệt web có thể nhúng vào ứng dụng.

**Content Provider:** Cho phép các ứng dụng có thể truy xuất dữ liệu từ các ứng dụng khác hoặc chia sẽ dữ liệu của chúng.

**Resource Manager:** Cung cấp khả năng truy xuất các tài nguyên non-code như hình ảnh hoặc file layout.

**Notification Manager:** Cung cấp khả năng hiển thị custom alert trên thanh status bar.

**Activity Manager:** Giúp quản lý vòng đời của một ứng dụng.

Libraries

Hệ điều hành Android bao gồm một tập các bộ thư viện C/C++ được sử dụng bởi nhiều thành phần của Android system. Những tính năng này được cung cấp cho các lập trình viên thông qua bộ framework của Android. Dưới đây là một số thư viện cốt lõi:

**System C library:** một thể hiện được xây dựng từ BSD của bộ thư viện hệ thống C chuẩn (libc), được điều chỉnh để tối ưu hóa cho các thiết bị chạy trên nền Linux.

**Media libraries:** Bộ thư viện hổ trợ trình diễn và ghi các định dạng âm than và hình ảnh phổ biến.

**Surface manager:** Quản lý hiển thị nội dung 2D và 3D.

**LibWebCore:** Một web browser engine hiện đại được sử dụng trong trình duyệt của Android lần trong trình duyệt nhúng web view được sử dụng trong ứng dụng.

**SGL:** Engine hổ trợ đồ họa 2D.

**3D libraries:** Một thể hiện được xây dựng dựa trên các APIs của OpenGL ES 1.0. Những thư viện này sử dụng các tăng tốc 3D bằng phần cứng lẫn phần mềm để tối ưu hóa hiển thị 3D.

**FreeType:** Bitmap and vector font rendering.

**SQLite:** Một DBMS nhỏ gọn và mạnh mẽ.

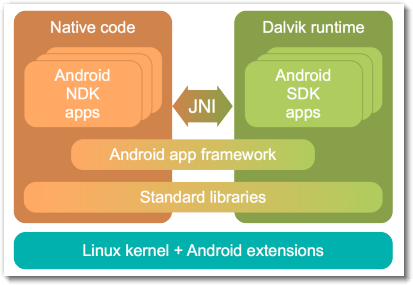
Android Runtime

Hệ điều hành Android tích hợp sẳn một tập hợp các thư viện cốt lõi cung cấp hầu hết các chức năng có sẵn trong các thư viện lõi của ngôn ngữ lập trình Java. Mọi ứng dụng của Android chạy trên một tiến trình của riêng nó cùng với một thể hiện của máy ảo Dalvik. Máy ảo Dalvik thực tế là một biến thể của máy ảo Java được sửa đổi, bổ sung các công nghệ đặc trưng của thiết bị di động. Nó được xây dựng với mục đích làm cho các thiết bị di động có thể chạy nhiều máy ảo một cách hiệu quả. Trước khi thực thi, bất kì ứng dụng Android nào cũng được convert thành file thực thi với định dạng nén Dalvik Executable (.dex). Định dạng này được thiết kế để phù hợp với các thiết bị hạn chế về bộ nhớ cũng như tốc độ xử lý. Ngoài ra máy ảo Dalvik sử dụng bộ nhân Linux để cung cấp các tính năng như thread, low-level memory management.

Linux Kernel

Hệ điều hành Android được xây dựng trên bộ nhân Linux 2.6 cho những dịch vụ hệ thống cốt lõi như: security, memory management, process management, network stack, driver model. Bộ nhân này làm nhiệm vụ như một lớp trung gian kết nối phần cứng thiết bị và phần ứng dụng.

Dưới đây là mô hinh hợp tác giữa máy ảo Dalvik và Navite code:



Hình 3 - Mô hình hợp tác giữa máy ảo Dalvik và Navite code

JNI: Java Native Interface (Tương tự khái niệm Application Programming Interface).

Java Native Interface: là một bộ framework cho phép mã lệnh viết bằng Java chạy trên máy ảo java có thể gọi hoặc được gọi bởi một ứng dụng viết bằng **native code** (Ứng dụng được viết cho một phần cứng cụ thể và trên một hệ điều hành cụ thể) hoặc những bộ thư viện viết bằng C, C++ hoặc Assembly.

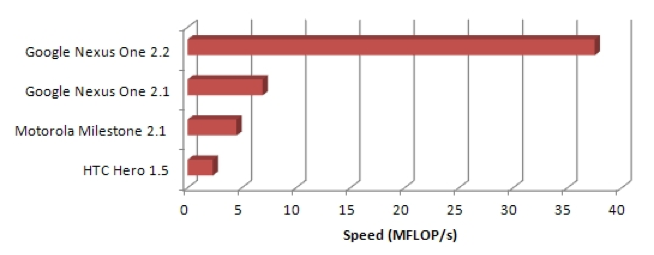
Bằng cách sử dụng JNI, Android cho phép các ứng dụng chạy trên máy ảo Dalvik có thể sử dụng những phương thức được viết bằng các ngôn ngữ cấp thấp như: C, C++, Assembly. Qua đó các nhà phát triển ứng dụng có thể xây dựng ứng dụng dựa trên các bộ thư viện viết bằng C, C++, Assembly nhằm tăng tốc độ thực thi của ứng dụng hoặc sử dụng những tính năng mức thấp mà ngôn ngữ Java không hổ trợ. Tuy nhiên người phát triển ứng dụng cần phải cân nhắc sự gia tăng độ phức tạp của ứng dụng khi quyết định sử dụng các bộ thư viện này.

# Phát triển ứng dụng trên Android

## Ngôn ngữ lập trình

Ngôn ngữ lập trình chính thức của Android là Java. Mặc dù các ứng dụng trên Android được phát triển dựa trên nền tảng Java, nhưng Android không hỗ J2ME và J2SE, là hai ngôn ngữ lập trình phổ dụng cho các thiết bị di động.

Dựa trên máy ảo Java của Sun, Google đã tinh chỉnh và phát triển nên máy ảo Dalvik để biên dịch mã Java với tốc độ biên dịch nhanh hơn và nhẹ hơn. Đến phiên bản Froyo 2.2, Android đã hỗ trợ Just-in-time Compiler (JIT) làm tăng tốc độ biên dịch Java lên gấp 2-5 lần so với các phiên bản trước.



Hình 4 - Biểu đồ so sánh tốc độ của máy ảo Java giữa các phiên bản Android (Càng lớn càng tốt)

Bằng cách sử dụng các frameworks của hãng thứ ba (VD: PhoneGap, Titanium,...), các nhà lập trình web cũng có thể phát triển ứng dụng Android một cách dễ dàng bằng các ngôn ngữ web phổ dụng như HTML, CSS, Javascript. Tuy nhiên số lượng các ứng dụng đi theo chiều hướng này chưa nhiều.

Ngoài ra các ứng dụng của Android sử dụng SQLite để quản lý cơ sở dữ liệu.

## Một số lưu ý

### Vấn đề phân mảnh phiên bản Android trên thị trường:

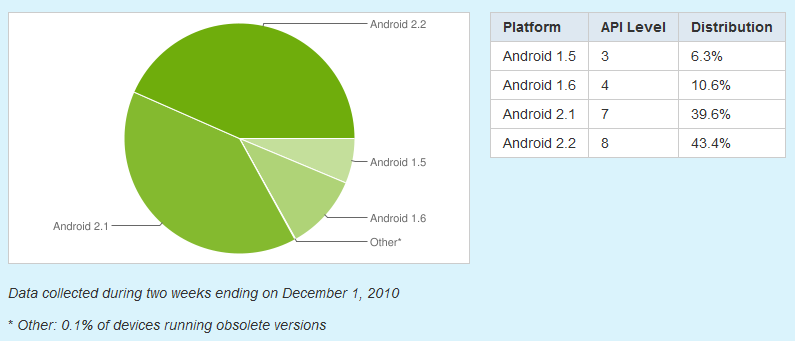
Do Android là hệ điều hành mã nguồn mở và miễn phí, bất kỳ một nhà sản xuất phần cứng nào cũng có thể sử dụng và tùy biến Android để cài đặt trên các thiết bị của mình. Điều này dẫn đến vấn để phân mảnh (fragmentation) khi trên thị trường tồn tại nhiều mẫu điện thoại Android, mỗi mẫu chạy một phiên bản Android khác nhau. Đôi khi các phiên bản Android này còn khác nhau về giao diện hay chức năng, do các nhà sản xuất tự chế thêm vào mã nguồn của Android. Một ví dụ tiêu biểu chính là giao diện Sense UI của HTC.

|  |  |
| --- | --- |
|  |  |

Hình 5 - So sánh giao diện nguyên bản của Android và giao diện Sense của riêng hãng HTC thiết kế

Mặt khác, do việc cập nhật hệ điều hành phụ thuộc hoàn toàn vào các nhà sản xuất, thế nên sau khi Google công bố phiên bản Android mới , người sẽ buộc phải chờ khá lâu mới được cập nhật.

Do vấn đề phân mảnh này, trên thị trường một lúc tốn tại nhiều phiên bản Android khác nhau, và người phát triển ứng dụng Android buộc phải xem xét đến khả năng ứng dụng do mình viết ra có thể sẽ không chạy đuợc hay thiếu di một số chức năng quan trọng khi chaỵ trên các máy chạy phiên bản Android cũ. Một ví dụ tiêu biểu là ứng dụng chính thức của mạng xã hội Twitter chỉ có thể chạy trên phiên bản Android 2.1, tức có 1/2 thị trường người dùng điện thoại Android không thể sử dụng ứng dụng này.



Hình 6 - Thống kê số lượng điện thoại sử dụng các phiên bản Android

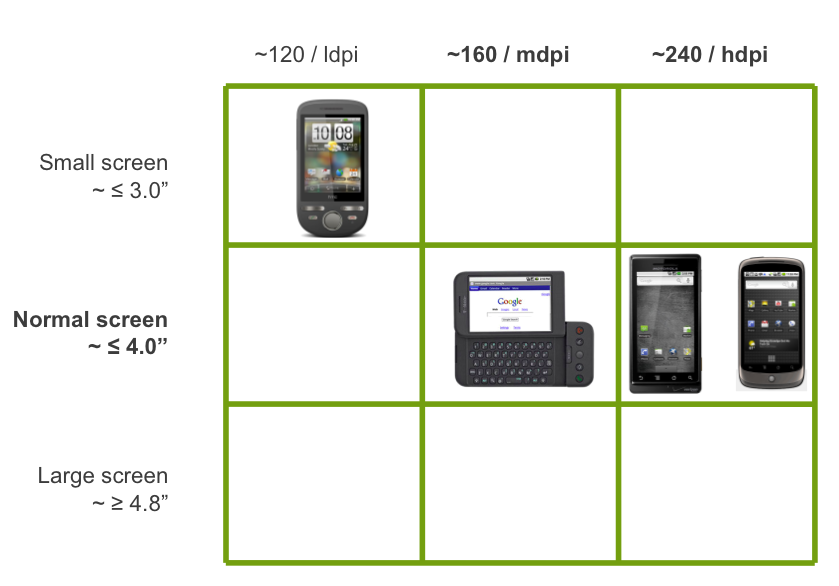
(Nguồn -<http://developer.android.com/resources/dashboard/platform-versions.html> – T12/2010)

### Vấn đề kích thước màn hình

Một lần nữa, do trên thị trường có rất nhiều nhà sản xuất điện thoại Android, và mặc dù Google yêu cầu tất cả mọi điện thoại Android phải đáp ứng 1 số tiêu chí chung về phần cứng (màn hình cảm ứng, GPS, 3G,...), các nhà sản xuất hoàn toàn tự do quyết định kích cỡ và độ phân giải điện thoại.

Như vậy, các nhà phát triển ứng dụng Android phải chuẩn bị trước khả năng ứng dụng của mình sẽ chạy trên một loạt các kích cỡ màn hình lớn nhỏ khácnhau.

Tuy nhiên, có một điều chắc chắn: 100% các điện thoại Android đều sử dụng màn hình cảm ứng đa điểm, còn bàn phím vật lý hay cảm ứng quang thì tùy máy. Dưới đây là bảng chứa thông số các kích thước màn hình của các thiết bị sử dụng Android tiêu biểu:



Hình 7 - Bảng thống kê các kiểu màn hình điện thoại Android

(Nguồn: slides thuyết trình giới thiệu Android – Hội thảo Google I/O T6/2010)

## Môi trường lập trình cho Android

Android SDK bao gồm các công cụ riêng lẻ như: debugger, các thư viện, trình giả lập điện thoại Android, các tài liệu hỗ trợ và code mẫu. Hiện Android cung cấp bộ công cụ này trên nhiều nền tảng hệ điều hành khác nhau (Windows, Linux, Mac,...), miễn là có sẵn Java Development Kit, Apache Ant và Python2.2 trở lên.

Môi trường lập trình (IDE) chính thức của Android là Eclipse (từ phiên bản 3.2) với sự hỗ trợ của plugin Android Development Tools (ADT). Tuy nhiên, người lập trình có thể sử dụng bất kỳ 1 IDE hay trình soạn thảo văn bản nào để viết code Java và XML rồi biên dịch nên ứng dụng hoàn chỉnh bằng cách sử dụng dòng lệnh (command lines).

Ứng dụng Android được đóng gói thành các file .apk và đuợc lưu trong thư mục /data/app của hệ điều hành Android.Java Development Kit (JDK) 5.0.

Một số công cụ hỗ trợ lập trình Android tiêu biểu:

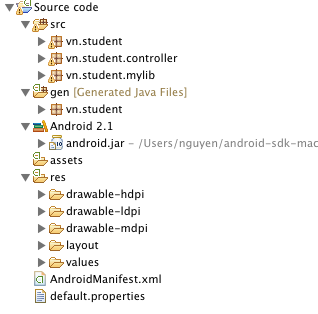
**SQLite Manager:** Là một addon của Firefox giúp quản lí cơ sở dữ liệu SQLite của Android.

**DroidDraw:** Giúp thiết kế file XML giao diện ứng dụng.

**Balsamiq Mockups và AdobeFireworks:** Giúp nhanh chóng phác thảo ý tưởng và giao diện sơ bộ của ứng dụng.

**StarUML:** Vẽ các lược đồ UML hỗ trợ phân tích thiết kế.

### Các thành phần cơ bản của một project Android trên Eclipse



Hình 8 - Cấu trúc thư mục và file của một dự án phần mềm Android trên Eclipse

**AndroidManifest.xml:** file XML mô tả ứng dụng và các thành phần đuợc cung cấp bởi ứng dụng (activities, services,...).

**build.xml:** Một file chứa mã script Ant (ant.apache.com) nhằm compile và cài đặt ứng dụng lên máy.

**default.properties**: file property tạo bởi script Ant trên.

**bin/** : nơi chứa ứng dụng sau khi được compile.

**bin/classes/** : chứa các lớp Java đã được compile.

**bin/classes.dex** : chứa các file executable tạo bởi các lớp Java.

**bin/yourapp.ap\_** : chứa các tài nguyên của ứng dụng, đóng gói thành 1 file zip.

**bin/yourapp-debug.apk** hay **bin/yourapp-unsigned.apk** : chứa chính ứngdụng Android của ta.

**libs/** : nơi chứa các file Java JAR ứng dụng yêu cầu (third party).

**src/** : nơi chứa mã nguồn Java của ứng dụng.

**res/** : chứa các tài nguyên của ứng dụng, như các icons, GUI layouts,...

**res/drawable/** : chứa file hình ảnh (PNG, JPEG,...).

**res/layout/** : chứa UI layout, dưới dạng XML.

**res/menu/** : chi tiết các menu, dưới dạng XML.

**res/raw/** : chứa các file khác (CSV chứa thông tin account,...).

**res/values/** : chứa các strings, dimensions,...

**res/zml/** : chứa các file XML khác cần cho ứng dụng.

**assets/** : nơi chứa các files tĩnh (static) được yêu cầu đi kèm với ứng dụng.

### File AndroidManifest.xml

Là nền tảng của mọi ứng dụng Android, file AndroidManifest.xml được đặt trong thư mục root và cho biết những thành phần có trong ứng dụng của: các activities, các services,...cũng như cách các thành phần ấy gắn bó với nhau.

Mỗi file manifest đều bắt đầu với một thẻ manifest:

|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.commonsware.android.search">  ...  </manifest> |

Khi tạo file manifest, điều quan trọng nhất là phải cung cấp thuộc tính của package, tức tên của Java package dùng làm cơ sở cho ứng dụng của ta. Sau khi đã đặt tên package, sau này khi cần ghi tên lớp trong file manifest, ta có thể rút ngắn, chẳng hạn với lớp “com.yourapp.android.search.Someclass” ta chỉ cần ghi “.Someclass” là đủ.

Các thành phần manifest khác là :

**uses-persmission:** chỉ định các quyền mà ứng dụng của ta đuợc cấp để hoạt động trôi chảy (như đã nói, các ứng dụng Android nằm dưới nhiều lớp bảo mật khác nhau).

**permission**: chỉ định các quyền mà các activities hay services yêu cầu các ứng dụng khác phaỉ có mới được truy cập dữ liệu của ứng dụng của ta.

**instrumentation**: chỉ định phần code cần được gọi khi xảy ra những sự kiện quan trọng (chẳng hạn khởi động activities) nhằm phục vụ việc ghi chú (logging) và tra soát (monitoring)

**uses-library**: nhằm kết nối với các thành phần có sẵn của Android (như service tra bản đồ,...)

**uses-sdk:** có thể có hoặc không, chỉ ra phiên bản củaAndroid mà ứng dụng này yêu cầu.

**application**: định nghĩa phần trung tâm của ứng dụng của file manifest.

Ví dụ file AndroidManifest:

|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.commonsware.android">  <uses-permission  android:name="android.permission.ACCESS\_LOCATION" />  <uses-permission  android:name="android.permission.ACCESS\_GPS" />  <uses-permission  android:name="android.permission.ACCESS\_ASSISTED\_GPS" />  <uses-permission  android:name="android.permission.ACCESS\_CELL\_ID" />  <application>  ...  </application>  </manifest> |

Tất nhiên, phần quan trọng của 1 file manifest chính là thành phần application. Mặc định, khi ta tạo 1 project Android mới, ta có sẵn 1 thành phần activity:

|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.commonsware.android.skeleton">  <application>  <activity android:name=".Now" android:label="Now">  <intent-filter>  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  </application>  </manifest> |

Thành phần này cung cấp các thông tin sau

**android:name** : tên class hiện thực activity này.

**android:label** : tên activity.

**intent-filter** : Một thành phần con, chỉ ra dưới điều kiện nào thì activity này được hiển thị.

## Các thành phần cơ bản của một ứng dụng Android

### Activity

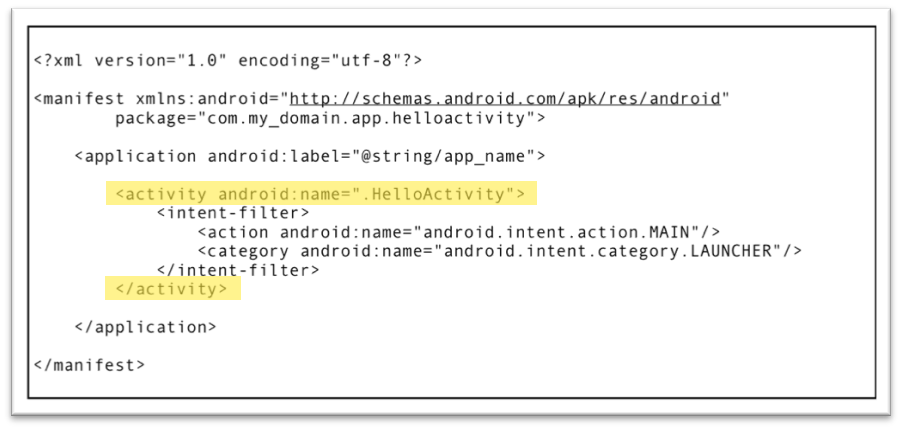
Định nghĩa Activity

Là thành phần tối quan trọng của bất kỳ một ứng dụng Android nào. Thuật ngữ Activity chỉ một việc mà người dùng có thể thực hiện trong một ứng dụng Android. Do gần như mọi activity đều tương tác với người dùng, lớp Activity đảm nhận việc tạo ra một cửa sổ (window) để người lập trình đặt lên đó một giao diện UI với setContentView(View). Một activity có thể mang nhiều dạng khác nhau: Một cửa sổ toàn màn hình (full screen window), một cửa sổ floating (với windowsIsFloating) hay nằm lồng bên trong 1 activity khác (với ActivityGroup).

Có 2 phương thức mà gần như mọi lớp con của Activity đều phải hiện thực:

* **onCreate(Bundle)** - Nơi khởi tạo activity. Quan trọng hơn, đây chính người lập trình gọi setContentView(int) kèm theo layout để thể hiện UI của riêng mình. Đồng thời còn có findViewById(int) giúp gọi các widget (buttons, text boxes, labels,..) để dùng trong UI.
* **onPause()** - Nơi giải quyết sự kiện người dùng rời khỏi activity. Mọi dữ liệu được người dùng tạo ra tới thời điểm này cần phải được lưu vào ContentProvider.

Để có thể sử dụng Context.startActivity(), mọi lớp activity đều phải được khai báo với tag <activity> trong file AndroidManifest.xml.



Hình 9 - Khai báo tag <activity> trong file AndroidManifest.xml

Vòng đời của một activity

Các activity được quản lí dưới dạng các activity stack - First-In-Last-Out: Khi một activity mới được khởi tạo, nó sẽ được đưa lên trên cùng stack, các activity khác muốn chạy trên nền (foreground) trở lại thì cần phải chờ tới khi Activity mới này kết thúc.

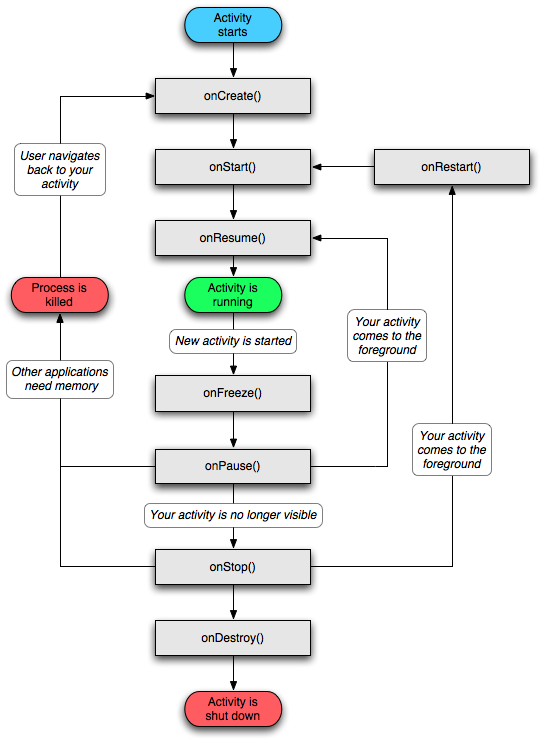
Một Activity có 4 trạng thái:

**Active** hay **Running:** Khi một activity đang chạy trên màn hình.

**Paused:** Khi một activity vẫn đang chạy trên màn hình nhưng đang bị một activity trong suốt (transparent) hay không chiếm toàn màn hình hiển thị phía trên. Tuy vẫn lưu trữ dữ liệu, nhưng các **paused activity** này sẽ bị hệ thống bắt chấm dứt khi đang thiếu bộ nhớ trầm trọng.

**Stopped:** Khi 1 activity bị che khuất hoàn toàn bởi 1 activity khác. Tuy vẫn lưu trữ dữ liệu, nhưng các stopped activity này sẽ thường xuyên bị hệ thống bắt chấm dứt để dành chỗ cho các tiến trình khác.

**Killed hay Shut down:** Khi 1 activity đang paused hay stopped, hệ thống sẽ xóa activity ấy ra khỏi bộ nhớ.



Hình 10 - Lược đồ vòng đời của 1 activity

Dựa vào lược đồ trên, thấy được có 3 vòng lặp quan trọng sau:

**Vòng đời toàn diện (Entire Lifetime):** Diễn ra từ lần gọi onCreate(Bundle) đầu tiên và kéo dài tới lần gọi onDestroy() cuối cùng.

**Vòng đời thấy được (Visible Lifetime):** Diễn ra từ khi gọi onStart() và kéo dài tới khi gọionStop(). Ở vòng đời này, activity được hiển thị trên màn hinh nhưng có thế không tương tác với người dùng ở trên nền. Các phương thức onStart(0 và onStop() có thể được gọi nhiều lần.

**Vòng đời trên nền (Foreground Lifetime):** Diễn ra từ khi gọi onResume(0 và kéo dài tới khi gọi onPause(). Ở vòng đời này, activity nằm trên mọi activity khác và tương tác được với người dùng. 1 activity có thể liên tục thay đổi giữa 2 trạng thái paused và resumed, chẳng hạn khi thiết bị sleep hay 1 intent mới được đưa tới.

Toàn bộ vòng đời của 1 activity được định nghĩa nhờ các phương thức sau:

|  |
| --- |
| public class Activity extends ApplicationContext {  protected void onCreate(Bundle savedInstanceState);  //Gọi khi mới tạo activity để setup các view, binding dữ liệu,...  //Kèm theo sau luôn là onStart().  protected void onRestart();  //Gọi sau khi activity bị stopped và trước khi được khởi động lại.  //Kèm theo sau luôn là onStart().  protected void onStart();  //Gọi khi activity hiện lên trước mắt người dùng.  //Kèm theo sau là onResume() nếu activity hiện lên nền hay onStop() nếu bị ẩn đi.  protected void onResume();  //Gọi khi activity bắt đầu tương tác với người dùng và đang trên cùng của activity stack.  //Kèm theo sau luôn là onPause().  protected void onPause();  //Gọi khi hệ thống sắp khởi động lại 1 activity khác trướcđó.  //kèotheo sau là onresume nếu activity trở lại trên cùng hay onStop() nếu bị ẩn đi.  protected void onStop();  //Gọi khi activity không còn hiển thị trước người dùng  //Kèm theo sau là onRestart() nếu activity hiện lên trở lại hay onDestroy nếu sắp xoá activity đi.  protected void onDestroy();  //Gọi ngay trước khi kết thúc activity, xảy ra khi hàm finish() được gọi hoặc khi hệ thống yêu cầu buộc phải kết thúc.  } |

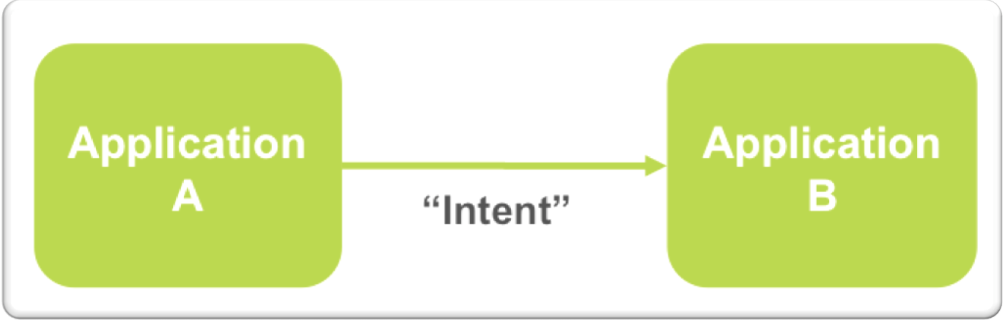
### Intent

Định nghĩa Intent

Khi **Tim Berners** phát minh ra giao thức Hypertext Transfer Protocol (HTTP), ông cũng đã phát minh ra một định dạng URLs chuẩn. Định dạng này là một hệ thống các động từ đi kèm các địa chỉ. Địa chỉ sẽ xác định nguồn tài nguyên như Web page, hình ảnh hay các server-side program. Động từ sẽ xác định cần phải làm cái gì với nguồn tài nguyên đó: GET để nhận dữ liệu về, POST để đưa dữ liệu cho nó để thực thi một công việc nào đó. Khái niệm **Intent** cũng tương tự, **Intent** là một mô tả trừu tượng của một hành động được thực thi. Nó đại diện cho một hành động đi kèm với một ngữ cảnh xác định. Với **Intent** thì có nhiều hành động và nhiều component (Một thể hiện của một class java dùng để thực thi các hành động được đặc tả trong **Intent**) dành cho **Intent** của Android hơn là so với HTTP verbs (POST, GET) và nguồn tài nguyên (hình ảnh, web page) của giao thức HTTP, tuy nhiên khái niệm vẫn tương tự nhau.

**Intent** được sử dụng với phương thức **startActivity()** để mở một **Activity**, và dùng với **broadcastIntent** để gởi nó đến bất kì **BroadcastReceiver** liên quan nào, và dùng với [**startService(Intent)**](http://developer.android.com/reference/android/content/Context.html#startService%28android.content.Intent%29), **bindService(Intent, ServiceConnection, int)** để giao tiếp với các **Service** chạy dưới nền.

Intent cung cấp một chức năng cho phép kết nối hai chương trình khác nhau trong quá trình thực thi (runtime) (Cung cấp khả năng cho phép hai chương trình khác nhau giao tiếp với nhau). Chức năng quan trọng và được sử dụng nhiều nhất của một **Intent** là mở một **Activity**, nơi mà nó có thểđược dùng như một vật kết nối các Activity lại với nhau (Truyền thông tin giữa hai Activity khác nhau).

****

Hình 11 - Sử dụng Intent để trao đổi thông tin giữa hai chương trình

Thành phần chính của Intent bao gồm:

**Action:** Xác định hành động sẽ được thực thi, các hành động này có thể là: ACTION\_VIEW, ACTION\_EDIT, ACTION\_MAIN…

**Data:** Các dữ liệu được sử dụng để hành động (Action) thao tác trên nó, như bản ghi về một người nào đó trong cơ sở dữ liệu chứa các contact của thiết bị.

Ví dụ về thành phần action/data:

**ACTION\_VIEW content://contacts/people/1** – hiển thị thông tin liên lạc của người có mã là “1”.

**ACTION\_DIAL content://contacts/people/1** – gọi điện cho người có mã là “1”.

**ACTION\_DIAL tel:123** – gọi đến số điện thoại “123”

Ngoài ra, Intent còn có các thuộc tính phụ sau:

**Category:** thông tin chi tiết về hành động được thực thi, ví dụ như **CATEGORY\_LAUNCHER** có nghĩa là nó sẽ xuất hiện trong Launcher như ứng dụng có mức level cao (top-level application), trong khi **CATEGORY\_ALTERNATIVE** chứa thông tin danh sách các hành động thay thế mà người dùng có thể thực thi trên phần dữ liệu mà **Intent** cung cấp.

**Type:** Chỉ định 1 kiểudữ liệu chính xác (kiểu MIME) được mang bởi intent. Thường thì type được suy ra từ chính dữ liệu. Bằng cách thiết lập thuộc tính type, bạn có thể vô hiệu hóa sự phỏng đoán kiểu dữ liệu và chỉ định rỏ một kiểu chính xác.

**Component:** Chỉ định rõ tên của lớp thành phần (Một thể hiện của một class java dùng để thực thi các hành động được đặc tả trong **Intent**) sử dụng cho Intent . Thông thường thì nó được xác định bằng cách tìm kiếm thông tin trong **Intent** (Các thông tin như Action, data/type, và category) và nối chúngvới các component (Một thể hiện của một Class java dùng để thực thi các hành động được đặc tả trong **Intent**) có thể đáp ứng được các yêu cầu sử lý của **Intent**.

**Extracts:** là một đối tượng **Bundle** dùng để chứa các thông tin kèm theo được dùng để cung cấp thông tin cần thiết cho component. Ví dụ: Nếu có một Intent đặc tả hành động send email thì những thông tin cần chứa trong đối tượng **Bundle** là subject, body…

Intent Resolution

Intent có 2 dạng chính sau:

**Explicit Intents**: Xác định rỏ một component (Một thể hiện của một class java dùng để thực thi các hành động được đặc tả trong **Intent**) (thông qua phương thức setComponent(ComponentName) hoặc setClass(Context, Class)) cung cấp lớp sẽ thực thi các hành động được đặc tả trong **Intent**. Thông thường thì những **Intent** này không chứa bất kỳ thông tin nào khác (như category, type) mà đơn giản chỉ là cách để ứng dụng mở các **Activity** khác bên trong một **Activity**.

**Implicit Intents**: Không chỉ định một component nào cả, thay vào đó, chúng sẽ chứa đủ thông tin để hệ thống có thể xác định component có sẵn nào là tốt nhất để thực thi hiệu quả cho **Intent** đó.

Khi sử dụng **Implicit intents**, do tính chất chuyên quyền của loại **Intent** này,ta cần phải biết phải làm gì với nó. Công việc này được đảm nhiệm bởi tiến trình của **Intent resolution**, nó giúp chỉ định **Intent** đến một **Actvity**,

**BroadcastReceiver**, hoặc **Service** (hoặc thỉnh thoảng có thểlà 2 hay nhiều hơn một activity/receiver) để có thể xử lý các hành động được đặc tả trong **Intent**.

Bất cứ thành phần nào (Activity, BroadcastReceiver, Service) khi muốn sử dụng trong ứng dụng đều phải được đăng kí trong file **AndroidManifest.xml**. Trong đó cầnđịnh nghĩa một thẻ **<intent-fillter>** cung cấp các thông tin để hệ thống có thể xác định được cái mà các component này (Activity, BroadcastReceiver, Service) có thể xử lý được (những action mà component này có thể thực hiện được).

**Intent Fillter** là bản đặc tả có cấu trúc của các giá trị của **Intent** dùng để xác định component phù hợp để thực hiệncác hành động được đặc tả trong **Intent**. Một **Intent Fillter** nối các hành động, categories và dữ liệu (hoặc thông qua type hoặc sheme) trong Intent. **Intent Fillter** được khai báo trong **AndroidManifest.xml** và sử dụng thẻ **intent-fillter**.

Một Intent Fillter có các thành phần chính sau:

**Action**: Tên hành động mà component có thể thực thi.

**Type:Kiểu hành động mà component có thể thực thi.**

**Category: Phân nhóm các hành động.**

**Đối với những dữ liệu không phải là nội dung cụ thể (VD: URI) thì việc xem xét lựa chọn Intent phù hợp sẽ dựa vào lược đồ(Scheme) của dữ liệu được cung cấp (VD: http:// mailto: …)**

****Luật xác định component phù hợp Intent****

**Để xác định một thành phần là phù hợp với một Intent hệ thống sẽ tiến hành xem xét từ trên xuống.**

**Trước tiên khi một Intent được gọi, Android sẽ tìm kiếm những component (Activity, BroadcastReceiver, Service) có action-name phù hợp với Intent.**

**Nếu có component phù hợp Android sẽmở component đó lên để thực thi các hành động theo yêu cầu.**

**Nếu có nhiều hơn 1 component có action-name phù hợp thì Android sẽ yêu cầu người dùng chọn component phù hợp.**

**Ngược lại nếu không có component nào phù hợp Android sẽ tiến hành xem xét kiểu dự liệu của Intent cung cấp xem có component nào có đủ năng lực để sử lý kiểu dữ liệu đó không. Nếu khôngđược Android sẽ tiến hành xem xét scheme của dữ liệu đó để tìm kiếm component phù hợp. Nếu vẫn không tìm được component phù hợp Android sẽ tiến hành xem xét các component có chung Category với Intent để xác định component.**

### Service

Định nghĩa Service

Một service là một thành phần của ứng dụng, thể hiện mong muốn ứng dụng thực hiện các hành động trong khi không tương tác với người dùng hoặc cung cấp chức năng cho các ứng dụng khác sử dụng. Nói một cách đơn giản, service là các tác vụ (task) chạy ngầm dưới hệ thống nhằm thực hiện một nhiệm vụ nào đó. Mỗi class Service phải chứa thẻ <service> được khai báo trong file **AndroidManifext.xml**. Services có thể được bắt đầu bởi Context.startService() và Context.bindservice()

Cũng như các đối tượng của ứng dụng khác, services chạy trên luồng (Thread) của tiến trình chính. Có nghĩa là nếu service của bạn dự định chạy các hành vi có cường độ lớn hoặc các hành vi có thể gây nghẽn mạch, nó sẽ tự sinh ra luồng (Thread) của chính nó để làm thực hiện các tác vụ được đặc tả trong Service.

Cần lưu ý:

Service không phải là một tiến trình độc lập. Đối tượng Service không chạy trên tiến trình (process) của riêng nó (trừ khi có một yêu cầu đặt biệt), Service chạy trên tiến trình của chương trình.

Service không phải là một luồng (thread).

Service có chứa 2 đặc trưng cơ bản sau:

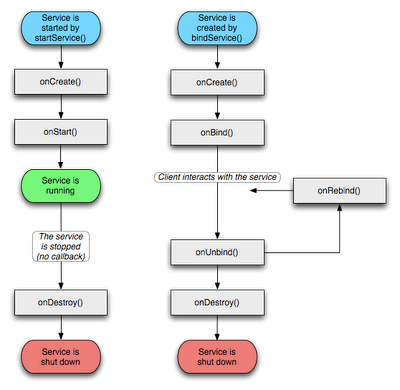
Là một chức năng để ứng dụng thông báo với hệ thống về việc mà nó muốn thực hiện ở phía dưới nền (ngay cả khi người dùng không tương tác trực tiếp tới ứng dụng). Điều này tương ứng với việc gọi phương thức Context.startservice(), nó sẽ yêu cầu hệ thống lên lịch cho service để chạy cho đến khi bị chính service hay người nào đó dừng nó lại.

Là một chức năng cho phép ứng dụng để cung cấp các chức năng của nó cho các ứng dụng khác. Điều đó tương ứng với việc gọi Context.bindService(), cho phép một long-standing connection được tạo ra để service có thể tương tác với nó.

Khi một Service được tạo ra, việc hệ thống cần làm là tạo ra thành phần và gọi hàm onCreate() và bất kỳ lệnh callback thích hợp nào trên luồng chình. Nó phụ thuộc vào Service thi hành hành vi thích hợp nào, chẳng hạn tạo luồng thứ hai để thực hiện tiến trình.

Vòng đời của 1 Service

Có 2 lý do để một service được chạy bởi hệ thống. Bằng cách gọi phương thức Context.startService() và Context.bindService(). Nếu người dùng gọi phương thức Context.startService() thì hệ thống sẽ nhận về một đối tượng **service** (tạo mới nó và gọi hàm onCreate() nếu cần thiết) và sau đó lại gọi phương thức onStartCommand(Intent, int ,int) với các biến được cung cấp bởi client. Service mới được tạo ra này sẽ chạy cho tới khi có lệnh gọi phương thức Context.stopService() hoặc stopSelf(). Dù cho một service có được start bao nhiêu lần đi nữa thì nó sẽ bị tắt ngay khi phương thức Context.stopService() hoặc phương thức stopSelf() được gọi.Service có thể dùng phương thức stopSelf(int) để đảm bảo rằng service sẽ không được tắt cho đến khi Intent đã khởi tạo nó được xử lý.



Hình 12 - Lược đồ vòng đời của 1 Service

Dựa theo hình trên, 1 service được bắt đầu bởi phương thức startService() là service dùng để thực hiện các tác vụ ngầm dưới nền, service được bắt đầu bởi phương thức bindService() là service được dùng để cung cấp các chức năng cho các chương trình khác.

BroadcastReceiver

Là lớp nhận về các intents được gửi bởi sendBroadcast(). Có thể tạo instance cho lớp này bằng 2 cách: hoặcvới Context.registerReceiver() hay thông qua tag <receiver> trong file AndroidManifest.xml.

Có hai lớp broadcast quan trọng cần xét đến:

Broadcasts bình thường (Normal broadcasts): Được gửi bởi context.sendBroadcast tới các receivers hoàn toàn không theo 1 trật tự nào, dẫn tới đôi khi tồn tại cùng 1 lúc nhiều receivers khác nhau. Tuy nhanh nhưng không thể trao đổi kết quả nhận về giữa các receivers với nhau.

Broadcasts có thứ tự (Ordered broadcasts) - Được gửi bởi Context.sendOrderedBroadcast() tới từng receiver một. Như vậy, kết quả nhận về của 1 receiver có thể trở thành input của 1 receiver khác, hay có thể ngay lập tức dừng việc broadcast tới các receiver khác. Trật tự chạy các receivers được quyết định bởi thuộc tính android:priority trong intent-filter tương ứng.

### Content Provider

Là nơi lưu trữ và cung cấp cách truy cập dữ liệu do các ứng dụng tạo nên. Đây là cách duy nhất mà các ứng dụng có thể chia sẻ dữ liệu của nhau.

Android cung cấp sẵn content providers cho 1 số kiểu dữ liệu thông dụng như âm thanh, video, thông tin sổ điện thoại,... (Xem thêm trong gói android.provider). Người lập trình cũng có thể tự tạo ra các class con (subclass) của Content Provider để lưu trữ kiểu dữ liệu của riêng mình.

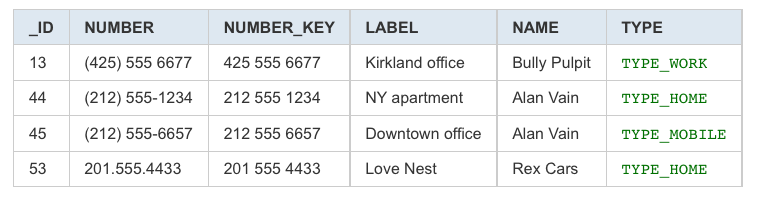
Cách thức hoạt động của Content Provider:

Mọi content provider, tuy có thể khác nhau về cách thức lưu trữ dữ liệu, đều sử dụng chung 1 interface để tìm và trả về dữ liệu, bao gồm việc thêm, bớt và sửa dữ liệu. Việc này được thực hiện thông qua các đối tượng ContentResolver khi gọi getContentResolver()

ContentResolver cr = getContentResolver();

Mô hình dữ liệu

Content providers sử dụng dữ liệu dưới dạng 1 table đơn giản, trong đó mỗi hàng là 1 record và mỗi cột là dữ liệu của 1 kiểu nhất định.



Hình 13 - Ví dụ mô hình dữ liệu thông tin số điện thoại trong máy

Mỗi record có 1 trường “\_ID” độc nhất để liên kết với các bảng khác.

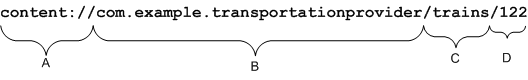
URI (Uniform Resource Identifier)

URI là dãy các ký tự string được dùng để:

Mở 1 trang web trong trình duyệt, bắt đầu với “http://”

Chỉ tới các file tài nguyên có sẵn trong máy (VD: 1 file video, danh sách contacts,...), bắt đầu với "content://"

Dựa vào hình 13 sau:



Hình 14 – Các thành phần của 1 URI

A – Tiền tố mặc định của 1 URI, không thay đổi được.

B – Xác định content provider. Đối với các ứng dụng hãng thứ 3, đây chính là tên đầy đủ của 1 lớp.

C – Xác định loại dữ liệu nào cần lấy về. Nếu nguồn cung cấp chỉ cho lấy 1 loại duy nhất thì có thể để trống, hoặc nếu có nhiều loại thì có thể phân ra từng phần 1 (land/bus, land/train, sea/ship,…)

D – ID của 1 record, khi cần lấy về chỉ duy nhất 1 record.

Khi tạo 1 content provider, Android tự động tạo 1 constant cho URI của provider đó để đảm bảo code hiệu quả và dễ cập nhật sau này. Constant này - CONTENT\_URI - sẽ xác định cho ContentProvider phải liên kết với bảng nào của nguồn cung cấp.

Ví dụ:

URI cho bảng liên kết số điện thoại với tên người là:

android.provider.Contacts.Phones.CONTENT\_URI

URI cho bảng liên kết hình đại diện với tên người là:

android.provider.Contacts.Photos.CONTENT\_URI

Tìm kiếm 1 Content Provider

Để băt đầu tìm kiếm (query) 1 content provider, trước nhất người lập trình cần có trong tay các yếu tố sau:

URI xác định provider.

Tên các trường (field) dữ liệu sẽ lấy về.

Các kiểu dữ liệu của những trường trên.

ID, chỉ khi cần tìm 1 record xác định nào đó.

Để bắt đầu tìm kiếm, người lập trình có thể dùng 1 trong 2 phương thức: ContentResolver.query() hoặc Activity.managedQuery(). Cả 2 phương thức trên đều có chung arguments và đều trả về 1 đối tượng con trỏ (Cursor)

|  |
| --- |
| public final Cursor query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)  public final Cursor managedQuery (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) |

Đầu tiên, dù với query() hay managedQuery() thì ta đều cần cung cấp URI của bên cung cấp, tức CONTENT\_URI. Chú ý nếu muốn giới hạn chỉ tìm kiếm 1 URI duy nhất, ta thêm ID vào cuốiURI như sau: “content://.../23”, với 23 là ID ta cần tím kiếm

Hoặc ta có thể dùng các hàm hỗ trợ như ContentUris.withAppended() và Uri.withAppendedPath().

Ví dụ:

|  |
| --- |
| Uri myPerson = ContentUris.withAppendedId(People.CONTENT\_URI,23); |

Các biến truyền vào khác của query() và managedQuery() là:

projection - Tên của cột dữ liệu cần trả về. Nếu NULL thì trả về tất cả các cột.

selection - Tên các hàng dữ liệu sẽ trả về. Sử dụng với WHERE giống như khi truy vấn SQL. Nếu NULL thì trả về tất cả các hàng.

selectionArgs - quy định giá trị trả về,VD: selectionArgs = {value\_as\_string} sẽ ép kiểu dữ liệu trả về dưới dạng string.

sortOrder - Xác định thứ tự sắp xếp các hàng. Sử dụng với ORDER BY giống như khi truy vấn SQL. Nếu NULL thì không theo thứ tự gì cả.

Tạo mới 1 Content provider

Để tạo mới 1 content provider, cần phải thoả mãn các điều kiện:

Thiết lập hệ thống lưu trữ dữ liệu như thế nào: bằng các file hay sử dụng cơ sở dữ liệu SQLite.

Extend lớp ContentProvider.

Khai báo 1 content provider trong file AndroidManifest.xml

Để định nghĩa 1 lớp con ContentProvider, ta cần hiện thực 6 phương thức trừu tượng (abstract) được khai báo trong lớp ContentProvider:

query()

insert()

update()

delete()

getType()

onCreate()

Phương thức query() phải trả về 1 đối tượng Cursor. Mặc dù Cursor về cơ bản là 1 interface, Android cung cấp sẵn 1 số đối tượng Cursor tạo sẵn. Ví dụ; SQLiteCursor dành cho cơ sở dữ liệu SQLite, MatrixCursor cho dữ liệu không lưu trong SQLite.

Android cũng khuyên nên gọi ContentResover.notifyChange() để báo cho các listeners biếtdữ liệu sẽ được hiệu chỉnh. Ngoài ra người lập trình cũng nên thêm vài thao tác sau để giúp việc truy xuất các lớp hiệu quả hơn:

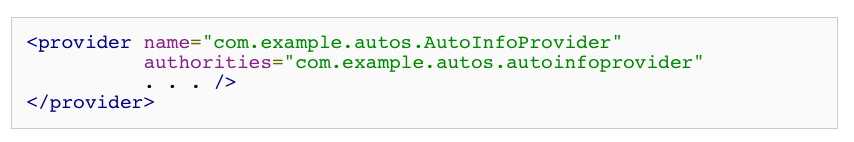
Định nghĩa một Uri dạng public static final, với tên là CONTENT\_URI (như đã trình bày ở trên).

Định nghĩa tên các cột mà content provider sẽ trả về. Thôg thường thì đây cũng chính là tên các cột trong cơ sở dữ liệu SQLite mà chúng đại diện. Tiếp theo, người lập trình nên định nghĩa các public static String constants để các ứng dụng khác có thể ghi rõ tên các cột và các hướng dẫn khác 1 cột interger tên "\_id" (với constant \_ID) cũng cần được định nghia để sử dụng khi cần lấy về các record độc nhất. Trong SQLite, trường \_ID nên có kiểu như sau:INTEGER PRIMARY KEY AUTOINCREMENT, trong đó thành phần AUTOINCREMENT không nhất thiết phải có, nhưng nên được đưa vào để đảm bảo các ID luôn độc nhất, tránh trường hợp khi xóa 1 row thì ID của row tiếp theo sẽ được cấp ID giống hệ của row vừa bị xóa.

Các ghi chú giúp người khác ghi đọc biết đọc dữ liệu như thế nào.

Nếu content provider này xử lý 1 kiểu dữ liệu hoàn toàn mới, người lập trình cần định nghĩ 1 kiểu MIME mới để trả về trong sự thể hiện của ContentProvider.getType().

Để khai báo 1 Content Provider mới, trước nhất cần phải khai báo trong tag <provider> của file AndroidManifest.xml.



Hình 15 - Khai báo tag <provider> trong file AndroidManifest.xml

Trong đó, thuộc tính name là tên đầy đủ của lớp con ContentProvider, và authorities là phần xác định provider của URI "content:"

### View

View là thành phần cơ bản để xây dựng giao diện người dùng cho 1 ứng dụng Android. View là 1 lớp căn bản của widgets (widgets được dùng để tạo các nút nhấn, text fields,...).

Lớp con ViewGroup là lớp căn bản của layouts, có thể coi như 1 cái hộp vô hình chứa nhiều Views hay ViewGroups khác và xác định các thuộc tính layout.

Sử dụng Views:

Trong 1 cửa sổ màn hình, mọi views đều được sắp xếp theo thứ tự trên 1 cây trong 1 hay nhiều file XML nằm trong res/layout.

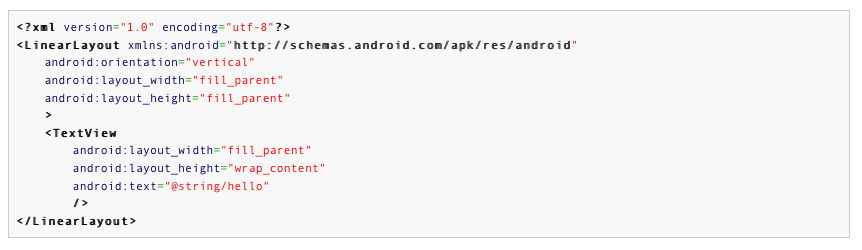
Khi tạo 1 cây chứa các views, người lập trình cần xác định những giá trị sau:

Các thuộc tính - Chẳng hạn định xem 1 TextView sẽ hiển thị chữ gì. Mỗi view có những thuộc tính và phương thức khác nhau.

Mức độ tập trung (Focus) - Thay đổi sự tập trung tuỳ theo input của người dùng với requestFocus().

Listeners - Định các listeners để có hành động khi 1 sự kiện gì đó xảy ra với setOnFocusChangeListener(View.OnFocusChangeListener)

Nhìn thấy hay không - với setVisibility(int)



Hình 16 - Ví dụ 1 cây Views

Một số Views thường dùng:

**TextView:** Hiển thị 1 khung text và cho phép người dùng thay đổi.

**ImageView:** Hiển thị 1 hình ảnh xác định từ file tài nguyên hay qua 1 content provider.

**Button:** Hiển thị 1 nút nhấn.

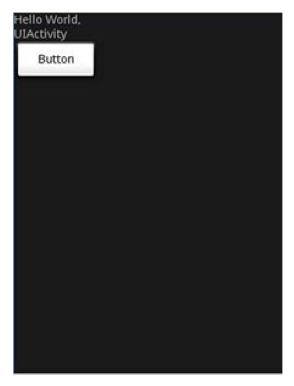
**Checkbox:** Hiển thị 1 nút nhấn với 2 trạng thái khác nhau.

**KeyboardView:** Hiển thị bàn phím ảo để nhập liệu.

**WebView:** Hiển thị các trang web bằng Webkit.

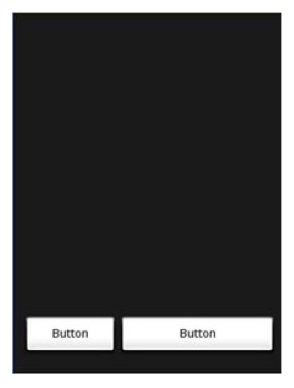
Một số ViewGroups thường dùng:

**LinearLayout:** Các view được xếp theo 1 hàng hay 1 cột duy nhất.



Hình 17 - LinearLayout

**AbsoluteLayout:** Cho phép xác định chính xác vị trí của từng view.



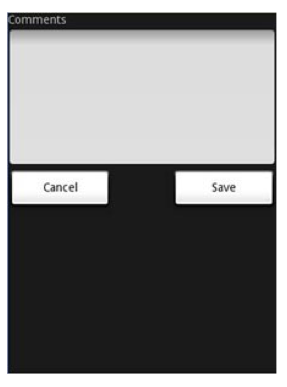
Hình 18 - AbsoluteLayout

**TableLayout:** Sắp xếp các view theo các cột và hàng. Mỗi hàng có thể có nhiều view, mà mỗi view sẽ tạo nên 1 ô.



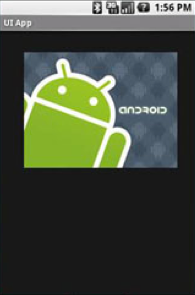
Hình 19 - TableLayout

**RelativeLayout:** Cho phép xác định vị trí các view theo mối quan hệ giữa chúng (VD; canh trái, phải,...)



Hình 20 - RelativeLayout

**FrameLayout:** Là 1 placeholder cho phép đặt lên đó 1 view duy nhất. View đặt lên FrameLayout luôn được canh lề trái phía trên.



Hình 21 - FrameLayout

**ScrollView:** Là 1 FrameLayout đặc biệt cho phép trượt (scroll) 1 danh sách dài hơn kích thước màn hình.



Hình 22 - ScrollView

### Lưu trữ dữ liệu

Android có nhiều cách để lưu trữ dữ liệu. Một cách đơn giản nhất là sử dụng hệ thống preferences. Android cho phép các activity và ứng dụng lưu giữ preferences theo dạng cặp từ khóa và giá trị, nằm giữa các lần gọi một activity. Mục đích chính là để lưu trữ thông tin cấu hình chi tiết mà người dùng hiệu chỉnh, chẳng hạn trang web người dùng xem gần đây nhất. Bạn có thể lưu trữ bất kì thông tin nào trong preferences bằng từ khóa có kiểu String và kiểu dữ liệu nguyên thủy (boolean, String…)

Preferences có thể dùng cho activity độc lập hay được chia sẻ giữa các activity của một ứng dụng. Và tương lai, Preference có thể được chia sẻ giữa các ứng dụng với nhau.

Bên cạnh việc sử dụng Preferences để lưu trữ dữ liệu thì cũng còn một cách đơn giản khác để thực hiện công việc lưu trữ dữ liệu là sử dụng tập tin. Android cung cấp 2 mô hình để truy cập tập tin:

Một cho các tập tin được đóng gói sẵn cho ứng dụng.

Một dành cho tập tin được tạo trên thiết bị thực thi ứng dụng đó.

Đối với dữ liệu tĩnh, chẳng hạn danh sách các từ dùng để kiểm tra chính tả. Cách đơn giản nhất để thực hiện điều đó là lưu vào một tập tin, và tập tin này phải được đặt trong thư mục res/raw của project. Nhờ vậy, tập tin sẽ được đặt trong tập tin APK của ứng dụng như một nguồn tài nguên cơ bản.

Ngoài 2 phương pháp trên, Android còn hỗ trợ hệ quản trị cơ sở dữ liệu quan hệ SQLite để quản lý dữ liệu lớn và có quan hệ phức tạp. Nhờ vậy, người lập trình có thể thực hiện quản lý dữ liệu một cách dễ dàng, linh hoạt hơn.

Hệ quản trị cơ sở dữ liệu SQLite

SQLite là gì

Khi lập trình trên di động hay các thiết bị có dung lượng bộ nhớ hạn chế, người ta thường dùng SQLite. SQLite là một hệ quản trị cơ sở dữ liệu nhúng được hiện thực từ chuẩn SQL-92. Giống với cái tên của nó, SQLite chiếm dung lượng nhỏ (khoảng 275KB) nên việc truy xuất dữ liệu được nhanh chóng, không chiếm dụng quá nhiều tài nguyên hệ thống. Do SQLIte là phần mềm mã nguồn mở nên nó không bị giới hạn tác quyền. Vì lý do đó mà SQLite được nhiều hãng sử dụng (Adobe, Apple, Google, Sun, Symbian) và các dự án mã nguồn mở (Mozila, PHP, Python).

Đặc biệt, đối với Android, SQLite rất thích hợp để tạo cơ sở dữ liệu cho các ứng dụng trên nền Android. Từ khi SQLite có giao diện SQL thì nó càng được sử dụng dễ dàng và thông dụng hơn.

Sqlite tuy nhẹ hơn so với các hệ cơ sở dữ liệu khác nhưng cũng không khác biệt nhiều. SQLite cũng sử dụng ngôn ngữ truy vấn SQL (SELECT, INSERT, DELETE…), thích hợp cho các ứng dụng trên điện thoại, đặc biệt là ứng dụng trên hệ điều hành Android vì Android Runtime có thể chứa cả SQLite mà không cần phải chia nhỏ ra.

SQLite có một hệ thống câulệnhSQL đầy đủ với các triggers, transactions... Các câu truy vấn cũng như các hệ cơ sở dữ liệu khác. SQLite như bản thu nhỏ của so với các hệ CSDL khác, vì vậy nó không thể có đầy đủ các chứ năng trên chiếc điện thoại di động của bạn.

SQLite là một lựa chọn thích hợp dành cho ứng dụng trên hệ điều hành Android. Ngoài dung lượng lưu trữ nhỏ gọn, SQLite còn cho phép sử dụng Unicode, kiểu dữ liệu không được cài đặt trong một số phiên bản Android.

Đặc trưng của SQLite

SQLite được hiện thực từ tiêu chuẩn SQL-92 của một ngôn ngữ SQL nhưng vẫn còn chứa một số khiếm khuyết.

Tuy SQLite hỗ trợ triggers nhưng bạn không thể viết trigger cho view. Hoặc SQLite không hỗ trợ lệnh ALTER TABLE, do đó, bạn không thể thực hiện chỉnh sửa hoặc xóa cột trong bảng.

SQLite không hỗ trợ ràng buộc khóa ngoại , các transactions lồng nhau, phép kết right outer join, full outer join.

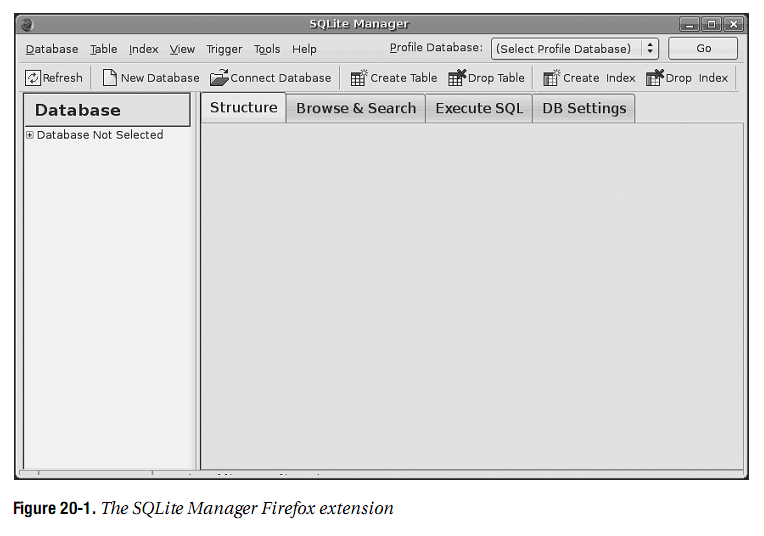
SQLite sử dụng kiểu dữ liệu khác biệt so với hệ quản trị cơ sở dữ liệu tương ứng. Bạn có thể insert dữ liệu kiểu string vào cột kiểu integer mà không gặp phải bất kỳ lỗi nào.

Vài tiến trình hoặc luồng có thể truy cập tới cùng một cơ sở dữ liệu. Việc đọc dữ liệu có thể chạy song song, còn việc ghi dữ liệu thì không được phép chạy đồng thời.

Ngoài các khiếm khuyết trên thì Sqlite cung cấp cho người dùng gần như đầy đủ các chưng năng mà một hệ cơ sở dữ liệu cần có như tạo database; tạo bảng; thêm, xóa, sửa dữ liệu.

SQLite Manager

SQLite Manager là công cụ hiệu quả hỗ trợ cho việc xây dựng cơ sở dữ liệu cho ứng dụng Android đó là SQLite Manager, một add-on miễn phí của Firefox (<https://addons.mozilla.org/en-US/firefox/addon/5817>)



Hình 23 - Màn hình làm việc của addon SQlite Manager

Sử dụng SQLite Manager để quản lí cơ sở dữ liệu Android

Tạo database mới :

Vào menu Database/ New Database

Tạo bảng mới:

Tại Database vừa tạo, nhấp chuột phải chọn Create table. Thiết lập các giá trị thích hợp.

Xử lý dữ liệu:

Chọn bảng cần nhập và chọn tab Browse & Search

Nút Add: Thêm dữ liệu

Nút Duplicate: Nhân đôi dữ liệu

Nút Edit: Chỉnh sửa dữ liệu

Nút Delete: xóa dữ liệu

Thực thi câu truy vấn:

Chọn tab Execute SQL, Nhập câu lệnh SQL vào khung Enter SQL và chọn Run SQL.

Kết quả sẽ được trình bày ở khung phía dưới. Nếu câu SQL có lỗi sẽ được hiển thị tại ô Last Error.

Tùy chỉnh Database: Chọn tab DB Settings và thiết lập các giá trị như mong muốn

**Import Database vào ứng dụng Android**

Sau khi tạo CSDL bằng SQLite Manager,chọn menu File/Export Database. Lưu file .sql vừa tạo vào máy.

Để import database vào ứng dụng, người lập trình cần :

Build and Run chương trình Android.

Khởi động máy ảo xong, chọn nút DDMS ở góc trên bên phải chương trình Eclipse để xem cấu trúc file.

Vào thư mục chứa database có dường dẫn như sau :  
/data/data/your.app.package/databases/your-db-name

Sau đó chọn nút import và chọn tập tin database. Lưu ý : Phải đổi tên file từ \*.sql thành \* trước khi import vào chương trình.