

Lời nói đầu

Giáo trình này được soạn thảo dùng để hỗ trợ giảng dạy cho môn Mobile Programming, tại Đại học FPT Thành phố Hồ Chí Minh. Giáo trình chỉ thảo luận các vấn đề lập trình trên Android, tuy nhiên do mức độ cập nhật rất nhanh của môn học, chúng tôi đã phải cố gắng rất nhiều để giáo trình trở nên dễ hiểu, chi tiết và sát với thực tế.

Chúng tôi sẽ thường xuyên cập nhật và hiệu chỉnh giáo trình, kịp thời với những thay đổi nhanh chóng của công nghệ.

Tôi xin tri ân đến các bà đã nuôi dạy tôi, các thầy cô đã tận tâm chỉ dạy tôi. Chỉ có cống hiến hết mình cho tri thức tôi mới thấy mình đền đáp được công ơn đó.

Tôi xin cảm ơn gia đình đã hy sinh rất nhiều để tôi có được khoảng thời gian cần thiết thực hiện được giáo trình này.

Mặc dù đã dành rất nhiều thời gian và công sức, hiệu chỉnh chi tiết và nhiều lần, nhưng giáo trình không thể nào tránh được những sai sót và hạn chế. Tôi thật sự mong nhận được các ý kiến góp ý từ bạn đọc để giáo trình có thể hoàn thiện hơn. Nội dung giáo trình sẽ được cập nhật định kỳ, mở rộng và viết chi tiết hơn; tùy thuộc những phản hồi, những đề nghị, những ý kiến đóng góp từ bạn đọc.

Các bạn đồng nghiệp nếu có sử dụng giáo trình này, xin gửi cho tôi ý kiến đóng góp phản hồi, giúp giáo trình được hoàn thiện thêm, phục vụ cho công tác giảng dạy chung.

Phiên bản

Cập nhật ngày: 10/05/2015

Thông tin liên lạc

Mọi ý kiến và câu hỏi có liên quan xin vui lòng gửi về:

Dương Thiên Tứ

91/29 Trần Tấn, P. Tân Sơn Nhì, Q. Tân Phú, Thành phố Hồ Chí Minh

Facebook: <https://www.facebook.com/tu.duongthien>

E-mail: thientu2000@yahoo.com

Android Platform

Hệ điều hành Android

Android là hệ điều hành mã nguồn mở dựa trên Linux, được thiết kế cho các thiết bị di động như điện thoại thông minh và máy tính bảng. Android được phát triển bởi liên minh OHA (Open Handset Alliance, 11/2007, <http://openhandsalliance.com>), dẫn đầu bởi Google. Android cung cấp một phương pháp tiếp cận thống nhất để phát triển ứng dụng cho các thiết bị di động, có nghĩa là các nhà phát triển chỉ cần phát triển cho Android, và các ứng dụng của họ sẽ có thể chạy trên các thiết bị khác nhau được hỗ trợ bởi Android.

Phiên bản beta đầu tiên của Android Software Development Kit (SDK) được phát hành bởi Google vào năm 2007 trong khi đó phiên bản thương mại đầu tiên, Android 1.0, được phát hành vào tháng 9 năm 2008. Ngày 15 tháng 10 năm 2014, Google công bố phiên bản Android 5.0 Lollipop.

Mã nguồn của Android được phát hành theo giấy phép phần mềm miễn phí mã nguồn mở. Google công bố hầu hết mã nguồn theo Giấy phép Apache phiên bản 2.0 và phần còn lại, những thay đổi phần nhân Linux, theo Giấy phép GNU General Public phiên bản 2.

Một vài tính năng của Android được liệt kê dưới đây:

Tính năng	Mô tả
Giao diện người dùng	Màn hình cơ bản của hệ điều hành Android cung cấp một giao diện người dùng đẹp và trực quan.
Kết nối	Hỗ trợ nhiều giao thức kết nối như GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (A2DP và AVRCP), Wi-Fi, LTE, NFC và WiMAX.
Lưu trữ	Sử dụng SQLite, một hệ cơ sở dữ liệu quan hệ gọn nhẹ, được sử dụng với mục đích lưu trữ dữ liệu có cấu trúc.
Đa phương tiện	Hỗ trợ nhiều giao thức truyền thông đa phương tiện và định dạng đa phương tiện như H.263, H.264 AVC, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, và BMP.
Truyền thông điệp	Hỗ trợ SMS và MMS.
Trình duyệt Web	Trình duyệt tích hợp dựa trên engine mã nguồn mở WebKit, kết hợp công cụ JavaScript v8 của Chrome, hỗ trợ cả HTML5 và CSS3.
Cảm ứng đa điểm	Android có hỗ trợ cảm ứng đa điểm mà ban đầu đã được tạo sẵn trong thiết bị cầm tay chẳng hạn như HTC Hero.
Đa tác vụ	Người dùng có thể chuyển từ tác vụ này sang tác vụ khác và nhiều ứng dụng khác nhau có thể chạy cùng một lúc.
Widget thay đổi kích thước	Các widget có thể thay đổi kích thước, người dùng có thể kéo giãn chúng để hiển thị nội dung nhiều hơn hoặc thu nhỏ chúng để tiết kiệm không gian.
Đa ngôn ngữ	Hỗ trợ văn bản đơn hướng và văn bản hai chiều.
GCM	Google Cloud Messaging (GCM) là một dịch vụ cho phép các nhà phát triển gửi thông điệp dữ liệu ngắn cho người dùng trên các thiết bị Android, mà không cần một giải pháp đồng bộ độc quyền.
Wi-Fi Direct	Công nghệ cho phép các ứng dụng dò tìm và bắt cặp trực tiếp, trên một kết nối peer-to-peer bằng thông cao.
Android Beam	Công nghệ dựa trên NFC ¹ , cho phép người dùng ngay lập tức chia sẻ thông tin, chỉ cần chạm hai điện thoại kích hoạt NFC với nhau.
Lập trình	Cung cấp Android Studio, là môi trường phát triển ứng dụng phong phú: giả lập thiết bị, debug, theo dõi bộ nhớ và hiệu suất hoạt động.

Các ứng dụng Android thường được phát triển bằng ngôn ngữ Java sử dụng Android Software Development Kit (Android SDK). Sau khi phát triển, các ứng dụng Android có thể được đóng gói và phát hành thông qua cửa hàng trực tuyến như Google Play hoặc Amazon Appstore.

Android có mặt trên hàng trăm triệu thiết bị di động tại hơn 190 quốc gia trên thế giới. 500 triệu thiết bị Android được kích hoạt kể từ khi Android được chính thức phát hành năm 2008. Mỗi ngày, hơn 1.3 triệu thiết bị Android mới được kích hoạt trên toàn thế giới.

Kiến trúc của Android

Hệ điều hành Android là một ngăn xếp các đơn thể phần mềm (software stack) được tạm chia thành năm phần và bốn lớp chính. Liệt kê từ dưới lên như sau:

1. Linux kernel

Lớp dưới cùng là nhân Linux đã tùy biến phù hợp với thiết bị di động. Chúng cung cấp các chức năng cơ bản của hệ điều hành như quản lý tiến trình, quản lý bộ nhớ, quản lý thiết bị (máy ảnh, bàn phím, màn hình hiển thị), mạng và bảo mật, v.v... Các chức năng sau, tích hợp trong nhân Linux, là đặc thù trên Android:

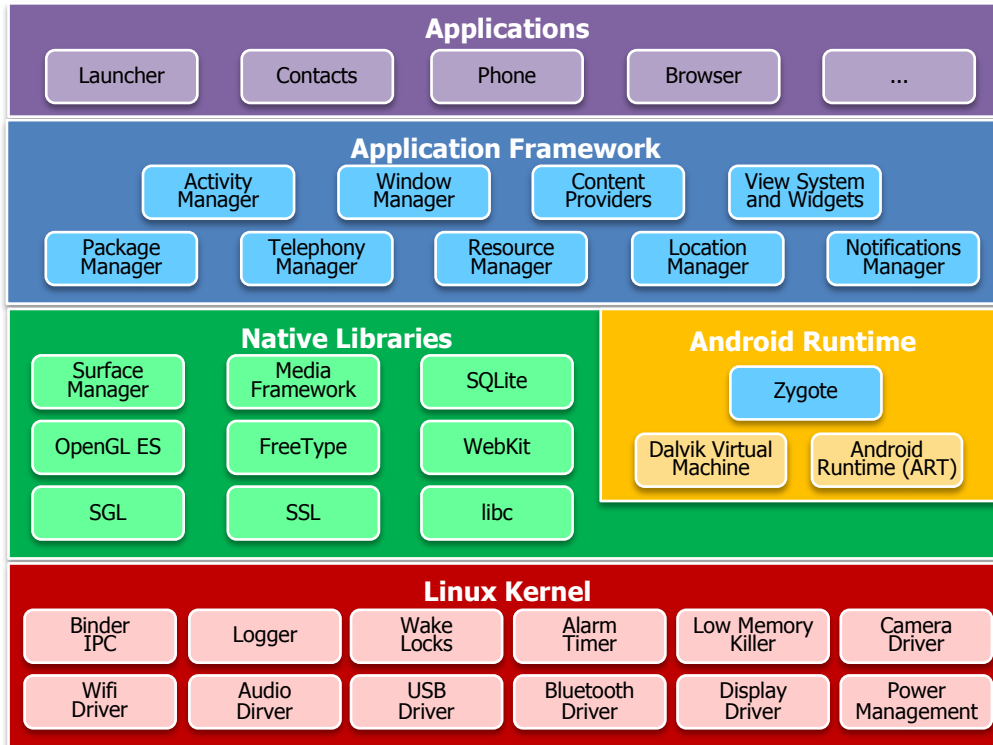
- Binder: cơ chế IPC (Inter-Process Communication) thiết kế riêng cho Android. Binder hoạt động như kênh truyền thông trung tâm cho phép ứng dụng giao tiếp với hệ thống, phone, dịch vụ.
- Logger: Android cung cấp dịch vụ ghi nhận (logging) tập trung toàn hệ thống, có thể tổng hợp các ghi nhận từ hệ thống cũng như từ các ứng dụng, cung cấp cái nhìn tổng thể để hỗ trợ xử lý sự cố.
- Power management: Android được thiết kế để hoạt động trên các thiết bị di động với tài nguyên hạn chế. Nguồn pin là một trong những tài nguyên quan trọng nhất. Vì vậy, thiết bị Android thường chuyển sang chế độ ngủ (sleep mode) sử dụng ít pin. Tuy nhiên, wake locks cũng được đưa vào nhân Linux để kích hoạt ứng dụng ngăn thiết bị chuyển sang chế độ ngủ khi thực

¹ Near-Field Communications, công nghệ giao tiếp tầm ngắn, dựa trên RFID (Radio-Frequency IDentification), là công nghệ kết nối không dây giữa các thiết bị khi có sự tiếp xúc trực tiếp hay để gần nhau. Thường dùng để giao tiếp giữa thiết bị với thẻ thông minh, đầu đọc thẻ hoặc thiết bị NFC tương thích khác.

hiện các tác vụ quan trọng. Ngoài ra, nhân Linux cũng có alarm timer, cho phép đánh thức thiết bị khỏi chế độ ngủ để thực hiện các tác vụ được lập lịch trước.

- Low memory killer: việc nạp ứng dụng vào bộ nhớ tốn nhiều chi phí nên Android giữ các ứng dụng đang chạy trong bộ nhớ ngay cả khi người dùng không còn tương tác với chúng. Điều này cho phép chuyển nhanh giữa các ứng dụng đang chạy nhưng lại có nhược điểm là chiếm nhiều tài nguyên bộ nhớ. Khi bộ nhớ giảm xuống một ngưỡng nhất định, low memory killer (còn gọi là Viking Killer) sẽ loại bỏ dần các ứng dụng đang chạy kể từ ứng dụng kém quan trọng nhất. Ứng dụng chạy bề mặt quan trọng hơn ứng dụng chạy nền. Trạng thái của ứng dụng chạy nền bị hủy sẽ được lưu và phục hồi khi người dùng quay lại ứng dụng đó.

- Hệ thống tập tin: Android dùng hệ thống tập tin YAFFS2 (Yet Another Flash File System). Để dễ nâng cấp, nó được bố trí trên một số phân vùng (partition) đĩa: boot, system, recovery, data và cache.



Kiến trúc của Android Platform

2. Native Libraries

Trên lớp Linux kernel là lớp chứa một tập hợp các thư viện mà chức năng của chúng bộc lộ thông qua lớp Android Runtime. Một số thư viện đáng chú ý:

- SQLite: hệ cơ sở dữ liệu quan hệ cho phép các ứng dụng dễ dàng lưu trữ và truy cập nhanh chóng dữ liệu của chúng.
- WebKit: engine Web mã nguồn mở, hiển thị HTML/CSS và thực thi JavaScript, cho phép ứng dụng tích hợp công nghệ Web.
- OpenGL ES: cung cấp hỗ trợ đồ họa 2D và 3D.
- Open Core: cung cấp media framework, cho phép ứng dụng ghi và phát audio/video.
- OpenSSL: cung cấp giao thức bảo mật tầng Transport, SSL và TLS, đảm trách truyền thông mạng an toàn.

3. Android Runtime

Phần này cung cấp một thành phần quan trọng được gọi là máy ảo Dalvik. Dan Bornstein thiết kế Dalvik VM như một máy ảo Java được tối ưu hóa đặc biệt để hoạt động trên thiết bị di động. Dalvik VM sử dụng các tính năng của nhân Linux như quản lý bộ nhớ và đa luồng, cũng là tính năng của Java.

Dalvik VM cho phép tất cả các ứng dụng Android chạy trong tiến trình riêng của nó. Java bytecode sẽ được chuyển thành dex bytecode để chạy trên máy ảo Dalvik. Dalvik VM đã được tối ưu sao cho một thiết bị có thể chạy nhiều máy ảo cùng lúc một cách hiệu quả.

Từ Android 4.4, bạn có thể kích hoạt thiết bị để dùng ART thay cho Dalvik VM. ART (Android Run Time) là phiên bản kế tiếp của Dalvik VM, có hai đặc điểm cải tiến so với Dalvik VM:

- Trình biên dịch Ahead-of-Time (AOT) biên dịch ra mã gốc một lần trước khi dùng đến, thay thế cho trình biên dịch Just-in-Time (JIT) của Dalvik VM.
- Gabage Collection (GC) cải tiến.

ART chuyển đổi tự động định dạng .dex của Dalvik VM sang định dạng .oat của ART khi ứng dụng cài đặt trên thiết bị.

Để kích hoạt ART trên máy ảo dùng Android 4.4+, vào Settings > Developer options > Select runtime > Use ART rồi khởi động lại thiết bị.

Android Runtime cũng cung cấp một tập hợp các thư viện lõi cho phép các nhà phát triển ứng dụng Android viết các ứng dụng Android bằng ngôn ngữ lập trình Java chuẩn.

Zygote là cha của tất cả tiến trình ứng dụng trên Android. Nó là một trong những tiến trình lõi bắt đầu khi hệ thống khởi động. Zygote có hai vai trò quan trọng:

- Khi hệ thống khởi động, Zygote khởi động một thực thể VM mới và khởi tạo các dịch vụ Android lõi: Power, Telephony, Content và Package.
- Mỗi ứng dụng chạy trong một VM dành riêng, việc khởi động một VM và nạp các đơn thể ứng dụng vào

bộ nhớ yêu cầu nhiều chi phí. Zygote giải quyết vấn đề này bằng "forking": tiến trình Zygote sẽ tự nhân bản thành tiến trình mới, tiến trình nhân bản cũng dùng chung tài nguyên nạp trước của Android.

4. Application Framework

Lớp Application Framework cung cấp giao diện cho ứng dụng tương tác với Android và thiết bị. Lớp này định nghĩa và quản lý vòng đời của các đơn thể Android và liên lạc giữa các đơn thể đó. Nó cũng định nghĩa một tập các cơ chế bất đồng bộ dành riêng cho Android, giúp ứng dụng đơn giản hóa việc quản lý thread: HandlerThread, AsyncTask, IntentService, AsyncQueryHandler và Loaders.

5. Applications

Tất cả các ứng dụng Android mà người dùng giao tiếp ở lớp trên cùng. Chúng là các ứng dụng tải về từ Android market, ứng dụng hệ thống (Launcher, Contacts, Phone, Browser) và các ứng dụng do bạn viết.

Các ứng dụng Android cực kỳ đa dạng, thuộc rất nhiều thể loại: games (arcade, brain, casual, cards, racing, sports), entertainment, tools, communication, productivity, personalization, music & audio, social, media & video, travel & local.

Hệ thống phiên bản

Android dùng hệ thống ba phiên bản, liên hệ chặt chẽ với nhau: phiên bản của Android SDK, mã phiên bản và API Level.

Phát hành	Phiên bản Android SDK	API Level	Mã phiên bản (codename, theo alphabet)
October 2014	Android 5.0	21	Lollipop
July 2014	Android 4.4w	20	KitKat with Wearable Extensions
October 2013	Android 4.4 - 4.4.4	19	KitKat
July 2013	Android 4.3 - 4.3.1	18	Jelly Bean MR1
November 2012	Android 4.2 - 4.2.2	17	Jelly Bean MR2
July 2012	Android 4.1 - 4.1.2	16	Jelly Bean
December 2011	Android 4.0.3 - 4.0.4	15	Ice Cream Sandwich MR1
October 2011	Android 4.0 - 4.0.2	14	Ice Cream Sandwich
July 2011	Android 3.2 - 3.2.6	13	Honeycomb MR2
May 2011	Android 3.1	12	Honeycomb MR1
February 2011	Android 3.0	11	Honeycomb
February 2011	Android 2.3.3 - 2.3.7	10	Gingerbread MR1
December 2010	Android 2.3 - 2.3.2	9	Gingerbread
May 2010	Android 2.2 - 2.2.3	8	Froyo
January 2010	Android 2.1	7	Eclair MR1
December 2009	Android 2.0.1	6	Eclair 0 1
October 2009	Android 2.0	5	Eclair
September 2009	Android 1.6	4	Donut
April 2009	Android 1.5	3	Cupcake
February 2009	Android 1.1	2	–
September 2008	Android 1.0	1	–

Sự thay đổi nhanh chóng các phiên bản buộc người phát triển liên tục theo dõi, cập nhật thông tin về thiết bị, Android API để có những thay đổi phù hợp.

Khi tạo project mới, bạn được yêu cầu chỉ định `android:minSdkVersion`, là API Level thấp nhất mà ứng dụng của bạn sẽ chạy trên đó. Khi chọn `android:minSdkVersion`, bạn xem xét cân đối giữa % số thiết bị có thể cài đặt ứng dụng và các đặc điểm của ứng dụng mà bạn muốn API Level hỗ trợ. Android Studio sẽ đề nghị giúp bạn trị `android:minSdkVersion` bằng cách tham khảo biểu đồ phân phối phiên bản Android tại: <https://developer.android.com/about/dashboards/index.html>

Các tính năng của ứng dụng đang xây dựng phải được hỗ trợ trên phiên bản `android:minSdkVersion`. Nếu bạn chọn API Level quá thấp, chương trình sẽ cảnh báo và bạn phải điều chỉnh API Level cho trị `android:minSdkVersion` trong tập tin manifest để phù hợp với tính năng đang sử dụng. Nếu bạn muốn hỗ trợ các phiên bản cũ bạn phải gọi các lớp trong thư viện hỗ trợ `android.support`, tuy nhiên thư viện hỗ trợ không hỗ trợ *tất cả* API mới.

API Level bạn thiết lập cho máy ảo hoặc của máy thật phải lớn hơn `android:targetSdkVersion`. Nếu không phù hợp, khi chạy ứng dụng, cửa sổ Choose Device sẽ không có thiết bị mục tiêu hiển thị để bạn chọn.

Cộng đồng phát triển

Nếu bạn gặp vấn đề khi phát triển ứng dụng trên Android, bạn sẽ nhận được hỗ trợ rất tốt khi tham khảo các site sau:

- Stack Overflow (www.stackoverflow.com)
- Google Android Training (<http://developer.android.com/training/index.html>)
- Android Discuss (<http://groups.google.com/group/android-discuss>)
- Android Source Code (<http://source.android.com>)

Môi trường phát triển

Android có một môi trường phát triển toàn diện, có thể chạy trên các hệ điều hành phổ biến. Do Android phát triển rất nhanh, yêu cầu về phiên bản trong phần này có thể thay đổi, bạn cần thường xuyên kiểm tra để cập nhật thông tin mới.

Android Toolchain

Dự án mã nguồn mở Android (AOSP - Android Open Source Project) cung cấp một chuỗi công cụ (toolchain) cho các nhà phát triển ứng dụng. AOSP xây dựng toàn bộ môi trường phát triển bằng cách tích hợp một cách cẩn thận các công cụ phát triển mã nguồn mở được lựa chọn hiện có, để tạo ra một toolchain toàn diện. Cách tiếp cận này giúp cho các nhà phát triển ứng dụng phát triển các ứng dụng di động bằng cách sử dụng các công cụ mà họ đã quen thuộc.

Android toolchain hình thành bởi bốn thành phần chính.

1. Android Software Development Kit

Android Software Development Kit (Android SDK) là đơn thể chính của Android toolchain. Nó cung cấp:

- Các thư viện Java API cho hệ nền.
- Công cụ đóng gói ứng dụng.
- Công cụ giả lập thiết bị.
- Công cụ tối ưu hóa và làm rối (obfuscator) bytecode.
- Cầu nối debug cho Android (ADB – Android Debug Bridge).
- Code mẫu và hướng dẫn.
- Tài liệu cho hệ nền.

2. Android Native Development Kit

Android dựa trên nhân Linux để cung cấp các chức năng của hệ điều hành. Sự kết hợp giữa nhân Linux và các thư viện BSD C cung cấp tất cả các thành phần cần thiết để thực hiện các ứng dụng không phải bằng Java và phụ thuộc hệ nền. Điều này cho phép tạo ứng dụng từ các ngôn ngữ lập trình sinh mã máy như C, C++ và hợp ngữ.

Android Native Development Kit (NDK) cung cấp tập các công cụ cho Android SDK, được thiết kế để cho phép các nhà phát triển xây dựng và nhúng mã từ ngôn ngữ gốc và các ứng dụng dựa trên Java.

3. Android Development Tools trên Eclipse

Android Development Tools trên Eclipse (Eclipse ADT) là nỗ lực đầu tiên để cung cấp một môi trường phát triển tích hợp (IDE) được thiết kế riêng cho phát triển ứng dụng Android. Giống các đơn thể khác của Android toolchain, điều này được thực hiện bằng cách cung cấp các tùy chỉnh và các tính năng bổ sung trên nền Eclipse IDE có sẵn.

4. Android Studio

Tháng 5/2013, Google công bố Android Studio, một IDE mới để phát triển ứng dụng Android, được tùy chỉnh dựa trên IntelliJ IDEA. Android Studio cung cấp một số tính năng bổ sung và cải tiến hơn Eclipse ADT.

Tháng 12/2014, Android Studio phiên bản 1.0 được phát hành. Android Studio được xem như Android IDE chính thức, thay thế cho Eclipse ADT mà Google đã loại bỏ link tải về từ 12/2014. Android Studio có sẵn cho các hệ điều hành phổ biến, như Windows, Mac OS X và Linux.

Để phù hợp với thực tế, trong tài liệu này chúng tôi giả định bạn đang dùng Android Studio trên Windows 7.

Thiết lập môi trường phát triển

1. Java Development Kit (JDK)

Yêu cầu tiên quyết cho Android Studio là Java Development Kit (JDK) và Java Runtime Environment (JRE). Android toolchain hỗ trợ nhiều loại JDK và JRE, nhưng chúng tôi giả định bạn dùng Oracle JDK. Android toolchain hỗ trợ Java phiên bản 6 và 7, chúng tôi đề nghị bạn dùng phiên bản 7, vì vào thời điểm phát hành tài liệu này, Android chưa hỗ trợ hoàn toàn Java 8.

Bạn có thể tải về phiên bản JDK 7 từ trang của Oracle: www.oracle.com/technetwork/java/javase/downloads/index.html

Bạn sẽ tìm thấy hướng dẫn cài đặt JDK trong tập tin tải về, hãy làm theo các hướng dẫn để cài đặt và cấu hình các thiết lập.

Sau đó, thiết lập các biến môi trường PATH và JAVA_HOME để tham chiếu đến các thư mục có chứa java và javac, thường là thư mục cài đặt JDK\bin và thư mục cài đặt JDK tương ứng.

Ví dụ, bạn cài đặt JDK trong C:\j2se7. Click phải lên My Computer, chọn Properties > Advanced > Environment Variables. Đặt biến môi trường JAVA_HOME là C:\j2se7 và thêm vào sau biến môi trường PATH có sẵn C:\j2se7\bin.

2. Android Studio

Vào trang cung cấp Android Studio: <http://developer.android.com/sdk/installing/studio.html>. Trang web sẽ xác định hệ điều hành và hiển thị nút cho phép tải về phiên bản phù hợp hệ điều hành của bạn. Chuyển đến phần All Android Studio Packages, tải về gói android-studio-bundle.

Thực hiện các bước cài đặt theo hướng dẫn, đặc biệt chú ý vị trí (thư mục) cài đặt Android Studio và Android SDK. Bạn không nên cài đặt theo mặc định mà nên chọn đường dẫn đơn giản, không có khoảng trắng, để truy cập. Ví dụ, cài đặt Android Studio trong D:\android_studio và Android sdk trong D:\android_sdk. Các bước cài đặt:

- Choose Components, cho phép chọn:

- + Android Studio, môi trường phát triển tích hợp các ứng dụng Android.
 - + Android SDK, đơn thể quan trọng nhất trong Android toolchain.
 - + Android Virtual Device (AVD), thiết bị Android ảo, được cấu hình trước và tối ưu để kiểm thử ứng dụng.
 - + Performance (Intel® HAXM), Intel HAXM (Intel Hardware Accelerated Execution Manager) là engine ảo hóa hỗ trợ bởi phần cứng để tăng tốc emulator chạy AVD. Bạn cũng có thể tải từ <https://software.intel.com>.
- Intel HAXM dùng khi máy tính của bạn có CPU hỗ trợ Intel Virtualization Technology (VT-x). Chắc rằng tính năng VT-x đã được kích hoạt: cấu hình Intel Virtual Technology trong BIOS phải được Enabled và Hyper-V (nếu có) phải được tắt.

- Configuration Settings – Install Locations, chỉ định vị trí lưu trữ Android Studio và Android SDK.
- Configuration Settings – Emulator Setup, thiết lập số RAM cho Intel HAXM.

Khi chạy Android Studio lần đầu, nó giãn nén và cài đặt Android SDK Tools kèm theo. Sau đó, khi Android Studio chạy Android Setup Wizard, bạn có thể cài đặt bổ sung Android SDK bằng cách vào Quick Start > Configure > SDK Manager (hoặc chạy SDK Manager .exe trong thư mục cài đặt Android SDK).

Cửa sổ Android SDK Manager sẽ liệt kê một số gói đã cài đặt, bạn có thể chọn để cài đặt thêm.

- SDK Platform cho các phiên bản bạn cần phát triển, là mục tùy chọn Target Name cho các AVD sẽ tạo.
- Intel x86 Atom System Image tương ứng, là mục tùy chọn CPU/ABI cho các AVD sẽ tạo.
- Trong phần Extras, nên chọn cài đặt đủ các gói: Android Support Library, Google Play services, Google USB Driver, Intel x86 Emulator Accelerator (HAXM installer).

Do HAXM installer chỉ tự sao chép vào <Android SDK>/extras/intel/Hardware_Accelerated_Execution_Manager nhưng vẫn chưa cài đặt, bạn phải tự cài đặt HAXM.

Chọn các gói, click nút Install xx packages... (xx là số các gói đề nghị cài đặt), chọn Accept License, click nút Install. Quá trình cài đặt thông qua Internet, có thể mất khá nhiều thời gian và chiếm nhiều dung lượng đĩa.

Nếu bạn cài đặt mặc định và không nhớ đường dẫn chỉ đến Android SDK, khởi động Android SDK Manager từ Android Studio và tìm thấy đường dẫn chỉ đến Android SDK ngay trên đầu cửa sổ. Bạn có thể đưa đường dẫn này vào biến môi trường Path để gọi trực tiếp các công cụ trong Android SDK.



Android Studio là một IDE phức tạp, để sử dụng hiệu quả xin xem phần phụ lục – Android Studio.

Thiết bị Android ảo

Chạy ứng dụng trên thiết bị ảo không thể nào so sánh với chạy trên thiết bị thật. Tuy nhiên, để kiểm thử ứng dụng trên một số lớn chủng loại thiết bị Android khác nhau, cài đặt các phiên bản Android khác nhau, cách tốt nhất vẫn là dùng thiết bị Android ảo. Phần này trình bày cách làm chủ AVD và giới thiệu một số giải pháp thiết bị Android ảo có thể sử dụng thay thế AVD.

1. Các thiết bị ảo

a) Android Virtual Device (AVD)

Để kiểm thử ứng dụng Android của bạn, bạn sẽ cần một thiết bị Android ảo. Vì vậy, trước khi bắt đầu viết mã, cần tạo ra một thiết bị Android ảo. Chạy SDK Manager bằng cách chọn icon  trên toolbar, rồi vào Tools > Manage AVDs... hoặc trong Android Studio, vào Tools > Android > AVD Manager, hoặc chọn icon  trên toolbar.


Trong Android Virtual Device Manager, click nút Create Virtual Device... Trong hộp thoại Virtual Device Configuration:

- Select Hardware: chọn phân loại (category) và thiết bị, bạn nên chọn Google Nexus (thiết bị "thuần" Android).

Nhấn nút Clone Device... để nhân bản thiết bị được chọn thành AVD của bạn, cấu hình cơ bản rồi nhấn OK để quay lại.

Chọn thiết bị mới vừa tạo, nhấn Next.

- System Image: chọn một trong những system image bạn đã tải về, hoặc tải về system image bạn chọn, bằng cách click Download trong cột Release Name, thường chọn x86. Nhấn Next.

Sau khi xác nhận cấu hình (bước Verify Configuration), chọn AVD trong danh sách Your Virtual Devices và click nút  để chạy thử AVD.

Một số phím tắt của AVD:

Home	nút Home, chuyển đến màn hình Home.
Esc	nút Back, quay lại màn hình trước.
F2	bật/tắt context sensitive menu.
F3/F4	bắt đầu/kết thúc gọi.
F5	mở search.
F6	bật/tắt chế độ dùng trackball.
F7	bật/tắt nút Power.
F8	bật/tắt mạng dữ liệu (3G)
Ctrl+F5/Ctrl+F6	tăng/giảm âm lượng.
Ctrl+F11	xoay thiết bị ngang/dọc.

Nếu bạn cấu hình AVD với CPU/ABI là Intel Atom (x86) và kích hoạt GPU, bạn có thể tăng tốc đáng kể AVD bằng Intel HAXM. Mặc dù AVD được cung cấp như thiết bị ảo chủ yếu, nhưng AVD chạy chậm và kém thân thiện. Thực tế, bạn nên dùng một trong các giải pháp máy ảo được giới thiệu sau, để thay thế AVD.

b) Genymotion

Xây dựng trên dự án mã nguồn mở AndroVM, Genymotion (<https://www.genymotion.com>) là một bộ công cụ đầy đủ cung cấp môi trường giả lập cho Android, cài đặt đơn giản. Genymotion cần thiết cho người phát triển ứng dụng, nhân viên kiểm thử, đặc biệt hữu ích khi phát triển game trên Android. Genymotion chạy nhanh hơn ADV rất nhiều, hỗ trợ OpenGL và nhiều chức năng khác, xem: <https://cloud.genymotion.com/page/doc/>.

Mặc dù phiên bản miễn phí (phải đăng ký tài khoản) hạn chế nhiều tính năng, nhưng đủ để bạn dùng để thử nghiệm ứng dụng Android của bạn.

Đầu tiên, bạn cần đăng ký tài khoản để tải về Genymotion, tài khoản này cũng được dùng khi tải về các thiết bị ảo.


Do Genymotion dùng máy ảo Oracle VirtualBox, bạn nên tải về bản tích hợp Genymotion Windows 32/64 bits (with VirtualBox).

Tiến trình cài đặt như vậy gồm hai giai đoạn: cài đặt VirtualBox và cài đặt Genymotion.

Ban đầu, Genymotion chưa có thiết bị ảo, bạn phải chọn tải chúng về: khởi động Genymotion, click Add và chọn tải về một hoặc nhiều thiết bị ảo từ danh sách thiết bị ảo do Genymotion cung cấp. Có thể chọn kích thước màn hình thiết bị ảo tùy ý vì bạn có thể kéo thu cửa sổ thiết bị ảo sao cho nằm gọn trong màn hình máy thật.

Từ Android Studio, nếu bạn muốn thử nghiệm ứng dụng của bạn trên Genymotion, thực hiện như sau:

- Khởi động Genymotion từ icon cài đặt. Chọn Settings > ADB, rồi chọn Use custom Android SDK tools (chạy khá lâu). Nhấn nút Browse tìm đến thư mục cài đặt Android SDK. Nhấn OK.

- Cài đặt plugin Genymotion cho Android Studio. Vào File > Settings > Plugins, trong ô tìm kiếm, nhập "Genymotion" và tiến hành cài đặt. Nếu thành công, icon  của plugin Genymotion sẽ hiển thị trên toolbar. Click icon để mở cửa sổ Genymotion Device Manager, chọn thiết bị ảo, click Start để khởi động thiết bị ảo. Bạn sẽ chọn được thiết bị ảo đang chạy này trong cửa sổ Choose Device khi chạy ứng dụng. Khi thiết bị ảo đã chạy, có thể đóng cửa sổ Genymotion Device Manager. Genymotion có một tính năng rất thuận tiện, khi cài đặt ứng dụng từ gói .apk lên máy ảo Genymotion, bạn chỉ cần kéo thả gói .apk lên máy ảo đang chạy.

c) BlueStacks App Player

BlueStacks App Player (<http://www.bluestacks.com/app-player.html>) cho phép chạy ứng dụng Android trong cửa sổ giả lập trên hệ điều hành Windows và MAC. Người dùng có thể trải nghiệm các ứng dụng Android tải từ các kho ứng dụng Android (Google Play Store, AmazonStore, ...) hoặc cài đặt trực tiếp từ tập tin .apk.

BlueStacks App Player cài đặt dễ dàng, sau khi khởi động BlueStacks App Player, bạn sẽ chọn được thiết bị này trong cửa sổ Choose Device khi chạy thử ứng dụng.

d) Andy OS

Andy OS (<http://www.andyroid.net/>) là một giải pháp máy ảo khác, cũng chạy trên Oracle VirtualBox, có nhiều ưu điểm so với BlueStacks nhưng chất lượng hình ảnh thì kém hơn Genymotion.


- Android Studio không cung cấp plugin cho Andy OS. Tuy nhiên, khi chạy ứng dụng bạn vẫn thấy máy ảo Andy OS đang chạy trong hộp thoại Choose Device và có thể chọn máy ảo Andy OS để chạy thử ứng dụng của bạn.

- Andy OS hỗ trợ microphone tốt, bạn có thể dùng các ứng dụng như Voice Search. Genymotion trên Windows lại hạn chế tính năng này. Nói chung Andy OS hỗ trợ media rất tốt, các thử nghiệm media (audio, video, camera) bạn nên dùng Andy OS.

- Từ Google Play, bạn có thể cài đặt ứng dụng Andy Remote Control lên thiết bị thật bên ngoài. Sau đó, thông qua Wifi, dùng ứng dụng này để điều khiển từ xa máy ảo đang chạy Andy OS, bạn chỉ đơn giản cung cấp địa chỉ IP của máy đang chạy Andy OS trên đó. Điều này cho phép bạn tận dụng những ưu thế của thiết bị thật (cảm biến gia tốc, màn hình cảm ứng) để chạy ứng dụng đặc biệt trên Andy OS như game tương tác vật lý, ứng dụng có dùng cảm biến.



2. Android Device Monitor (ADM)

ADM dùng để theo dõi máy ảo, ADM thay thế Dalvik Debug Monitor Server (DDMS) đã lạc hậu. Có hai cách truy cập ADM:

- + Dùng công cụ của SDK, <Android SDK>/tools/monitor.bat.
- + Dùng Android Device Monitor (ADM) của Android Studio, icon  trên tool bar.

Truy cập được ADM, bạn có thể thực hiện nhiều tác vụ lên AVD:

- Chụp hình màn hình AVD:

Trong ADM, tab Devices, click nút Screen Capture ; hoặc trong Android Studio, tool window Android (Alt+6), click icon  góc trên trái của tool window.

- Giả lập cuộc gọi hoặc gửi SMS đến thiết bị:

Chú ý, số phone của AVD là số hiệu port AVD hoạt động, bạn tìm thấy trên title bar của AVD, ví dụ 5554:xxx.

Khi AVD đang hoạt động, dùng một trong hai cách sau:



- + Nhập telnet localhost 5554 (số phone AVD). Để gửi SMS: sms send <sender> <message>, ví dụ: sms send 1234 Hello. Để giả lập cuộc gọi: gsm call <sender>, ví dụ: gsm call 1234.
- + Trong ADM, chọn tab Emulator Control, phần Telephony Actions, nhập vào trường Incoming Number số phone của người gọi, ví dụ: 5554. Chọn gửi SMS (nút radio SMS) hoặc giả lập cuộc gọi (nút radio Voice). Nếu gửi SMS, nhập nội dung tin nhắn vào Message. Click nút Send.

Trên AVD, nhấn F4 để kết thúc cuộc gọi.

Đôi lúc cần hai AVD cùng hoạt động, AVD này có thể thực hiện cuộc gọi hoặc gửi SMS đến AVD kia, số phone của AVD là số hiệu port AVD hoạt động.

- Chuyển tập tin vào AVD và ngược lại:

Khi AVD đang hoạt động, trong ADM, chọn tab  File Explorer, góc trên phải có các icon:

- + Pull a file from the device : "kéo" tập tin được chọn từ AVD về.
- + Push a file onto the device : "đẩy" một tập tin từ máy tính vào AVD.
- + Dấu +: tạo thư mục mới trên AVD.
- + Dấu -: xóa tập tin, thư mục trên AVD nếu có quyền.

3. ADB (Android Debug Bridge)

Khi DDMS khởi động, nó kết nối với ADB để theo dõi máy ảo. ADB xem như cầu nối giữa DDMS và máy ảo (hoặc máy thực).

Tập tin <Android SDK>/platform-tools/adb.exe là công cụ dòng lệnh cho phép bạn liên lạc với thiết bị ảo và bộ giả lập bằng cách thực thi các lệnh:

- Chạy trực tiếp:

Liệt kê các thiết bị ảo	adb devices
Dừng ADB server	adb kill-server
Khởi động ADB server	adb start-server
Sao chép tập tin đến thiết bị	adb push <local source file path> <device destination file path>
Lấy tập tin từ thiết bị	adb pull <device source file path> <local destination file path>
Cài đặt ứng dụng	adb install <file path to apk>
Gỡ bỏ ứng dụng	adb uninstall <package name>
Sao lưu nội dung thiết bị	adb backup
Phục hồi nội dung thiết bị	adb restore <archive name>
Khởi động bình thường	adb reboot
Khởi động trong chế độ recovery	adb reboot recovery

Khởi động trong bootloader adb reboot bootloader

- Dùng ADB Shell:

Khi có một AVD đang hoạt động, khởi động ADB Shell bằng lệnh adb shell, đây là một shell BASH, nhập lệnh tại dấu nhắc đặc quyền #, giống như phiên làm việc với hệ điều hành Linux.

Một số tiện ích gọi qua shell:

Dùng logcat để biết cách dùng LogCat, xem trợ giúp bằng lệnh logcat --help
 Dùng monkey monkey là công cụ kiểm thử, mô phỏng một loạt sự kiện nhập khác nhau
 Dùng sqlite3 chạy sqlite3 rồi thực thi các truy vấn với hệ cơ sở dữ liệu này

4. Lưu ý chung

- Máy ảo Genymotion chạy trên Oracle VM VirtualBox. Nếu card mạng ảo của VirtualBox, thường là VirtualBox Host-Only Network, không hoạt động do lý do nào đó, ví dụ do bạn vô tình chọn Disabled, máy ảo sẽ không chạy. Vào Network and Sharing Center để bảo đảm card mạng ảo đang hoạt động và vào VirtualBox để bảo đảm các thiết lập về Network cho máy ảo là đúng.

- Mặc định, Android không hiển thị tùy chọn Developer options trong phần Settings. Bạn phải tìm mục Build number, thường trong phần Settings > About phone, tap 7 lần lên mục Build number để kích hoạt Developer options.

Thiết bị Android thật

Nếu bạn có thiết bị Android thật, ứng dụng của bạn sẽ được thử nghiệm rất nhanh. Tuy nhiên, rõ ràng bạn chỉ thử nghiệm được trên một thiết bị cụ thể (mobile hoặc tablet) với một API Level cụ thể. Bạn có thể dùng tiện ích như Wifi ADB, không cần cáp USB. Tuy nhiên, để đơn giản, nên kết nối thiết bị Android thật đến máy tính bằng cáp Micro USB, chế độ Charge only.

- Cài đặt USB driver cho máy tính kết nối với thiết bị Android thật.

Nếu dùng Android Developer Phone (ADP) như Nexus, bạn cần cài đặt Google USB Driver thông qua Android SDK Manager, phần Extra. Nếu dùng Android Power Device như Samsung Galaxy, HTC, bạn cần cài đặt OEM USB driver.

Bạn cũng có thể dùng tiện ích như: USB Driver Tool (<http://visualgdb.com/UsbDriverTool/UsbDriverTool-sfx.exe>), click phải lên thiết bị trong danh sách Device Name rồi chọn Install Android ADB Driver trong menu tắt.

- Kích hoạt USB Debugging trên thiết bị Android thật.

Trên thiết bị Android thật, để kích hoạt USB debugging, vào: Settings > Applications > Development options, chọn checkbox USB Debugging. Tùy chọn USB Debugging có thể khác nhau một chút trên các thiết bị khác nhau, bạn tham khảo tài liệu hướng dẫn sử dụng dành riêng cho máy của bạn.

- Kiểm tra cài đặt thiết bị Android thật.

Dùng lệnh adb devices kiểm tra để chắc rằng ADB đã nhận ra thiết bị Android đang kết nối. Nếu ADB quản lý được thiết bị thật, bạn sẽ chọn được thiết bị này trong cửa sổ Choose Device của Android Studio khi chạy ứng dụng.

Bạn cần phải nhận thức rằng, thiết bị thật không được root sẵn như AVD. Thuật ngữ root là thao tác chiếm quyền cao nhất trong hệ điều hành Android để làm chủ hoàn toàn thiết bị, ví dụ cho phép duyệt và thao tác tự do trên hệ thống tập tin.

Xây dựng ứng dụng

1. Mô hình MVC

Khi xây dựng ứng dụng Android, có ba cách tiếp cận:

- Java-based: giống lập trình với Java Swing, dùng Java tạo layout, tạo các UI component và gán xử lý sự kiện, đặt các UI component vào layout và gọi setContentView() với layout tạo được. Khai báo các chuỗi sử dụng bằng Java.

```
public class MainActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout mLayout = new LinearLayout(this);
        Button mButton = new Button(this);
        mButton.setText("Tap me!");
        mButton.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                Toast.makeText(getBaseContext(), "You tap me!", Toast.LENGTH_SHORT).show();
            }
        });
        mLayout.addView(mButton);
        setContentView(mLayout);
    }
}
```

- XML-based: dùng XML định nghĩa layout với các UI component, khai báo UI component có gán xử lý sự kiện. Khai báo các chuỗi sử dụng bằng XML.

Tập tin strings.xml:

```
<resources>
    <string name="labelButton">Tap me!</string>
</resources>
```

Tập tin activity_main.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



```

        android:text="@string/labelButton"
        android:onClick="onClick" />
</LinearLayout>

```

Trong MainActivity.java, layout sẽ được tham chiếu từ tập tin XML như một tài nguyên trong tập tin R.java, tài nguyên này được chuyển đến phương thức setContentView().

```

public class MainActivity extends ActionBarActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Toast.makeText(getApplicationContext(), "You tap me!", Toast.LENGTH_SHORT).show();
    }
}

```

- Hybrid: dùng XML định nghĩa các chuỗi, layout, tạo các đơn thể UI với ID được gán. Thông qua tập tin R.java, dùng Java lấy UI component để gán xử lý sự kiện cho nó.

Khi xây dựng ứng dụng có UI và xử lý sự kiện, mô hình MVC (Model-View-Controller) thường được sử dụng. Nhưng do Android có vài cách tiếp cận như trên, các thành phần của mô hình MVC dễ bị nhầm lẫn.

Bạn có thể phân biệt các thành phần của mô hình MVC như sau:

- Model: mô hình hóa nghiệp vụ mà ứng dụng quan tâm. Nói cách khác, chúng chứa thao tác nghiệp vụ (business logic). Bao gồm các đối tượng nghiệp vụ (business object), các thư viện cung cấp thêm logic cho chúng, và các đối tượng dữ liệu (value object) mà chúng thao tác lên đó. Trong Android, nói chung chúng là các lớp do bạn tạo ra để xử lý dữ liệu cho nghiệp vụ.
- View: là các đối tượng người dùng nhìn thấy và tương tác. Trong Android, chúng là UI layout, UI control bạn định nghĩa trong tập tin XML hoặc trực tiếp bằng code. Chúng thực hiện việc tương tác với người dùng nhưng không thực hiện việc xử lý.
- Controller: kết nối giữa View và Model, chúng chứa thao tác của ứng dụng (application logic). Controller gọi Model để đáp ứng các sự kiện được kích hoạt bởi View và quản lý dòng dữ liệu đi từ Model đến để cập nhật lại View. Chúng nhận và xử lý truy vấn từ người dùng bằng cách chuyển truy vấn thành truy vấn nghiệp vụ đến Model. Trong Android, chúng là các lớp con của Activity, Fragment, Service.

2. Logcat

Android hỗ trợ logging (ghi nhận hoạt động) trong lớp android.util.Log, vài phương thức quan trọng như sau:

Log Level	Phương thức	Mục tiêu
ERROR	Log.e()	Ghi nhận các lỗi.
WARNING	Log.w()	Ghi nhận các cảnh báo.
INFO	Log.i()	Ghi nhận các thông điệp thông tin.
DEBUG	Log.d()	Ghi nhận các thông điệp gỡ lỗi.
VERBOSE	Log.v()	Ghi nhận thông điệp nhiều dòng. Độ ưu tiên thấp nhất.
WTF	Log.wtf()	Ghi nhận tình trạng nghiêm trọng (what a terrible failure).

Khi code, bạn khai báo một chuỗi hằng, dùng như tag để lọc các thông điệp của bạn. Sau đó gọi các phương thức của Log tùy tình huống, với tham số thứ nhất là tag được khai báo.


```

private static final String DEBUG_TAG = "MyFirstAppLogging";
Log.i(DEBUG_TAG, "In the onCreate() method");

```

Khi chạy ứng dụng, trong Android Studio, nhấn Alt+6 để mở Android DDMS, bạn sẽ thấy công cụ  Logcat chuyên thu thập và hiển thị các thông điệp log.

Trong listbox thả xuống góc trên phải của Logcat, chọn Edit Filter Configuration để mở hộp thoại Create New Logcat Filter, hộp thoại này cho phép cấu hình filter lọc các thông điệp log. Click + để thêm filter: nhập Name và by Log Tag (regex). Log Tag phân biệt chữ hoa chữ thường và có thể áp dụng regular expression (chú ý, không phải wildcard).

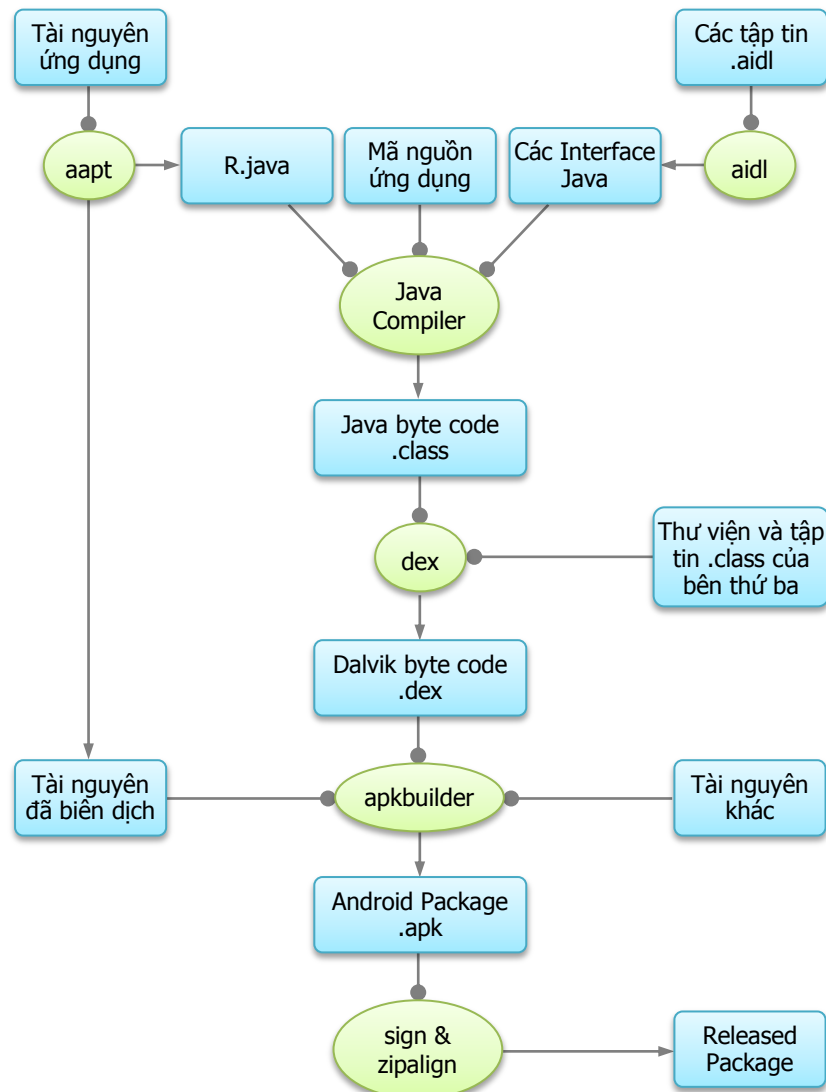
Khi LogCat đầy, xóa hết bằng cách click icon .

3. Build ứng dụng Android

Các công cụ tham gia quá trình build ứng dụng Android:

- aapt** Android Asset Packaging Tool, công cụ lấy các tập tin tài nguyên của ứng dụng (các tập tin XML như activity_main.xml, strings.xml và AndroidManifest.xml, ...) rồi biên dịch chúng. Tập tin R.java được sinh ra giúp tham chiếu bạn tài nguyên từ code của bạn.
- aidl** công cụ chuyển các tập tin .aidl (Android Interface Definition Language) thành các tập tin .java.
- dex** công cụ chuyển các tập tin Java byte code .class thành Dalvik byte code .dex (Davilk EXecutable), có thể thực thi trên máy ảo Dalvik.
- apkbuilder** công cụ đóng thành gói .apk (Android Application Package), đóng gói tài nguyên không biên dịch (ví dụ ảnh), tài nguyên được biên dịch và các tập tin .dex.

Hình dưới mô tả quá trình build một ứng dụng Android từ tài nguyên cung cấp và mã nguồn.



4. Dịch ngược gói .apk

Nếu bạn cần tham khảo mã nguồn hoặc kiểm tra kết quả build, bạn có thể dịch ngược (decompiler) gói .apk. Quy trình dịch ngược nói chung như sau:

- Gói .apk thực chất được nén bằng PKzip. Dùng WinRar giải nén gói .apk để nhận được classes.dex.
- Tập tin .dex là Dalvik bytecode, bạn phải chuyển thành Java bytecode bằng tiện ích dex2jar, tải về tại:

<http://sourceforge.net/projects/dex2jar/>

```
d2j-dex2jar.bat classes.dex
```

Bạn nhận được classesxx.jar.

- Dịch ngược gói .jar thành mã nguồn, dùng tiện ích Java decompiler như jd-gui.exe, tải về tại <http://jd.benow.ca/>. Trong giao diện của tiện ích này, chỉ đến gói .jar để dịch ngược.

5. Triển khai và phân phối ứng dụng

Để triển khai một ứng dụng Android đến người dùng, bạn cần gói released của ứng dụng để người dùng có thể cài đặt và chạy trên thiết bị của họ. Gói released cũng được gán certificate của nhà phát triển và được tối ưu hóa để chạy trên thiết bị của người dùng.

Gói APK thực chất là một tập tin ZIP chứa toàn bộ những thành phần giúp cho ứng dụng chạy được và phân biệt với các ứng dụng khác.



Có nhiều lựa chọn cho nhà phát triển để triển khai một ứng dụng đến với người dùng. Tuy nhiên, phổ biến nhất là Google Play (Android Market).

Việc chuẩn bị và tối ưu hóa gồm nhiều công đoạn nhưng dưới đây chỉ đề cập đến các công đoạn quan trọng nhất.

a) Bước 1 – tạo bản release - ready APK

Trước tiên cần thay đổi Build Variant của project từ debug trở thành release. Bạn mở tool window Build Variants (góc dưới trái), chuyển Build Variant của tất cả các module từ debug thành release.

Tiếp theo, bạn cần cấu hình thông tin khóa dùng sinh ra gói ứng dụng APK được ký.

Google cung cấp cho nhà phát triển một tiện ích giúp họ tự tạo certificate cho ứng dụng của mình mà không cần phải xin certificate hay chấp thuận từ một tổ chức nào. Nhà phát triển có thể sử dụng một certificate cho những ứng dụng Android của mình, nhờ đó, họ có thể:

- Dễ dàng cập nhật.
- Chạy nhiều ứng dụng với cùng một user ID.
- Chia sẻ code/dữ liệu.

Trong Android Studio:

- Chọn Build > Generate Signed APK... để mở hộp thoại Generate Signed APK Wizard.
- Chọn keystore có sẵn (Choose existing...) hoặc tạo keystore mới (Create new...).

Nếu chọn tạo certificate mới, hộp thoại New Key Store yêu cầu bạn nhập vào một số thông tin để tạo keystore và cặp khóa:

Key store path	Vị trí lưu keystore.
Alias	Tên đặt cho khóa để dễ nhớ.
Validity (years)	Thời hạn hiệu lực, Google đề nghị ít nhất 25 năm.
First and Last Name	Tên nhà phát triển ứng dụng.
Organizational Unit	Tên đơn vị trong công ty, tổ chức.
Organization	Tên công ty, tổ chức.
City or Locality	Thành phố hoặc địa phương.
State or Province	Bang hoặc tỉnh.
Country Code (XX)	Mã quốc gia, ví dụ VN.

Hộp thoại cũng yêu cầu nhập hai password: một để sử dụng keystore và một để sử dụng private key trong cặp khóa được tạo. Sau khi tạo keystore bạn có màn hình giống như khi bạn mở một keystore có sẵn. Chọn Next để sang bước kế tiếp.

- Sinh gói APK được ký

Khóa private trong keystore được dùng để ký certificate, certificate này được gán cho ứng dụng bằng cách cho Destination APK path chỉ đến thư mục app của project. Hộp thoại này cũng cho phép bạn kích hoạt ProGuard (mặc định là false) bằng cách chọn checkbox Run ProGuard.

ProGuard dùng để vô hiệu hóa các phương thức log, debug vì chúng sẽ gây ảnh hưởng đến hiệu suất của ứng dụng. Bạn có thể mở tập tin /src/build.gradle để xem cấu hình cho ProGuard:

```
buildTypes {
    release {
        minifyEnabled true // nén code
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```

Bạn có thể cấu hình tập tin proguard-rules.pro để xóa log. Mở tập tin, thêm vào phần -assumenosideeffects những phương thức thuộc lớp Log sẽ phải xóa.

```
-assumenosideeffects class android.util.Log {
    public static boolean isLoggable(java.lang.String, int);
    public static int v(...);
    public static int i(...);
    public static int w(...);
}
```

Bạn sẽ nhận được thông báo sinh gói APK đã ký nếu gán certificate thành công. Kiểm tra bằng cách cài đặt và chạy thử.

Khi gói APK được tạo ra, Google tự động gán một License key cho ứng dụng. Khóa này có thể được dùng để giới hạn quyền sử dụng ứng dụng nếu nhà phát triển muốn.

b) Bước 2 – quản lý ứng dụng bằng Developer Console

Để có thể đăng tải một ứng dụng lên Android Market, trước hết phải có một tài khoản để quản lý những ứng dụng của mình. Google cho phép bạn quản lý toàn bộ ứng dụng của bạn bằng Developer Console.

Bạn có thể sử dụng Developer Console theo các bước:

- Đăng nhập bằng tài khoản Google, nếu không có tài khoản Google có thể đăng ký miễn phí tại:

<https://play.google.com/apps/publish/signup>

- Đồng ý với các điều khoản về sử dụng Android Market.
 - Trả phí \$25, bạn chỉ phải trả một lần cho một tài khoản Google và thanh toán bằng Google Wallet.
 - Sau khi tài khoản được xác nhận, bạn sẽ nhận được thông báo gửi đến email mà bạn đã dùng để đăng ký.
- Bạn có thể tùy chỉnh lại thông tin tài khoản của mình bằng cách cấu hình trong mục Settings của Developer Console.

c) Bước 3 – đăng tải ứng dụng

Sau khi đã có tài khoản Google, bạn có thể đăng tải ứng dụng của mình lên Android Market. Việc đăng tải ứng dụng giúp người dùng có thể tải ứng dụng xuống thiết bị Android của họ và sử dụng. Việc đăng tải ứng dụng gồm các bước:

- Chọn Add new application trong Developer Console.
- Chọn ngôn ngữ mặc định (default language) và đặt tên (title) cho ứng dụng.
- Ở thẻ APK chọn Upload your first APK to Production, hộp thoại sẽ hiện ra cho phép bạn tìm tập tin APK có sẵn trong máy. Bạn cũng có thể kéo thả tập tin APK đã được tạo vào khung trên.

Sau khi đăng tải xong, Google sẽ tự động xác định các thiết bị hỗ trợ được ứng dụng vừa đăng tải, bạn có thể tùy chỉnh lại trong phần Supported devices.

Google cũng hỗ trợ nhà phát triển tạo các thông tin hiển thị lên Android Market sau khi đăng tải ứng dụng. Điều này được thực hiện thông qua Store Listing trong Developer Console:

- Chọn thẻ Store Listing trong Developer Console và nhập các thông tin cần thiết, giúp người dùng thêm thông tin chọn lựa ứng dụng của bạn.

Title	Tên của ứng dụng.
Short description	Mô tả tóm tắt về ứng dụng.
Full description	Mô tả chi tiết về ứng dụng (4000 từ).
- Chọn ảnh screenshot để người dùng có thể xem trước ứng dụng của bạn, 2-8 ảnh, ảnh ứng dụng chạy trên tablet 7" hoặc 10". Chọn ảnh screenshot tốt cũng góp phần thu hút người dùng lựa chọn ứng dụng của bạn.	
- Chọn Type và Category cho ứng dụng mà bạn đăng tải.	
Type	Kiểu ứng dụng của bạn, application hoặc game.
Category	Thể loại của ứng dụng của bạn: giải trí, kiến thức, giải đố, thể thao, v.v...
Content rating	Độ tuổi phù hợp cho người dùng mà bạn nhắm đến.

- Định giá và xác định vùng phân phối (Pricing & Distribution)

Khi đăng tải một ứng dụng lên Android Market, nhà phát triển cần xác định ứng dụng của mình có thu phí người dùng hay không. Nếu ứng dụng có thu phí, nhà phát triển phải cấu hình một tài khoản giao dịch (merchant account) để hỗ trợ thu phí mỗi khi có người dùng tải xuống ứng dụng. Ngoài ra, nhà phát triển cũng có thể quyết định ứng dụng của mình sẽ được phân phối ở quốc gia hoặc vùng lãnh thổ nào. Chỉ những người dùng thuộc quốc gia hoặc vùng lãnh thổ đã được chọn mới được phép tải ứng dụng.

Sau khi đã hoàn thành các bước trên, ứng dụng của bạn đã có thể sẵn sàng cho việc triển khai lên Android Market, bên góc phải của Developer Console sẽ hiển thị một lựa chọn Ready to publish. Chọn Publish this app để hoàn thành việc phát hành ứng dụng.

d) Cập nhật ứng dụng

Khi đã triển khai lên Android Market, ứng dụng cũng cần được cập nhật theo thời gian để nâng cấp hoặc sửa lỗi. Cập nhật một ứng dụng gồm các bước:

- Cập nhật phiên bản ứng dụng trong tập tin build.gradle (Module: app), các trường versionCode và versionName.

```
defaultConfig {
    applicationId "com.twe.android.FirstApp"
    minSdkVersion 14
    targetSdkVersion 19
    versionCode 2
    versionName "2.0"
}
```

- Thực hiện lại việc gán certificate cho ứng dụng, nhưng không cần phải tạo lại khóa, chỉ cần cung cấp mật khẩu dùng khóa đã tạo trước đó.

- Chọn Update new APK to Production trên Developer Console và kéo thả tập tin APK vừa được cập nhật vào.

- Hộp thoại hiển thị để bạn xem trước phiên bản và có thể nhập thông tin thay đổi vào phần What's new in this version?

- Chọn Publish now to Production để hoàn thành việc cập nhật ứng dụng.

Lưu ý, Google phải mất một khoảng thời gian để cập nhật ứng dụng lên Android Market.

Tài nguyên của ứng dụng

Ngoài viết code cho ứng dụng, bạn còn phải quan tâm đến các tài nguyên khác như nội dung tĩnh mà code của bạn sử dụng, các hình ảnh, các định nghĩa màu sắc, layout, các chuỗi sử dụng trong giao diện người dùng, v.v...

Tổ chức tài nguyên

Android SDK yêu cầu tài nguyên của ứng dụng được đặt trong các thư mục con của thư mục `app/src/main/res` của project.

```
app/
  src/
    main/
      res/
        drawable/
          icon.png
        layout/
          activity_main.xml
        values/
          strings.xml
```

Trong view Project, code nằm trong nhánh `app/java` còn tài nguyên nằm trong nhánh `app/res`.

Tên của thư mục con chứa tài nguyên là quan trọng, Android không thể tìm thấy tài nguyên nếu nó không được đặt đúng chỗ. Android Studio không tự động tạo tất cả thư mục con chứa tài nguyên. Bạn có thể tạo các thư mục tài nguyên còn thiếu nếu cần, bằng cách click phải lên thư mục `res` và chọn `New > Android Resource Directory` từ menu tắt, hộp thoại `New Resource Directory` sẽ mở, cho phép tạo thư mục tài nguyên mới.

Các nhóm tài nguyên

Android nhóm tài nguyên vào 9 nhóm dựa trên loại tài nguyên. Mỗi loại tài nguyên trong 9 nhóm này được lưu trữ trong thư mục con chỉ định của thư mục tài nguyên `src/res`. Khi bạn biên dịch và build ứng dụng, Android tự động sinh ra lớp Java `<application package name>.R`, chứa các hằng dùng để truy cập đến các tài nguyên này từ code của ứng dụng.

Bảng sau liệt kê các nhóm tài nguyên, thư mục con chứa chúng và các nhóm tương ứng bên trong lớp `R`.

Nhóm tài nguyên	Thư mục con	Tham chiếu trong code	Tham chiếu trong XML
Property Animation	<code>animator</code>	<code>R.animator</code>	<code>@animator/<file></code>
Tween Animation	<code>anim</code>	<code>R.anim</code>	<code>@anim/<file></code>
Color State List	<code>color</code>	<code>R.color</code>	<code>@color/<file></code>
Drawable	<code>drawable</code>	<code>R.drawable</code>	<code>@drawable/<file></code>
Layout	<code>layout</code>	<code>R.layout</code>	<code>@layout/<file></code>
Menu	<code>menu</code>	<code>R.menu</code>	<code>@menu/<file></code>
Raw	<code>raw</code>	<code>R.raw</code>	
Simple Values	<code>values</code>	<code>R.arrays</code> <code>R.bool</code> <code>R.color</code> <code>R.dimen</code> <code>R.integer</code> <code>R.string</code> <code>R.style</code>	<code>@bool/<id></code> <code>@color/<id></code> <code>@dimen/<id></code> <code>@integer/<id></code> <code>@string/<id></code> <code>@style/<id></code>
XML	<code>xml</code>	<code>R.xml</code>	

Tập tin `R` tự động được sinh và bạn không được thay đổi nội dung của nó. Mã nguồn của tập tin `R.java` được tự động sinh khi có sự kiện làm thay đổi tình trạng ứng dụng, ví dụ kéo và thả một tập tin từ bên ngoài vào project.

1. Property Animation

Tài nguyên property animation mô tả các bước để tạo hoạt hình một đối tượng bất kỳ trên màn hình bằng cách thay đổi các thuộc tính của chúng trong một khoảng thời gian chỉ định (interval).

Ví dụ: tập tin `src/res/animator/example.xml`

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <objectAnimator android:propertyName="x"
    android:duration="1000"
    android:valueFrom="10"
    android:valueTo="100"
    android:valueType="intType" />
</set>
```

Truy cập tài nguyên trong code:

```
AnimatorSet mAnimatorSet = (AnimatorSet) AnimatorInflater.loadAnimator(this, R.animator.example);
```

2. Tween Animation

Tài nguyên tween animation (tween – in between) mô tả các bước để tạo hoạt hình một đối tượng trên màn hình bằng cách áp dụng một chuỗi các biến hình đơn giản đến nội dung của nó.

Ví dụ: tập tin `src/res/anim/tween.xml`

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <scale android:fromXScale="1.0"
    android:toXScale="2.0"
```



```

        android:duration="1000" />
    <rotate android:fromDegrees="0"
        android:toDegrees="180"
        android:duration="1000" />
</set>

```

Truy cập tài nguyên trong code:

```

TextView mTextView = (TextView) findViewById(R.id.hello_world);
Animation mAnimation = AnimationUtils.loadAnimation(this, R.anim.tween);
mTextView.startAnimation(mAnimation);

```

3. Color State List

Tài nguyên này mô tả các màu cho các trạng thái khác nhau của một view. Ví dụ, một tập các màu áp dụng cho từng trạng thái của một button.

Ví dụ: tập tin src/res/color/button.xml

```

<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:color="#ffff0000"
        android:state_pressed="true" />
    <item android:color="#ff00ff00"
        android:state_focused="true" />
    <item android:color="#ff0000ff" />
</selector>

```

Truy cập tài nguyên trong code:

```

ColorStateList mColorStateList = getResources().getColorStateList(R.color.button);

```

4. Drawable

Tài nguyên này là các hình ảnh đồ họa có thể vẽ ra màn hình, bạn có thể tham chiếu từ code của ứng dụng bằng cách dùng phương thức `Resources.getDrawable()` với resource ID là hằng `R.drawable.<resource name>`.

Android cung cấp vài loại drawable.

Từ Android 4.3, bạn có thể sử dụng `res/mipmap` để lưu trữ hình ảnh "mipmap" (Latin: *multum in parvo*, much in little), thường là các icon. Tài nguyên mipmap sau đó có thể được truy cập bằng cách sử dụng `@mipmap` thay cho `@drawable`.

a) Tập tin bitmap

Tập tin bitmap là một tập tin ảnh có định dạng PNG, JPEG và GIF. Đặt chúng trong thư mục `src/res/drawable`, tên của ảnh là resource ID.

b) Tập tin bitmap XML

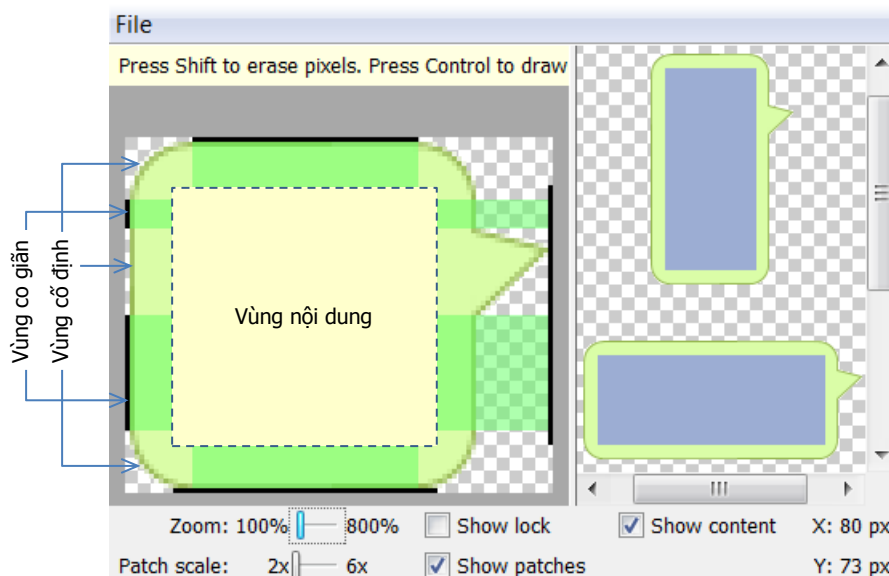
Bạn có thể áp dụng các thuộc tính thêm cho tập tin bitmap có sẵn bằng cách dùng tập tin bitmap XML. Nó là một tập tin XML chỉ đến một tập tin bitmap có sẵn và khai báo những thuộc tính thêm như anti-aliasing (khử răng cưa), dithering (phối màu để giảm số màu sử dụng), bằng cách dùng element `<bitmap>`.

```

<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@mipmap/ic_launcher"
    android:antialias="true" />

```

c) Tập tin nine-patch



Công cụ Draw 9 Patch dùng tạo tập tin nine-patch từ bitmap callout cho trước

Bên phải là callout khi co giãn, chú ý các vùng cố định của callout

Khi một tập tin bitmap co giãn, nó co giãn toàn bộ theo tỷ lệ. Đôi lúc ta muốn một số phần của bitmap (ví dụ các góc) giữ nguyên khi co giãn. Tập tin nine-patch giải quyết vấn đề này.

Tập tin nine-patch là một tập tin bitmap thông thường, có thêm các đường biên màu đen có kích thước 1 pixel nằm sát bên ngoài các cạnh của bitmap đó, cung cấp thêm các thông tin:

- Đường biên trái và đường biên trên xác định các vùng cho phép co giãn.
- Đường biên phải và đường biên dưới xác định biên cho vùng nội dung.

Để phân biệt với tập tin bitmap thông thường, tập tin nine-patch có thêm phần mở rộng .9., ví dụ callout.9.png. Khi hiển thị, các đường biên màu đen thêm vào sẽ không hiển thị.

Để dễ dàng tạo tập tin nine-patch từ tập tin bitmap có sẵn, Android SDK cung cấp công cụ Draw 9 Patch, tại <Android SDK>/tools/draw9patch.bat. Kéo thả tập tin bitmap vào công cụ, bên trái cho phép tạo các đường biên, bên phải cho thấy kết quả: drawable với vùng nội dung khi co giãn.

d) Tập tin XML nine-patch

Giống như tập tin bitmap XML, tập tin XML nine-patch tham chiếu đến một tập tin nine-patch có sẵn và cung cấp thêm tham số, ví dụ dithering, bằng cách dùng element <nine-patch>:

```
<nine-patch xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/callout"
    android:dither="true" />
```

e) Shape Drawable

Shape drawable là hình được định nghĩa trong định dạng XML bằng cách dùng element <shape>. Shape drawable hỗ trợ tất cả các hình như hình chữ nhật, hình tròn, đường thẳng, bầu dục, ... Bạn cũng có thể chỉ định các thuộc tính như bo góc, gradient và màu cho đối tượng hình ảnh thông qua tập tin tài nguyên XML bằng cách dùng các element tương ứng.

Ví dụ: tập tin src/res/drawable/shape.xml

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="4dp" />
    <gradient android:startColor="#FF00"
        android:endColor="#F0F0"
        android:angle="20" />
    <padding android:left="4dp"
        android:top="4dp"
        android:right="4dp"
        android:bottom="4dp" />
</shape>
```

f) State List

State list drawable là nhóm các ảnh ảnh xạ với những trạng thái khác nhau của một đối tượng. Dựa trên trạng thái hiện tại của đối tượng, Android dùng ảnh tương ứng trong danh sách được cung cấp. Bạn định nghĩa state list drawable trong định dạng XML bằng element <selector> với element lồng <item> thể hiện từng ảnh.

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_pressed"
        android:state_pressed="true" />
    <item android:drawable="@drawable/button_focused"
        android:state_focused="true" />
</selector>
```

5. Layout

Tài nguyên layout định nghĩa kiến trúc của UI. Nó chứa danh sách các view và các thuộc tính định nghĩa cách chúng được đặt lên màn hình. Các tập tin layout XML được đặt trong thư mục src/res/layout, tên của tập tin layout XML được sử dụng như định danh dùng tham chiếu đến chúng.

Ví dụ: tập tin src/res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/text_hello"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/button"
        android:text="@string/button_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Sau đó, bạn có thể áp dụng tài nguyên layout này cho một activity:

```
setContentView(R.layout.activity_main);
```

6. Menu

Tài nguyên menu định nghĩa cấu trúc các menu của ứng dụng như menu options, menu tắt (context), hoặc menu con.

Ví dụ: tập tin src/res/menu/menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity" >
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:orderInCategory="100"
          android:showAsAction="never" />
</menu>
```

Sau đó, "inflate" menu này trong code:

```
getMenuInflater().inflate(R.menu.menu_main, menu);
```

7. Raw

Tài nguyên raw (thô) là các tài nguyên không thuộc nhóm tài nguyên nào. Chúng được đặt vào thư mục src/res/raw. Tài nguyên raw không có các tập tin XML như các loại tài nguyên khác, chúng có định dạng bất kỳ. Bạn tham chiếu đến chúng trong code như sau: R.raw.<resource file name> (không có phần mở rộng).

Ví dụ, dùng phương thức Resources.openRawResource() để mở stream nhập src/res/raw/attributions.txt.

```
InputStream is = getResources().openRawResource(R.raw.attributions);
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String s = null;
try {
    while((s = br.readLine()) != null) {
        Toast.makeText(getBaseContext(), s, Toast.LENGTH_SHORT).show();
    }
    is.close();
    br.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

8. Values

Tài nguyên value là các trị đơn giản, như string, integer, boolean và trị màu. Chúng được đặt trong thư mục src/res/values. Các tập tin tài nguyên value có định dạng XML, một tập tin tài nguyên value có thể chứa nhiều trị.

Trong khi các tài nguyên khác được định danh với resource ID của chúng, Android cho phép linh động tổ chức các trị bất cứ cách nào có ý nghĩa. Ví dụ, bạn có thể đặt tài nguyên màu trong tập tin colors.xml, hoặc trong tập tin tài nguyên riêng cho từng activity.

Ví dụ: tập tin src/res/values/strings.xml

```
<resources>
    <string name="app_name">Hello Android</string>
    <bool name="registered">false</bool>
    <color name="title_background">#ffff0000</color>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <integer name="count">10</integer>
    <integer-array name="numbers">
        <item>1</item>
        <item>2</item>
    </integer-array>
</resources>
```

Các loại tài nguyên value có thể khai báo, bên trong element <resources>:

- Tài nguyên chuỗi (string)

Tài nguyên chuỗi là các chuỗi văn bản, thường dùng cho giao diện, định nghĩa thông qua R.string.<resource name>.

```
<string name="app_name">Hello Android</string>
```

Sau đó, tham chiếu nó trong code bằng phương thức getString() với resource ID là R.string.<resource name>.

```
String mAppName = getString(R.string.app_name);
```

Bạn cũng có thể định nghĩa mảng các chuỗi bằng cách dùng element <string-array>.

```
<string-array name="android_versions">
    <item>KitKat</item>
    <item>Lollipop</item>
</string-array>
```

Sau đó, tham chiếu nó trong code bằng phương thức getStringArray() với resource ID là R.array.<resource name>.

```
String[] mAndroidVersions = getResources().getStringArray(R.array.android_versions);
```

- Tài nguyên số lượng (quantity)

Android cung cấp một kiểu tài nguyên đặc biệt để định nghĩa các chuỗi xử lý số lượng khác nhau, cho phép bạn dùng chính xác các chuỗi về số lượng. Element <plurals> chứa một hoặc nhiều element <item> cho số lượng cần xử lý. Thuộc tính quantity của element <item> chỉ định trường hợp chuỗi sẽ được dùng, theo bảng sau:

Số lượng	Mô tả
Zero	Chuỗi dùng cho số lượng 0.

One	Chuỗi dùng cho số lượng 1.
Two	Chuỗi dùng cho số lượng 2.
Few	Chuỗi dùng cho số lượng ít (vài).
Many	Chuỗi dùng cho số lượng nhiều.
Other	Chuỗi dùng cho số lượng chỉ định. %d có thể dùng để giữ chỗ cho số.

```
<plurals name="books_found">
  <item quantity="zero">Book not found.</item>
  <item quantity="one">One book found.</item>
  <item quantity="other">%d books found.</item>
</plurals>
```

Sau đó, tham chiếu nó trong code bằng phương thức `getQuantityString()` với resource ID là `R.plurals.<resource name>`. Tham số thứ nhất để chọn chuỗi, tham số thứ hai để truyền cho %d.

```
String mZeroBooksFound = getResources().getQuantityString(R.plurals.books_found, 0, 0);
String mEightBooksFound = getResources().getQuantityString(R.plurals.books_found, 8, 8);
```

- Tài nguyên boolean

Bạn có thể định nghĩa tài nguyên Boolean bằng cách dùng element `<bool>`:

```
<bool name="registered">false</bool>
```

Sau đó, tham chiếu nó trong code bằng phương thức `getBoolean()` với resource ID là `R.bool.<resource name>`.

```
boolean mRegistered = getResources().getBoolean(R.bool.registered);
```

- Tài nguyên màu (color)

Bạn có thể định nghĩa các trị màu bằng các dùng element `<color>`:

```
<color name="title_background">#ffff0000</color>
```

Trị màu được chỉ định theo mô hình màu RGB (Red, Green, Blue), bạn có thể cung cấp thêm kênh A (Alpha) để chỉ định độ trong suốt (transparency), như bảng sau:

Định dạng	Mô tả	Ví dụ
#RGB	Red-Green-Blue, độ chính xác 1 số hex (0 đến F).	#F00 (red)
#ARGB	Alpha-Red-Green-Blue, độ chính xác 1 số hex (0 đến F).	#60F0 (green + transparency)
#RRGGBB	Red-Green-Blue, độ chính xác 2 số hex (00 đến FF).	#0000FF (blue)
#AARRGGBB	Alpha-Red-Green-Blue, độ chính xác 2 số hex (00 đến FF).	#6200FF00 (green + transparency)

Sau đó, tham chiếu nó trong code bằng phương thức `getColor()` với resource ID là `R.color.<resource name>`.

```
int mBackgroundColor = getResources().getColor(R.color.title_background);
```

- Tài nguyên kích thước (dimension)

Bạn có thể chỉ định kích thước cho các đơn thể UI bằng cách dùng element `<dimen>`:

```
<dimen name="horizontal_margin">16dp</dimen>
```

Trị kích thước chỉ định có đơn vị đo là: dp, sp, pt, px, mm và in.

Sau đó, tham chiếu nó trong code bằng phương thức `getDimension()` với resource ID là `R.dimen.<resource name>`.

```
float mHorizontalMargin = getResources().getDimension(R.dimen.horizontal_margin);
```

Kết quả trả về là trị kích thước nhân với đơn vị đo. Nếu muốn nhận trị pixel, dùng phương thức `getDimensionPixelSize()`.

- Tài nguyên số nguyên (integer)

Bạn cũng có thể định nghĩa các số nguyên bằng cách dùng element `<integer>`.

```
<integer name="count">10</integer>
```

Sau đó, tham chiếu đến nó trong code bằng phương thức `getInteger()` với resource ID là `R.integer.<resource name>`.

```
int mCount = getResources().getInteger(R.integer.count);
```

Cũng giống như mảng chuỗi, tài nguyên value cũng hỗ trợ mảng số nguyên. Bạn có thể định nghĩa mảng số nguyên bằng cách dùng element `<integer-array>`.

```
<integer-array name="numbers">
  <item>1</item>
  <item>2</item>
</integer-array>
```

Sau đó, tham chiếu nó trong code bằng phương thức `getIntArray()` với resource ID là `R.array.<resource name>`.

```
int[] mNumbers = getResources().getIntArray(R.array.numbers);
```

- Tài nguyên mảng

Mảng không chỉ hỗ trợ kiểu chuỗi và kiểu số nguyên. Bạn cũng có thể định nghĩa mảng các kiểu tài nguyên khác, ví dụ mảng các drawable, bằng cách dùng element `<array>`, thậm chí có thể định nghĩa mảng chứa các kiểu tài nguyên khác nhau.

```
<array name="colors">
  <item>#FFF</item>
  <item>#000</item>
</array>
```

Bạn có thể tham chiếu tài nguyên mảng trong code bằng phương thức `Resources.obtainTypedArray()` với resource ID là hằng `R.array.<resource name>`. Sau đó, lấy các phần tử của tài nguyên mảng thông qua các phương thức của lớp `content.res.TypedArray` tương ứng với kiểu tài nguyên.

```
TypedArray mColors = getResources().obtainTypedArray(R.array.colors);
int mBackground = mColors.getColor(0, Color.BLACK);
int mForeground = mColors.getColor(1, Color.WHITE);
```

9. Tài nguyên XML

Ngoài các tài nguyên có cấu trúc quy định, Android cho phép dùng tập tin XML tùy ý như một nguồn tài nguyên, lưu tại thư mục `src/res/xml`. AAPT xử lý sơ bộ (pre-parsing) chúng trong thời gian build, nên chúng được sử dụng nhanh hơn trong thời gian chạy. Bạn có thể tham chiếu đến chúng, bằng cách gọi phương thức `getXml()` với resource ID là hằng `R.xml.<resource file name>`. Phương thức này trả về một thực thể `content.res.XmlResourceParser`. Đây là lớp con của `XmlPullParser` được dùng để phân tích tập tin XML đó.

```
private String getEventsFromXMLFile(Activity activity)
    throws XmlPullParserException, IOException {
    XmlResourceParser parser = activity.getResources().getXml(R.xml.configuration);
    StringBuilder sb = new StringBuilder();
    for (int eventType = parser.getEventType();
        eventType != XmlPullParser.END_DOCUMENT;
        eventType = parser.next()) {
        switch (eventType) {
            case XmlPullParser.START_TAG:
                sb.append("\nStart tag " + parser.getName()); break;
            case XmlPullParser.TEXT:
                sb.append("\nText " + parser.getText()); break;
            case XmlPullParser.END_TAG:
                sb.append("\nEnd tag " + parser.getName());
        }
    }
    return sb.toString();
}
```

Tài nguyên mặc định và tài nguyên thay thế

Bên cạnh tập tài nguyên mặc định, ứng dụng cũng có thể chứa các tài nguyên thay thế cho các cấu hình thiết bị khác nhau. Như vậy, ứng dụng của bạn sẽ cung cấp các tài nguyên thay thế để hỗ trợ tùy ứng với cấu hình thiết bị cụ thể. Ví dụ: tài nguyên chuỗi thay thế để giải quyết vấn đề bản địa, tài nguyên drawable thay thế để giải quyết vấn đề mật độ màn hình khác nhau. Khi chạy ứng dụng, Android chọn tập tài nguyên đúng dựa trên cấu hình thiết bị.

Để định nghĩa tập tài nguyên thay thế:

- Tên tập tin tài nguyên thay thế giống tên tập tin tài nguyên mặc định, vì chúng dùng như resource ID trong code.
- Đặt trong thư mục riêng, có tên dạng `<resource directory>-<configuration qualifier>`.

Ví dụ, các chuỗi cho ngôn ngữ bản địa Việt nam đặt trong `src/res/values-vn/strings.xml`

Bạn có thể chỉ định một hoặc nhiều `<configuration qualifier>` bằng cách thêm chúng vào tên thư mục, phân cách bởi dấu `-`. Ví dụ: `values-vn-land`

Thứ tự thêm vào như sau:

Cấu hình	Mô tả	Ví dụ
MCC/MNC	Mobile Country Code và Mobile Network Operator.	mcc319 mcc310-mnc004
Ngôn ngữ và vùng	Mã ngôn ngữ theo ISO 639-1 hoặc mã vùng theo ISO 3166-1-alpha-2	en fr-rCA
Hướng của layout	Trái sang phải (mặc định) hoặc phải sang trái (ví dụ ngôn ngữ Arabic).	ldltr (trái sang phải) ldrtl (phải sang trái)
Chiều rộng nhỏ nhất	Kích thước nhỏ nhất của chiều rộng. Không thay đổi khi xoay thiết bị.	sw480dp (handset) sw600dp (tablet)
Chiều rộng có sẵn	Chiều rộng màn hình có sẵn, tính bằng dp. Thay đổi khi xoay thiết bị.	w720dp
Chiều cao có sẵn	Chiều cao màn hình có sẵn, tính bằng dp. Thay đổi khi xoay thiết bị.	h720dp
Kích thước màn hình	Màn hình phân giải thấp QVGA, kích thước tối thiểu 320x426 dp. Màn hình phân giải trung bình HVGA, kích thước tối thiểu 320x4270 dp. Màn hình phân giải trung bình VGA, kích thước tối thiểu 480x640 dp. Màn hình phân giải trung bình HVGA, kích thước tối thiểu 720x960 dp.	small normal large xlarge
Tỷ lệ màn hình	Dựa trên tỷ lệ màn hình, một màn hình rộng như WQVGA, WVGA và FWVGA là long. Ngược lại, như QVGA, HVGA và VGA là not long.	long notlong
Hướng màn hình	Hướng của màn hình khi xoay thiết bị: đứng (portrait) hoặc ngang (landscape). Thay đổi khi xoay thiết bị.	port land
Chế độ UI	Để cầm trên xe. Để cầm trên bàn. Hiển thị như TV. Đeo như đồng hồ.	car desk television watch
Chế độ ngày/đêm	Chọn các theme cho ngày hoặc đêm.	night notnight
Mật độ màn hình	Mật độ thấp ~120 dpi. Mật độ trung bình ~160 dpi. Mật độ cao ~240 dpi. Mật độ rất cao ~320 dpi. Mật độ cực cao ~480 dpi. Mật độ cực cực ~640 dpi.	ldpi mdpi hdpi xhdpi xxhdpi xxxhdpi

	Dùng cho tài nguyên không cơ giắc. Mật độ màn hình TV ~213 dpi.	nodpi tvdpi
Màn hình cảm ứng	Thiết bị không có màn hình cảm ứng. Thiết bị có màn hình cảm ứng.	notouch finger
Bàn phím	Thiết bị có bàn phím lộ. Thiết bị có bàn phím ẩn và bàn phím phần mềm. Thiết bị có bàn phím phần mềm.	keyexposed keyshidden keyssoft
Thiết bị nhập cơ bản	Thiết bị không có bàn phím nhập. Thiết bị có bàn phím nhập phần cứng qwerty. Thiết bị có bàn phím nhập phần cứng 12-key.	nokeys qwerty 12key
Phím điều hướng	Phím điều hướng có sẵn hoặc không.	navexposed navhidden
Phương thức điều hướng cơ bản ngoài màn hình cảm ứng	Thiết bị không có điều hướng nào khác ngoài màn hình cảm ứng. Thiết bị có directional pad. Thiết bị có trackball. Thiết bị có directional wheel.	nonav dpad trackball wheel
Phiên bản platform	API Level được thiết bị hỗ trợ	v4

Assets

Một cách khác để đóng gói tài nguyên cho ứng dụng là dùng assets. Bạn đặt các tập tin tài nguyên (hình ảnh, âm thanh, ...) trong thư mục src/main/assets dưới dạng các tập tin chỉ đọc. Chú ý các tập tin này có tên là *chữ thường*.

Khác với các loại tài nguyên đã được trình bày phần trên, sẽ được biên dịch trong quá trình build; các tài nguyên assets được sao chép nguyên bản (tên tập tin, cấu trúc thư mục) đến thiết bị, gọi là tài nguyên không biên dịch.

Tài nguyên assets không truy cập bằng resource ID, chúng được truy cập thông qua tên tập tin gốc như sau:

- Dùng phương thức open() của content.res.AssetManager trả về stream nhập.
- Dùng stream nhập này cho các thao tác truy cập tiếp theo.

Ví dụ truy cập tập tin tài nguyên hình ảnh chứa trong thư mục /assets/angel.png

```
private static Bitmap getBitmapFromAsset(View rootView, String strName) throws IOException {
    AssetManager mAssetManager = rootView.getContext().getAssets(); // Context.getAssets()
    InputStream is = mAssetManager.open(strName); // lấy stream nhập từ tập tin tài nguyên
    Bitmap mBitmap = BitmapFactory.decodeStream(is); // dữ liệu stream được giải mã thành bitmap
    is.close();
    return mBitmap;
}

// dùng trong onCreateView() của fragment
ImageView mImage = (ImageView) rootView.findViewById(R.id.image);
try {
    Bitmap mBitmap = getBitmapFromAsset(rootView, "angel.png");
    if (mBitmap != null) {
        mImage.setVisibility(View.VISIBLE);
        mImage.setImageBitmap(mBitmap);
    }
} catch (IOException ioe) {
    mImage.setVisibility(View.GONE);
}
}
```

Phương thức setImageBitmap() không quan tâm đến độ phân giải màn hình như phương thức setImageResource(), bạn nên cấu hình trước, ví dụ dùng Bitmap.setDensity(DisplayMetrics.DENSITY_XHIGH).

Assets trở nên tiện dụng khi bạn đóng gói các trang web với HTML, CSS, JavaScript và hình ảnh. Bạn có thể đặt chúng trong thư mục assets, chúng sẽ giữ nguyên tên và cấu trúc thư mục. Sau đó, dùng thực thể android.webkit.WebView để tải chúng thông qua một URL có dạng file:///android_assets/<file_name>.

```
WebView mWebView = (WebView) findViewById(R.id.web_view);
mWebView.loadUrl("file:///android_assets/index.html");
```

Các đơn thể của ứng dụng

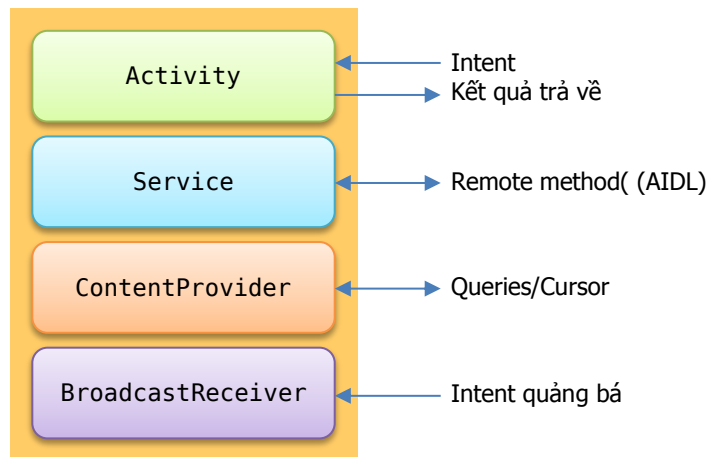
Các đơn thể của ứng dụng

1. Các đơn thể chính

Các đơn thể của ứng dụng là các khối xây dựng cơ bản trong một ứng dụng Android. Các đơn thể này được liên kết linh động bởi tập tin manifest của ứng dụng, tập tin `AndroidManifest.xml`. Tập tin này khai báo các đơn thể của ứng dụng và cách chúng tương tác với nhau.

Bạn cần phải hiểu rõ cách hoạt động của bốn đơn thể chính có thể được sử dụng trong một ứng dụng Android:

Đơn thể	Mô tả
Activity	Xử lý giao diện người dùng và tương tác giữa người dùng với màn hình.
Service	Xử lý các tiến trình chạy ngầm dưới nền liên quan với ứng dụng.
Content Provider	Xử lý truy cập dữ liệu và các vấn đề liên quan cơ sở dữ liệu.
Broadcast Receiver	Xử lý liên lạc giữa Android OS và các ứng dụng.



a) Activity

Một activity thể hiện một màn hình duy nhất, là giao diện mà người dùng tương tác. Ví dụ, một ứng dụng email có thể có một activity dùng hiển thị danh sách các email mới, một activity khác dùng soạn thảo email, và activity khác nữa dùng đọc email. Mặc dù các activity hoạt động phối hợp với nhau để hình thành giao diện người dùng gắn kết, nhưng từng activity là độc lập với activity khác. Nếu một ứng dụng có nhiều hơn một activity, thì một trong chúng sẽ được đánh dấu là activity sẽ hiển thị khi ứng dụng khởi chạy. Di chuyển từ một activity hiện hành đến một activity kế tiếp được thực hiện thông qua đối tượng Intent.

Một activity được cài đặt như một lớp con của lớp `android.app.Activity`.

b) Service

Một service là một đơn thể chạy dưới nền để thực hiện các tác vụ chiếm nhiều thời gian. Vì vậy, service không có giao diện người dùng. Ví dụ, một service có thể chơi nhạc dưới nền trong khi người dùng đang sử dụng một ứng dụng khác nhau, hoặc service lấy dữ liệu qua mạng mà không khóa tương tác của người dùng với một activity.

Có thể kết nối (bind) đến một service đang chạy hoặc khởi động một service nếu nó chưa chạy. Khi đã kết nối, từ code gọi service bạn có thể giao tiếp, trao đổi dữ liệu với service đó.

Một service được cài đặt như một lớp con của lớp `android.app.Service`.

c) Content Provider

Với Android, dữ liệu có thể được lưu trữ trong hệ thống tập tin, trong cơ sở dữ liệu SQLite hoặc tại một nơi khác trên mạng. Content provider cài đặt một tập các phương thức chuẩn cho phép lấy và lưu trữ dữ liệu. Ứng dụng không gọi các phương thức này một cách trực tiếp. Thay vào đó, ứng dụng dùng đối tượng thuộc lớp `ContentResolver` và gọi các phương thức thay thế của nó. Một content resolver có thể làm việc với kiểu content provider bất kỳ.

Content provider cho phép truy cập đến các kho dữ liệu dùng chung trên thiết bị Android: contacts, music, video, pictures.

Một content provider được cài đặt như một lớp con của lớp `android.content.ContentProvider` và cài đặt một tập phương thức chuẩn cho phép truy cập dữ liệu.

d) Broadcast Receiver

Broadcast receiver là đơn thể đáp ứng các thông điệp quảng bá (broadcast) đến từ các ứng dụng khác hoặc đến từ hệ thống. Ví dụ, ứng dụng có thể quảng bá để các ứng dụng khác biết rằng một số dữ liệu đã được tải về điện thoại và sẵn sàng cho các ứng dụng khác sử dụng, broadcast receiver sẽ chặn bắt các thông tin quảng bá này và sẽ có các hành động đáp ứng thích hợp. Broadcast receiver không có giao diện người dùng nhưng có thể khởi chạy một activity để đáp ứng thông tin nhận được, hoặc giống như service, nó có thể dùng notification để gửi cảnh báo đến người dùng.

Một broadcast receiver được cài đặt như một lớp con của lớp `android.content.BroadcastReceiver` và mỗi thông điệp được quảng bá như một đối tượng `android.content.Intent`.

2. Các đơn thể bổ sung

Các đơn thể bổ sung được sử dụng để xây dựng các đơn thể nêu trên, xây dựng các thao tác của chúng, và tạo hệ thống kết nối giữa chúng.

Các đơn thể này là:

Đơn thể	Mô tả
Fragment	Thể hiện một hành vi hoặc một phần của giao diện người dùng trong một activity.
View	Các thành phần giao diện người dùng được vẽ lên màn hình như button, list, form, v.v...
Layout	Xử lý cách bố trí màn hình và các view.
Intent	Đối tượng thông điệp dùng liên lạc (chuyển thông tin, điều khiển) giữa các đơn thể với nhau.
Resources	Tài nguyên cho ứng dụng như chuỗi, hằng và các đối tượng vẽ được (drawable, ví dụ hình ảnh).
AndroidManifest	Tập tin cấu hình cho ứng dụng.

Intent là đối tượng thông điệp bất đồng bộ để liên lạc giữa các đơn thể. Trong tập tin manifest, một đơn thể có một hay nhiều intent filter để báo cho Android biết các loại intent mà đơn thể đó xử lý.

3. Context

android.content.Context cung cấp giao diện chứa thông tin toàn cục về môi trường ứng dụng, nó thể hiện ngữ cảnh hiện tại của ứng dụng. Các đơn thể khác, thường khi vừa mới được tạo, sẽ cần tham chiếu Context để lấy thông tin về các đơn thể đang có sẵn.

Hiện thực của lớp abstract Context là ContextImpl được cung cấp bởi hệ thống, gọi là base context. Để lấy context, chúng ta thường gọi các phương thức của ContextWrapper, đây là một đối tượng đại diện (proxy) cho base context chứa bên trong nó. Các đơn thể quan trọng như Application, Activity và Service đều là lớp con của ContextWrapper, nên chúng có thể hành động như context cho các phần khác của ứng dụng.

Tùy thuộc vào loại đơn thể mà nó liên kết, Context lấy được thuộc các loại sau:

- Application context: thực thể duy nhất (singleton) gắn liền với vòng đời của ứng dụng, truy cập thông qua phương thức `getApplication()` từ activity/service hoặc `getApplicationContext()` từ ContextWrapper.
- Activity/service context: thực thể gắn liền vòng đời với activity/service. Nếu đang ở trong một activity/service, nó chính là `this`, do Activity và Service đều là lớp con của ContextWrapper. Nếu đang ở bên trong một lớp nội của activity/service, lấy Context bằng phương thức `getBaseContext()`.

Do từ Context bạn có thể truy cập đến rất nhiều đơn thể quản lý khác (AssetManager, ActivityManager, ...) nên Context là đơn thể quan trọng, được sử dụng thường xuyên để truy cập nhiều thông tin, thường trong các trường hợp sau:

- Tạo một đối tượng mới, như view, adapter, listener. Truyền Context để đối tượng mới có thể truy cập đến đơn thể cha.

```
TextView mTextView = new TextView(getContext()); // khi tạo view động từ code
Toast.makeText(getBaseContext(), "Toast Message", Toast.LENGTH_SHORT).show();
ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), ...);
```

- Truy cập các nguồn tài nguyên chung, các service hệ thống như LayoutInflater, SharedPreferences, các tập tin và thư mục riêng của ứng dụng. Nhiều phương thức hỗ trợ công việc này: `getAssets()`, `getResources()`, `getPackageManager()`, `getSharedPreferencesFile()`.

```
String greeting = getResources().getString(R.string.hello);
mContext.getSystemService(LAYOUT_INFLATER_SERVICE)
getApplicationContext().getSharedPreferences(...);
```

- Lấy các đơn thể không tường minh như content provider, broadcast receiver, intent.

```
getApplicationContext().getContentResolver().query(uri, ...);
```

Activity

1. Vòng đời activity

a) Các phương thức callback

Một activity thể hiện một *màn hình đơn* với layout (bố trí) giao diện người dùng. Ví dụ, một ứng dụng email có thể có một activity trình bày một danh sách các email mới, một activity khác dùng soạn thảo email, và một activity khác nữa dùng đọc email.

Nếu một ứng dụng có nhiều hơn một activity, thì một trong số activity đó được đánh dấu là activity sẽ hiển thị ứng dụng khi khởi chạy (MAIN) trong tập tin manifest. Từ activity hiện hành có thể khởi chạy các activity kế tiếp bằng cách dùng intent.

Android khởi động các ứng dụng của mình bằng cách gọi phương thức callback `onCreate()` của activity đầu tiên. Hình bên là sơ đồ vòng đời activity, cho thấy chuỗi các phương thức callback khởi chạy một activity và chuỗi các phương thức callback hủy bỏ một activity.

Vòng đời của activity có thể phân thành:

- Thời gian hiển thị (visible lifetime): giữa `onStart()` và `onStop()`. Trong thời gian này, người dùng nhìn thấy activity trên màn hình, mặc dù nó có thể không ở trên cùng hoặc người dùng không tương tác được.
- Thời gian chạy bề mặt (foreground lifetime): giữa `onResume()` và `onPause()`. Trong thời gian này, activity nằm ngay trên cùng và người dùng có thể tương tác với chúng.

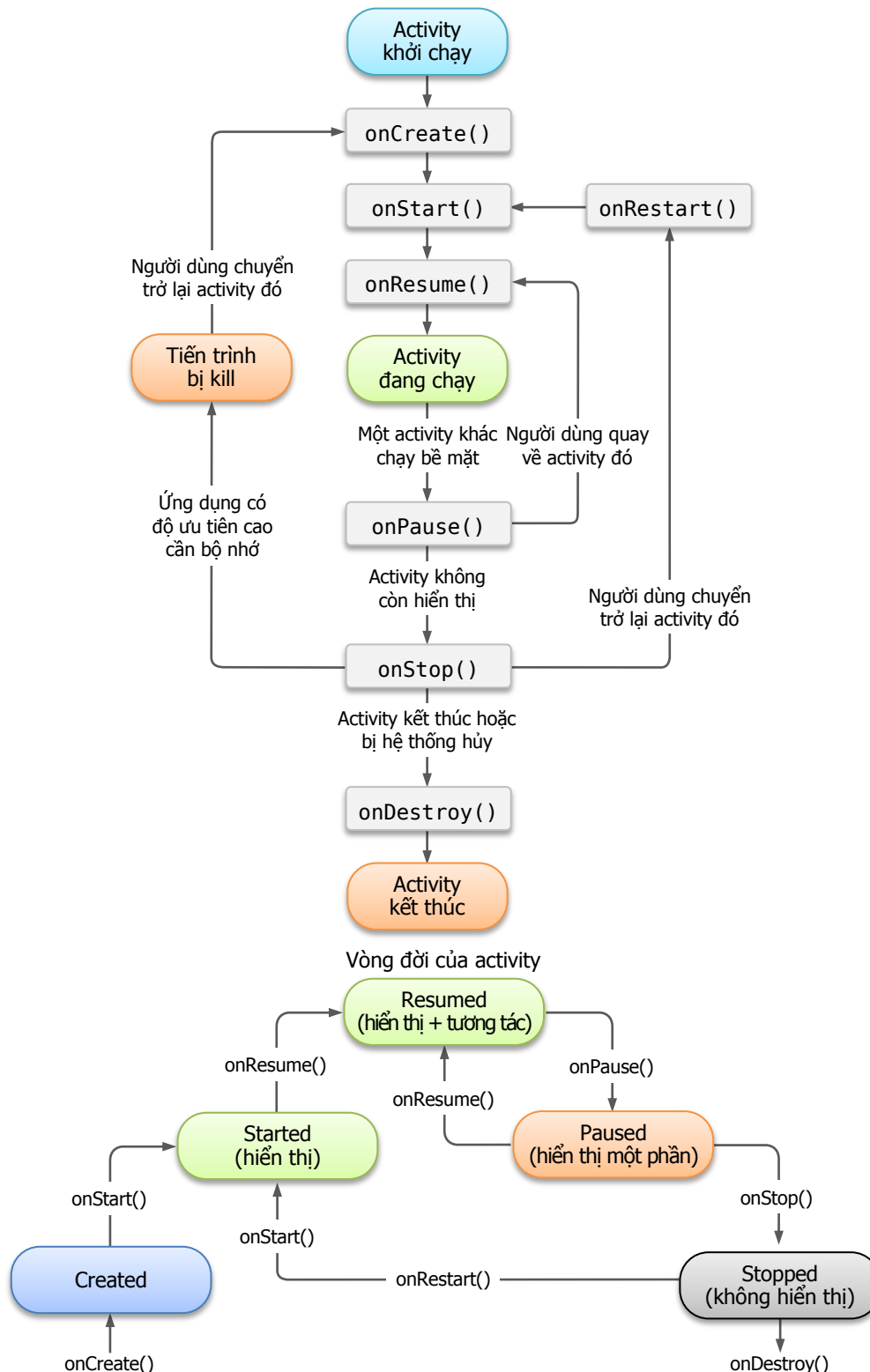
Vòng đời của ứng dụng không được điều khiển bởi bản thân nó mà được điều khiển bởi Android OS. Android OS có thể dùng hoặc hủy ứng dụng Android *đang chạy nền* một cách đột ngột, ví dụ để đáp ứng cuộc gọi đến hoặc đáp ứng với tình trạng thiếu bộ nhớ trầm trọng.

Bạn không cần cài đặt tất cả các phương thức callback mà có thể chọn lựa viết lại các phương thức callback cần thiết để móc (hook) vào vòng đời, thực hiện các công việc mong muốn khi Activity thay đổi trạng thái.

- Lớp Activity phải cài đặt phương thức `onCreate()` để thiết lập khởi tạo cho activity. Đây là vị trí thích hợp để "inflate" UI và thiết lập các listener.

- Phương thức `onResume()` và `onPause()` thường được dùng để bật/tắt những gì bạn muốn chạy trong lúc activity *nhìn thấy được*. Điều này quan trọng với các thiết bị tiêu thụ nhiều pin như GPS và các cảm biến. Với ứng dụng media, đây là nơi bật/tắt audio, video và animation. Khi activity bị hủy đột ngột, `onPause()` là cơ hội cuối để lưu những thay đổi, nên cơ sở dữ liệu thường được hoàn tất trong phương thức này.

- Phương thức `onStop()` và `onDestroy()` chỉ được gọi activity kết thúc một cách bình thường.



Các trạng thái của activity và các phương thức callback khi chuyển trạng thái

Các phương thức callback của activity được trình bày trong bảng sau:

Phương thức callback	Mô tả
<code>onCreate()</code>	Phương thức callback đầu tiên, được gọi khi activity được <i>tạo ra lần đầu</i> . Nhận một đối tượng Bundle chứa thông tin trạng thái activity trước đó. Theo sau luôn là <code>onStart()</code> . Phương thức này truyền một đối tượng Bundle chứa thông tin của activity trước.
<code>onStart()</code>	Được gọi khi activity <i>trở nên nhìn thấy được</i> (visible) bởi người dùng, nhưng chưa tương tác được. Theo sau là <code>onResume()</code> nếu activity chạy bề mặt hoặc <code>onStop()</code> nếu activity bị ẩn đi (hidden).
<code>onResume()</code>	Được gọi khi <i>người dùng bắt đầu tương tác</i> với activity của ứng dụng, activity đang trên đỉnh stack. Theo sau luôn là <code>onPause()</code> .
<code>onPause()</code>	Được gọi khi <i>người dùng không còn tương tác với activity</i> . Activity tạm dừng bị mất focus nhưng người dùng vẫn thấy nó (bị che một phần bởi activity trên đỉnh stack). Activity tạm tắt (sleep) không nhận dữ liệu nhập của người dùng và không thực hiện bất kỳ mã nào. Theo sau là <code>onResume()</code> nếu activity trở lại bề mặt hoặc <code>onStop()</code> nếu activity bị ẩn đi. Lúc này, activity cũng có thể bị hủy (kill) bởi hệ thống, tuy nhiên <code>onPause()</code> vẫn được gọi trước đó.

onStop()	Được gọi khi activity <i>không còn nhìn thấy được</i> . Có thể bị che phủ hoàn toàn do một activity khác đang đứng trên đỉnh stack. Theo sau là onRestart() nếu activity trở lại bề mặt, hoặc theo sau là onDestroy() nếu activity bị hủy (kill) bởi hệ thống khi bộ nhớ là cần thiết cho nơi khác.
onDestroy()	Được gọi khi activity <i>bị hủy</i> bởi hệ thống.
onRestart()	Được gọi khi activity <i>khởi động lại sau khi dừng</i> . Theo sau luôn là onStart().

b) Theo dõi vòng đời activity

Chúng ta tạo ra các tình huống trong vòng đời của activity để xem các phương thức callback chạy.

Trong MainActivity.java, chọn viết lại các phương thức onStart(), onResume(), onPause(), onStop(), onDestroy(), onRestart(), onSaveInstanceState() và onRestoreInstanceState(). Trong mỗi phương thức callback này, dùng Log.d() để ghi vào LogCat, giống như sau:

```
private static final String TAG = "LifeCycle";
Log.d(TAG, "activity::onDestroy()");
```

Trong Android Studio, mở Logcat (Alt+6), thêm filter:

Name : Activity Callback
by Log Tag : LifeCycle
by Log Level : Debug

Thực hiện các thao tác sau để đưa activity vào các trạng thái khác nhau, theo dõi các phương thức callback:

- Khởi động ứng dụng: onCreate → onStart → onResume
- Nhấn nút Home, ứng dụng sẽ tạm dừng: onPause → onSaveInstanceState → onStop
- Mở lại ứng dụng, ứng dụng sẽ chạy tiếp: onRestart → onStart → onResume

Trong AVD, vào Settings > Developer options, chọn Don't keep activities. Sau đó, khi nhấn nút Home, activity sẽ bị hủy luôn, bạn thấy thêm onDestroy sau onStop; và khi mở lại ứng dụng phải bắt đầu từ onStart, không phải từ onRestart.

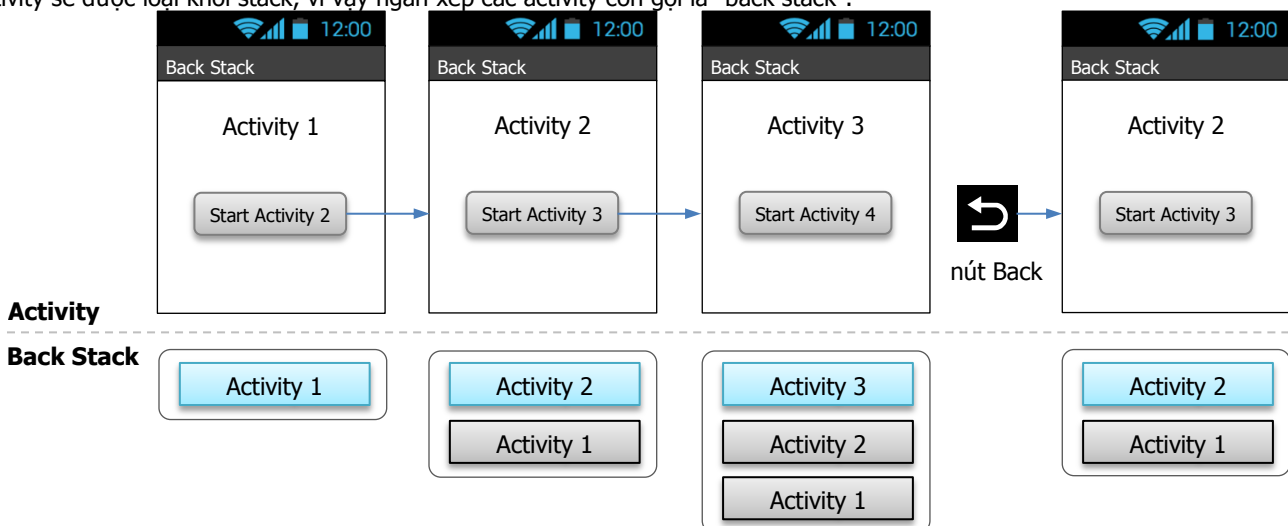
- Nhấn nút Back, ứng dụng sẽ đóng: onPause → onStop → onDestroy
- Khởi động lại ứng dụng: onCreate → onStart → onResume
- Quay thiết bị: onPause → onSaveInstanceState → onStop → onDestroy → onCreate → onStart → onRestoreInstanceState → onResume. Quay thiết bị trở lại: tương tự.

- Từ ADM giả lập cuộc gọi đến AVD, ứng dụng sẽ tạm dừng: onPause → onSaveInstanceState → onStop

Chọn hủy cuộc gọi, ứng dụng sẽ chạy tiếp: onRestart → onStart → onResume

2. Ngăn xếp các activity

Các activity được quản lý trong một ngăn xếp (activity stack), khi một activity mới được khởi động, nó được đặt trên đỉnh của stack và trở thành *running activity* (activity đang chạy). Những activity khác vẫn còn nằm trong stack và sẽ không được chạy ở bề mặt trong khi running activity vẫn còn trên đỉnh stack. Nếu người dùng nhấn nút ⏮ (nút Back) trên thiết bị thì running activity sẽ được loại khỏi stack, vì vậy ngăn xếp các activity còn gọi là "back stack".



Ngăn xếp Activity. Màu xanh là running activity.

3. Bundle

android.os.Bundle là một cơ chế đơn giản dùng truyền dữ liệu giữa các activity. Bundle là một tập hợp các cặp <key, value>. Bundle có một tập các phương thức:

- putXxx(), putXxxArray() lưu trữ hoặc mảng các trị có kiểu Xxx chỉ định vào Bundle. Phương thức nhận cặp <key, value> như tham số. Các phương thức putSerializable() và putParcelable() dùng lưu các đối tượng tùy biến.
- getXxx(), getXxxArray() lấy trị (hoặc mảng các trị) có kiểu chỉ định từ Bundle. Phương thức nhận tham số key và trả về trị được lưu trữ. Với các phương thức getSerializable() và getParcelable(), bạn phải ép kiểu trị trả về. Trị trả về sẽ bằng null nếu không so trùng kiểu.

Kiểu dữ liệu đặt vào Bundle có thể là Boolean, Double, String, v.v... và cả kiểu Bundle, hoặc mảng các trị có kiểu đó. Với dữ liệu có kiểu đối tượng tùy biến. Các lớp đối tượng đó phải cài đặt giao diện Serializable hoặc Parcelable.

- Serializable là interface chuẩn của Java, nó đơn giản "đánh dấu" rằng lớp đã cài đặt Serializable. Sau đó, trong tình huống nhất định, lớp cài đặt nó tự động được "chuỗi hóa", gọi là marshaling/unmarshaling. Cài đặt Serializable đơn giản nhưng chậm do dùng cơ chế reflection và tạo nhiều đối tượng rác.

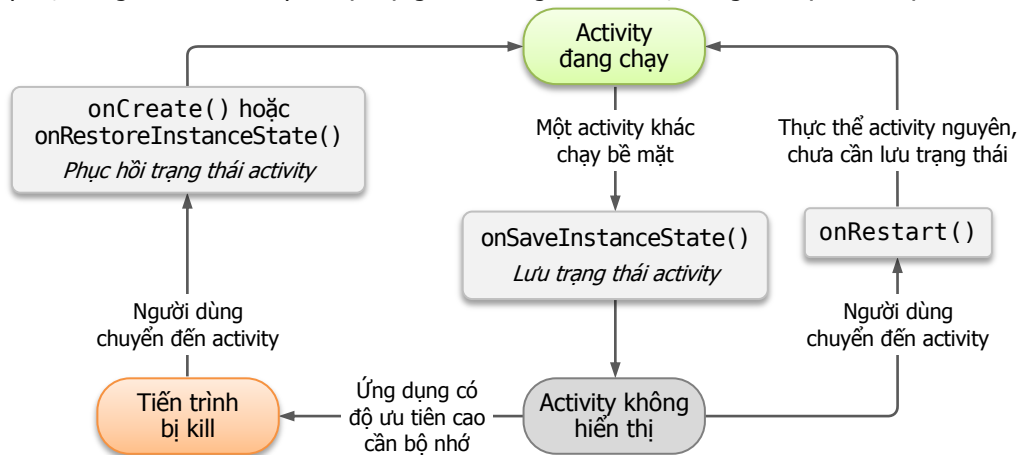
- Parcelable là interface của Android SDK, bạn phải tự tùy chỉnh marshaling/unmarshaling bằng cách cài đặt các phương thức của interface như writeToParcel() và thêm thành viên static Parcelable.Creator cho lớp được cài đặt. Cài đặt Parcelable cho một lớp phức tạp hơn nhưng hiệu suất "chuỗi hóa" cải thiện đáng kể, nhanh hơn nhiều.

```
Bundle bundle = new Bundle();
Bundle.putDouble("pi", 3.1415); // đặt trị vào Bundle với key "pi"
...
Double d = bundle.getDouble("pi"); // lấy trị có key "pi" từ Bundle
```

a) Lưu trạng thái trong vòng đời một activity

Trong một số tình huống, activity có thể bị hủy đột ngột làm mất thông tin hiện hành trên activity. Ví dụ:

- Khi người dùng thay đổi ngôn ngữ mặc định, thay đổi kích thước font chữ mặc định.
 - Khi người dùng nhấn nút Back.
 - Khi có tiến trình khác đột nhiên chạy bề mặt, ví dụ như có cuộc gọi đến, activity hiện hành bị che khuất và trong trạng thái đó nó có thể bị hủy bởi hệ thống nếu xảy ra thiếu bộ nhớ trầm trọng.
 - Khi xoay thiết bị, activity hiện hành bị hủy và tạo lại với layout thích hợp với chiều sử dụng hiện tại (portrait hoặc landscape).
- Khi mở lại activity cũ, thông tin trên activity trở lại trạng thái khởi gán ban đầu, thông tin hiện hành bị mất.



Lưu và phục hồi trạng thái cho activity

Giải pháp là lưu và phục hồi thông tin của activity như trước khi thay đổi, bằng cách dùng Bundle và hai phương thức callback (không mô tả trong sơ đồ vòng đời phần trên):

Phương thức callback	Mô tả
onSaveInstanceState(Bundle)	Được gọi trước onPause() hoặc trước onStop() (Android 3+), code lưu trữ thông tin cho activity đặt tại đây. Không gọi khi người dùng nhấn nút Back.
onRestoreInstanceState(Bundle)	Được gọi trước onStart(), code phục hồi thông tin cho activity đặt tại đây.

Phương thức onCreate() có tham số Bundle, sẽ là null nếu activity mới khởi chạy. Nếu activity bị hủy do lý do nào đó thì Bundle sẽ chứa thông tin trạng thái lần trước. Đây cũng chính là Bundle truyền cho hai phương thức trên.

Ví dụ sau đơn giản lưu văn bản đang nhập trong EditText, xoay màn hình để thấy văn bản vẫn bảo toàn trong EditText.

```
private static final String SAVED_KEY = "saved_value";
private EditText mEditText;

@Override protected void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    savedInstanceState.putCharSequence(SAVED_KEY, (CharSequence) mEditText.getText());
}

@Override protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    mEditText.setText(savedInstanceState.getCharSequence(SAVED_KEY));
}
```

b) Truyền thông tin giữa hai activity

Một activity thường thể hiện một UI trực quan duy nhất, trên đó một số hành động có thể được thực hiện. Di chuyển từ một activity này đến một activity khác được thực hiện bằng cách từ activity hiện tại gửi intent khởi động activity kế tiếp.

Activity thứ nhất thực hiện lời gọi startActivity() với tham số là đối tượng Intent, lời gọi được gửi đến ActivityManager của Android OS, ActivityManager sẽ tạo một Activity mới từ thông tin có trong intent và gọi phương thức onCreate() của activity mới này. Activity thứ hai cũng nhận được intent do activity thứ nhất gửi, trong đó có thể chứa một danh sách các tên (name) hoặc Bundle chứa các extras là dữ liệu cần chuyển tiếp.

Nếu bạn muốn nhận kết quả trả ngược về, có vài cách:

- Từ activity thứ hai, khi quay về ví dụ bằng nút Back, gửi ngược lại activity thứ nhất một intent với extras chứa kết quả.

- Dùng phương thức `startActivityForResult()` thay cho phương thức `startActivity()` khi gọi `ActivityManager`. Kết quả trả về được nhận bởi phương thức `callback onActivityResult()` của activity thứ nhất. Trong activity thứ hai, thực hiện trả kết quả về bằng phương thức `setResult()`.

Ví dụ MainActivity gửi intent khởi động SubActivity:

```
public class MainActivity extends Activity {
    Button mButton;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mButton = (Button) findViewById(R.id.button1);
        mButton.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, SubActivity.class);
                Bundle bundle = new Bundle();
                bundle.putInt("a", 123);
                bundle.putInt("b", 456);
                intent.putExtras(bundle);
                // 1122 là requestCode, phân biệt kết quả nhận từ nhiều activity con nếu có
                startActivityForResult(intent, 1122);
            }
        });
    }

    @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == 1122 && resultCode == RESULT_OK) {
            Bundle bundle = data.getExtras();
            Toast.makeText(this, "Result: " + bundle.getInt("a") + " + " + bundle.getInt("b") +
                " = " + bundle.getInt("sum"), Toast.LENGTH_LONG).show();
        }
    }
}
```

SubActivity nhận dữ liệu từ intent, tính toán, và trả kết quả về bằng phương thức `setResult()`.

```
public class SubActivity extends Activity {
    Bundle bundle;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        bundle = getIntent().getExtras();
        int a = bundle.getInt("a");
        int b = bundle.getInt("b");
        bundle.putInt("sum", a + b); // dùng lại bundle
        finish();
    }

    @Override public void finish() {
        Intent intentResult = new Intent();
        intentResult.putExtras(bundle);
        setResult(RESULT_OK, intentResult);
        super.finish();
    }
}
```

Khai báo của hai activity trong tập tin manifest, chú ý phần `<intent-filter>`:

```
<application android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity android:name=".MainActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".SubActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
```

Intent

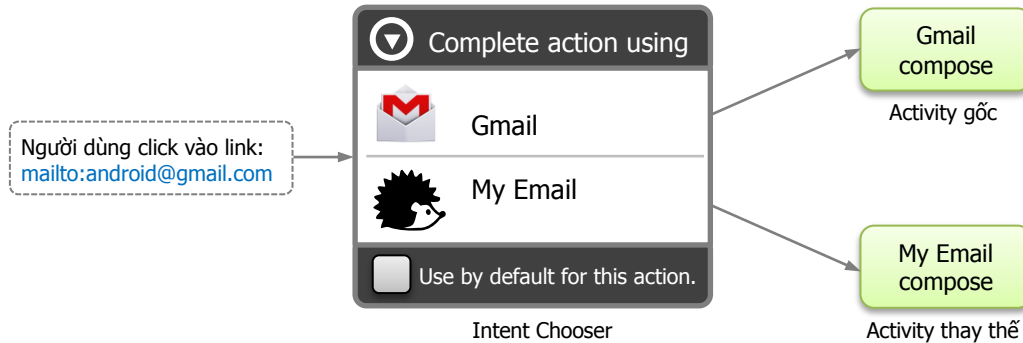
1. Đối tượng Intent

Đối tượng Intent được xem như một đối tượng thông điệp, gửi từ đơn thể này đến đơn thể khác, thể hiện *ý muốn* của đơn thể gửi intent muốn đơn thể nhận thực hiện. Ví dụ: "tôi muốn xem một bản ghi trong contact", "xin mở giùm tôi trang web này", "hãy hiển thị cho tôi màn hình Order Confirmation". Các đơn thể gửi/nhận intent nằm cùng trong một ứng dụng hoặc thuộc về các ứng dụng khác nhau, là một trong ba đơn thể chính của ứng dụng: activity, service và broadcast receiver.

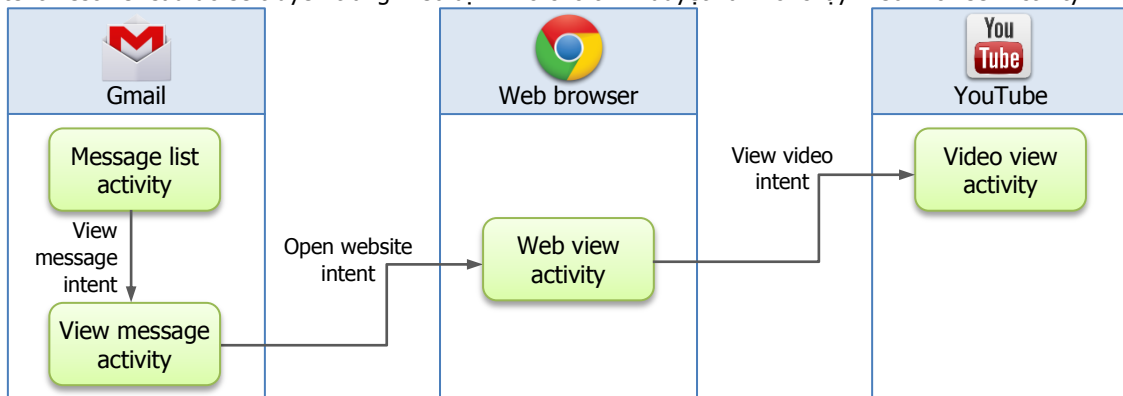
Intent có thể dùng để:

- Khởi động một cách tường minh một activity hoặc một service.
- Khai báo ý muốn của bạn để bắt đầu một activity hoặc một service (intent không tường minh).
- Quảng bá một sự kiện (event) đã xảy ra.

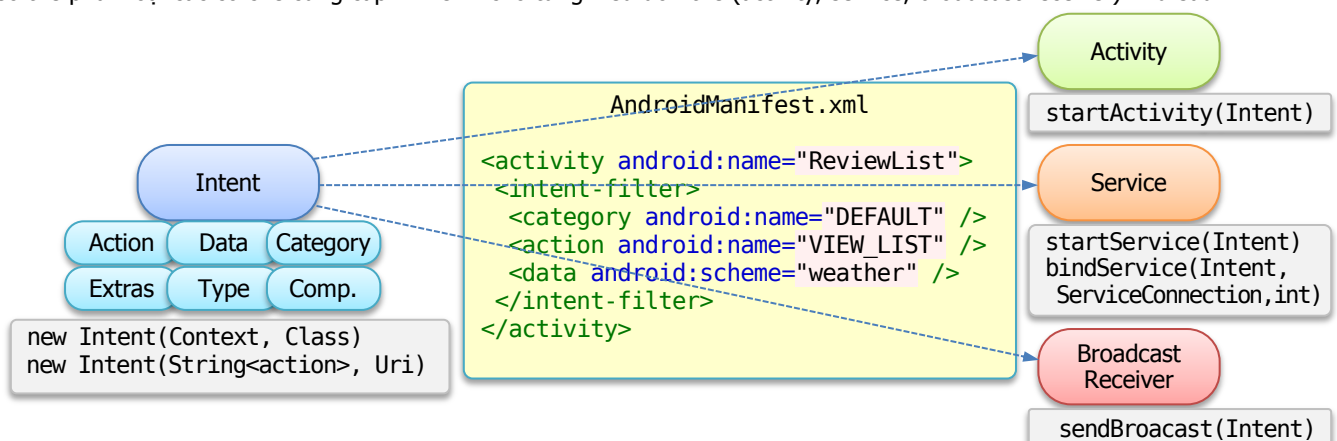
Ví dụ, giả sử đơn thể activity của ta muốn khởi động ứng dụng email (email client) và gửi email bằng cách dùng thiết bị Android. Như vậy, đơn thể activity của ta sẽ gửi intent ACTION_SEND kèm theo một *chooser* đến Intent Resolver, *chooser* này sẽ cung cấp giao diện để người dùng chọn cách gửi email.



Ví dụ khác, giả sử đơn thể activity của ta cần mở URL trong trình duyệt trên thiết bị Android. Như vậy, đơn thể activity sẽ gửi intent ACTION_WEB_SEARCH đến Intent Resolver để mở URL chỉ định trong trình duyệt. Intent Resolver sẽ phân tích một danh sách các activity và chọn trong chúng một activity phù hợp nhất với intent nhận được, trong trường hợp này là Web Browser Activity. Intent Resolver sau đó sẽ truyền trang Web định mở cho trình duyệt và khởi chạy Web Browser Activity.



Có thể phân loại các cơ chế cung cấp Intent cho từng kiểu đơn thể (activity, service, broadcast receiver) như sau:



Cơ chế	Mô tả
Context.startActivity() hoặc Activity.startActivityForResult()	Đối tượng intent được truyền như tham số đến phương thức này để khởi chạy một activity mới hoặc lấy một activity có sẵn để thực hiện tác vụ mới nào đó.
Context.startService()	Đối tượng intent được truyền như tham số đến phương thức này để khởi chạy một service hoặc cung cấp các chỉ thị mới đến service đang chạy nền.
Context.bindService()	Đối tượng intent được truyền để hình thành một kết nối giữa đơn thể gọi và service đích. Nó có thể tùy chọn khởi chạy service nếu service không chạy sẵn.
Context.sendBroadcast()	Đối tượng intent được truyền như tham số đến phương thức này để cung cấp thông điệp cho tất cả các broadcast receiver quan tâm.

Một khi intent được tạo và lưu chuyển, Android phải phân giải intent để tìm ra ứng viên nhận intent. Có hai kiểu intent được gửi, phân biệt bởi cách chúng được phân giải và lưu chuyển: intent không tường minh và intent tường minh.

a) Intent không tường minh (implicit)

Trong các intent này, tên tường minh của đơn thể mục tiêu không có (trường tên component để trống). Chúng thường dùng để kích hoạt các đơn thể thuộc ứng dụng khác (ví dụ khởi động một service) hoặc dùng như intent quảng bá.

Intent không tường minh chỉ được phân phối đến đơn thể đích nếu đơn thể đó có khai báo intent filter chặn xử lý nó. Để khai báo intent filter, trong tập tin manifest, tìm element khai báo đơn thể đích, và khai báo loại intent cần chặn xử lý trong element con <intent-filter>.

Lưu ý là do có thể có nhiều đơn thể cùng khai báo rằng nó có khả năng xử lý một loại intent chỉ định, Android OS sẽ xem xét các đơn thể tiềm năng, người dùng sẽ được nhắc chọn đơn thể thích hợp và intent sẽ được gửi đến đơn thể đó.

Ví dụ:

- Tạo một project với activity chứa WebView, activity này nhận intent không tường minh, trích lấy URL từ nó và chuyển cho WebView để tải trang web tại URL về:

```
public class WebViewActivity extends ActionBarActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web_view);
        Intent intent = getIntent();
        Uri data = intent.getData();
        URL url = null;
        try {
            url = new URL(data.getScheme(), data.getHost(), data.getPath());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        WebView webView = (WebView) findViewById(R.id.webView);
        webView.loadUrl(url.toString());
    }
}
```

Tập tin manifest của project phải khai báo quyền truy cập Internet:

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

Khai báo của WebViewActivity trong tập tin manifest này cũng chứa <intent-filter> để lọc intent không tường minh:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
</intent-filter>
```

Project này không có activity khởi chạy từ đầu, nên khi chạy, Android Studio sẽ hiển thị hộp thoại Edit configuration, yêu cầu chỉ dẫn cách tiến hành khi ứng dụng đã cài đặt. Trong phần Activity, bạn chọn Do not launch Activity.

- Tạo một project khác, có nút nhấn gửi đi một intent không tường minh:

```
// xử lý sự kiện nhấn nút
public void showWebPage(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.trainingwithexperts.com"));
    startActivity(intent);
}
```

Khi ứng dụng gửi intent, Android OS tìm thấy hai activity có intent filter bắt được intent trên: WebViewActivity và Browser (cấu hình là trình duyệt mặc định). Một cửa sổ hiển thị đề nghị bạn chọn activity sẽ nhận intent.

b) Intent tường minh (explicit)

Các intent chỉ định đơn thể mục tiêu của chúng bằng tên (lớp Java), thường dùng bên trong ứng dụng. Ví dụ activity khởi động một service cấp dưới hoặc khởi động một activity khác. Activity đích được chỉ định cụ thể bằng tên nên không cần sự lựa chọn của Android OS.

```
// tham số this là context gửi intent đi, trong trường hợp này là activity gửi
Intent intent = new Intent(this, TargetActivity.class);
intent.putExtra("Key1", "ABC");
intent.putExtra("Key2", "123");
startActivity(intent);
```

Nếu đơn thể không khai báo một intent filter nào, nó chỉ có thể được kích hoạt bởi một intent tường minh.

2. Các thành phần của đối tượng Intent

Để thể hiện ý muốn, đối tượng Intent giữ một cấu trúc dữ liệu, mô tả tóm tắt về tác vụ muốn được thực hiện cho đơn thể nhận intent. Các thông tin lưu trong cấu trúc dữ liệu này gọi là các thành phần của đối tượng Intent. Tùy theo kiểu liên lạc hoặc tác vụ cần thực hiện, bạn chọn thiết lập các thành phần cần thiết. Trong đó, quan trọng nhất là:

- Action, chỉ định hành động muốn được thực hiện.

- Data, chỉ định dữ liệu sẽ được xử lý.

Khi đơn thể mục tiêu nhận intent, nó dễ dàng rút trích thông tin chứa trong intent nhận được:

```

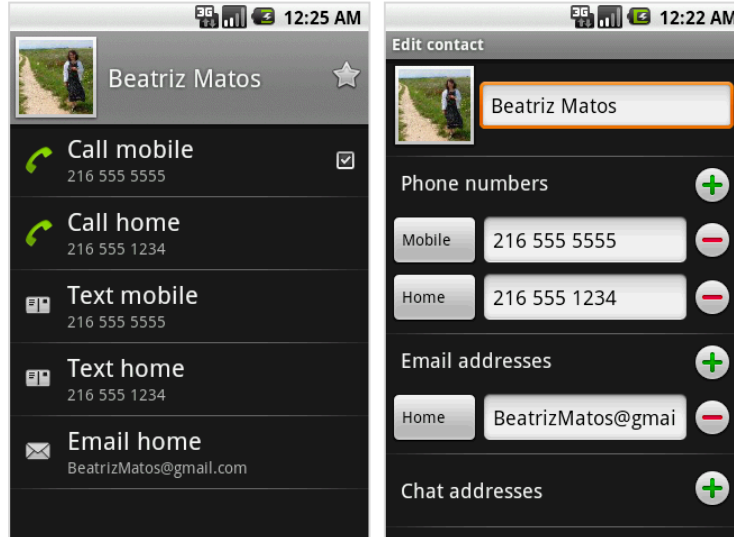
Intent intent = getIntent();           // Activity.getIntent()
Uri data = intent.getData();
if (intent.getAction().equals(Intent.ACTION_VIEW)) {
    // view action
} else if (intent.getAction().equals(Intent.ACTION_EDIT)) {
    // edit action
}

```

Các thông tin thứ cấp tùy chọn khác là: Category, Component, Type và Extras.

a) Action

Đây là thành phần bắt buộc của đối tượng Intent. Action xác định phần lớn các thành phần còn lại của đối tượng Intent.



Action: trái, dùng ACTION_VIEW; phải, dùng ACTION_EDIT. Data: "content://contacts/people/2";

Các action built-in được định nghĩa như các hằng trong lớp Intent. Bảng sau liệt kê các action built-in quan trọng:

Action	Mô tả
ACTION_MAIN	Bắt đầu như điểm vào chính, không nhận dữ liệu.
ACTION_VIEW	Hiển thị dữ liệu đến người dùng.
ACTION_ATTACH_DATA	Dùng chỉ ra rằng một số phần của Data cần gắn liền một số vị trí.
ACTION_EDIT	Cung cấp truy cập cho phép chỉnh sửa tường minh đến phần Data chỉ định.
ACTION_CHOOSER	Hiển thị một activity chọn, cho phép người dùng chọn một mục trước khi xử lý.
ACTION_GET_CONTENT	Cho phép người dùng chọn một loại dữ liệu cụ thể và trả nó về.
ACTION_DIAL	Hiển thị phone dialer với số được chỉ định bởi phần Data, như: Uri.parse("tel:555-1234").
ACTION_CALL	Thực hiện một lời gọi lập tức đến đối tượng được chỉ định bởi phần Data.
ACTION_SEND	Gửi dữ liệu đến người nào đó.
ACTION_SENDTO	Gửi tin nhắn đến người nào đó, chỉ định bởi phần Data, như: Uri.parse("sms:5551234").
ACTION_ANSWER	Xử lý một lời gọi đến.
ACTION_INSERT	Chèn mục trống vào container chỉ định.
ACTION_DELETE	Xóa dữ liệu chỉ định.
ACTION_RUN	Chạy phần Data.
ACTION_SYNC	Thực hiện một đồng bộ dữ liệu.
ACTION_SEARCH	Thực hiện một tìm kiếm.
ACTION_WEB_SEARCH	Thực hiện một tìm kiếm trên Web.
ACTION_FACTORY_TEST	Điểm vào chính cho kiểm tra.
ACTION_TIME_CHANGED	Đã thiết lập thời gian.
ACTION_TIMEZONE_CHANGED	Thay đổi timezone.
ACTION_BOOT_COMPLETED	Quảng bá một lần, sau khi hệ thống kết thúc khởi động (boot).
ACTION_POWER_CONNECTED	Nguồn điện bên ngoài đã được kết nối đến thiết bị.
ACTION_BATTERY_CHANGED	Quảng bá sticky chứa thông tin sạc, cấp độ, và các thông tin khác về pin.
ACTION_SHUTDOWN	Thiết bị đang tắt.
ACTION_ALL_APPS	Liệt kê tất cả các ứng dụng có sẵn trên thiết bị.
ACTION_BATTERY_LOW	Quảng bá này tương đương với hộp thoại hệ thống "Low battery warning".
ACTION_BATTERY_OKAY	Sẽ được gửi sau ACTION_BATTERY_LOW khi pin đã trở lại trạng thái đủ.
ACTION_BUG_REPORT	Mở activity để thông báo lỗi.
ACTION_CALL_BUTTON	Người dùng nhấn nút Call hoặc giao diện tương đương để đặt cuộc gọi.
ACTION_CAMERA_BUTTON	Người dùng nhấn nút Camera.
ACTION_CONFIGURATION_CHANGED	Bảo cấu hình hiện hành của thiết bị (orientation, local, v.v...) đã thay đổi.
ACTION_DATE_CHANGED	Bảo ngày đã thay đổi.

ACTION_DEFAULT	Tương đương ACTION_VIEW, action chuẩn được thực hiện trên phần Data.
ACTION_DEVICE_STORAGE_LOW	Quảng bá sticky chỉ tình trạng thiếu bộ nhớ trên thiết bị.
ACTION_DEVICE_STORAGE_OK	Chỉ tình trạng thiếu bộ nhớ trên thiết bị không còn nữa.
ACTION_DOCK_EVENT	Quảng bá sticky cho việc thay đổi trạng thái docking (gắn vào đế) của thiết bị.
ACTION_DREAMING_STARTED	Gửi sau khi hệ thống bắt đầu dreaming (tương tự screensaver).
ACTION_DREAMING_STOPPED	Gửi sau khi hệ thống ngừng dreaming.
ACTION_GTALK_SERVICE_CONNECTED	Kết nối Google Talk đã thành lập.
ACTION_GTALK_SERVICE_DISCONNECTED	Kết nối Google Talk đã ngắt.
ACTION_HEADSET_PLUG	Cắm hoặc rút tay nghe khỏi thiết bị.
ACTION_INPUT_METHOD_CHANGED	Thay đổi phương thức nhập.
ACTION_INSERT_OR_EDIT	Chọn một mục có sẵn, hoặc chèn vào một mục mới, rồi chỉnh sửa nó.
ACTION_INSTALL_PACKAGE	Khởi động cài đặt thiết bị.
ACTION_LOCALE_CHANGED	Thay đổi local (vùng, ngôn ngữ) hiện hành của thiết bị.
ACTION_MEDIA_BUTTON	Người dùng nhấn nút Play (media).
ACTION_MEDIA_CHECKING	Media bên ngoài đang có, và đang được kiểm tra đĩa.
ACTION_MEDIA_EJECT	Người dùng muốn loại bỏ thiết bị lưu trữ media bên ngoài.
ACTION_MEDIA_REMOVED	Media bên ngoài đã được loại bỏ.
ACTION_NEW_OUTGOING_CALL	Một cuộc gọi đi đã được đặt.
ACTION_PASTE	Tạo một mục mới trong container chỉ định, khởi tạo nó từ nội dung của clipboard.
ACTION_REBOOT	Có khởi động lại (reboot) thiết bị. Chỉ dùng bởi mã hệ thống.
ACTION_SCREEN_OFF	Gửi sau khi màn hình tắt.
ACTION_SCREEN_ON	Gửi sau khi màn hình bật.
ACTION_SEND_MULTIPLE	Cung cấp nhiều dữ liệu cho người khác.
ACTION_SET_WALLPAPER	Hiện thị thiết lập cho việc chọn wallpaper.
ACTION_VOICE_COMMAND	Bắt đầu lệnh Voice.
ACTION_WALLPAPER_CHANGED	Thay đổi wallpaper của hệ thống hiện hành.

Action trong một đối tượng Intent có thể được thiết lập bằng phương thức `setAction()` và đọc bằng `getAction()`.

Ví dụ về một số action chuẩn:

- Mở trang youtube.

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.youtube.com"));
startActivity(intent);
```

- Nghe tập tin nhạc trên card SD:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
Uri data = Uri.parse("file:///sdcard/Music/ChieuBenKiaSong.mp3");
String type = "audio/mp3";
intent.setDataAndType(data, type);
startActivity(intent);
```

- Tìm kiếm trên web:

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH).putExtra(SearchManager.QUERY, "android tutorials");
startActivity(intent);
```

- Gọi cho số chỉ định:

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:555-1234"));
startActivity(intent);
```

cần khai báo quyền thực thi trong tập tin manifest:

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

- Cung cấp geoCode của một vị trí:

```
String geoCode = "geo:0,0?q=quang+trung+software+city";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(geoCode));
startActivity(intent);
```

cần khai báo quyền thực thi trong tập tin manifest:

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

- Mở phần thiết lập hệ thống:

```
Intent intent = new Intent(android.provider.Settings.ACTION_SETTINGS);
startActivity(intent);
```

b) Data

Data thường là URI của dữ liệu sẽ được xử lý bởi Action. Ví dụ, nếu Action là ACTION_EDIT, thì Data sẽ chứa URI của tài liệu để hiển thị và chỉnh sửa. Ví dụ: "tel:/216 555-1234", http://maps.google.com.

Dùng phương thức `setData()` để chỉ định URI, dùng phương thức `setType()` để chỉ định kiểu MIME; và phương thức `setDataType()` nếu chỉ định cả hai. Ngược lại, dùng `getData()` để lấy URI và dùng `getType()` để lấy kiểu MIME.

Cặp Action/Data cho thấy một "ý muốn" đầy đủ. Một số ví dụ:

STT	Mô tả cặp Action/Data
1	ACTION_VIEW content://contacts/people/1 Hiển thị thông tin cá nhân trong contact list có định danh "1".

2	<code>ACTION_DIAL content://contacts/people/1</code> Gọi đến cá nhân có định danh "1" trong contact list.
3	<code>ACTION_VIEW tel:123</code> Hiển thị thông tin số điện thoại 123.
4	<code>ACTION_DIAL tel:123</code> Gọi số điện thoại 123.
5	<code>ACTION_EDIT content://contacts/people/1</code> Biên soạn thông tin cá nhân trong contact list có định danh "1".
6	<code>ACTION_VIEW content://contacts/people/</code> Hiển thị danh sách các cá nhân trong contact list. Chọn một cá nhân cụ thể để xem sẽ dùng intent khác.
7	<code>ACTION_VIEW http://www.google.com</code> Mở trang Google trong trình duyệt.

c) Category

Category là phần tùy chọn của đối tượng Intent. Nó là một chuỗi chứa thông tin bổ sung về loại đơn thể sẽ xử lý intent, thể loại (category) của đơn thể mục tiêu hoặc chỉ thị về cách khởi động activity đích. Phương thức `addCategory()` đặt một category vào trong đối tượng Intent, phương thức `removeCategory()` xóa một category thêm vào trước đó. Phương thức `getCategories()` trả về một tập các Category hiện có trong đối tượng Intent.

Bảng sau liệt kê một số category chuẩn quan trọng.

Category	Mô tả
<code>CATEGORY_APP_BROWSER</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy trình duyệt.
<code>CATEGORY_APP_CALCULATOR</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Calculator.
<code>CATEGORY_APP_CALENDAR</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Calendar.
<code>CATEGORY_APP_CONTACTS</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Contacts.
<code>CATEGORY_APP_EMAIL</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Email.
<code>CATEGORY_APP_GALLERY</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Gallery.
<code>CATEGORY_APP_MAPS</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Maps.
<code>CATEGORY_APP_MARKET</code>	Activity này cho phép người dùng duyệt và tải về ứng dụng mới.
<code>CATEGORY_APP_MESSAGING</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Messaging.
<code>CATEGORY_APP_MUSIC</code>	Sử dụng với <code>ACTION_MAIN</code> để khởi chạy ứng dụng Music.
<code>CATEGORY_BROWSABLE</code>	Các activity có thể triệu gọi an toàn trình duyệt hỗ trợ Category này.
<code>CATEGORY_CAR_DOCK</code>	Một activity sẽ chạy khi thiết bị được gắn vào car dock (để gắn trên xe).
<code>CATEGORY_CAR_MODE</code>	Sử dụng để chỉ định activity này có thể được dùng trong môi trường xe.
<code>CATEGORY_DEFAULT</code>	Thiết lập nếu activity là một tùy chọn cho hành động mặc định.
<code>CATEGORY_DESK_DOCK</code>	Một activity sẽ chạy khi thiết bị được chèn vào desk dock (để gắn để bàn).
<code>CATEGORY_DEVELOPMENT_PREFERENCE</code>	Activity này là một development preference panel.
<code>CATEGORY_EMBED</code>	Có khả năng chạy bên trong một activity cha.
<code>CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST</code>	Được sử dụng khi code trong kiểm thử thiết bị.
<code>CATEGORY_HE_DESK_DOCK</code>	Một activity sẽ chạy khi thiết bị được chèn vào một dock digital (HE: high end).
<code>CATEGORY_HOME</code>	Home activity, activity đầu tiên hiển thị khi thiết bị khởi động.
<code>CATEGORY_INFO</code>	Cung cấp thông tin về gói.
<code>CATEGORY_LAUNCHER</code>	Sẽ được hiển thị khi khởi chạy cấp cao nhất.
<code>CATEGORY_LE_DESK_DOCK</code>	Một activity sẽ chạy khi thiết bị được chèn vào một dock analog (LE: low end).
<code>CATEGORY_MONKEY</code>	Activity được sử dụng bởi monkey hoặc công cụ kiểm thử tự động khác.
<code>CATEGORY_OPENABLE</code>	Sử dụng để chỉ định rằng intent <code>GET_CONTENT</code> chỉ muốn URI được mở với <code>ContentResolver.openInputStream</code> .
<code>CATEGORY_PREFERENCE</code>	Activity này là một preference panel.
<code>CATEGORY_TAB</code>	Sử dụng khi tab trong một <code>TabActivity</code> .
<code>CATEGORY_TEST</code>	Được sử dụng khi kiểm thử.
<code>CATEGORY_UNIT_TEST</code>	Được sử dụng trong unit test.

d) Extras

Extras là các cặp key-value cung cấp thông tin thêm sẽ chuyển đến đơn thể nhận intent. Extras có thể được thiết lập và đọc bằng cách dùng các phương thức `putExtra()`, `putExtras()` và `getExtras()` tương ứng.

Bảng sau liệt kê một số dữ liệu extras chuẩn cho intent.

Extras	Mô tả
<code>EXTRA_ALARM_COUNT</code>	Extras kiểu <code>int</code> trong intent <code>AlarmManager</code> , báo cho ứng dụng được triệu gọi, số cảnh báo đang chờ xử lý được intent chuyển đến.
<code>EXTRA_ALLOW_MULTIPLE</code>	Dùng chỉ rằng intent <code>ACTION_GET_CONTENT</code> cho phép người dùng chọn và trả về nhiều mục chọn (item).
<code>EXTRA_ALLOW_REPLACE</code>	Extras kiểu <code>boolean</code> dùng với intent <code>ACTION_INSTALL_PACKAGE</code> để cài đặt gói.
<code>EXTRA_BCC</code>	Mảng chuỗi lưu các địa chỉ email sẽ dùng trong BCC (blind carbon copied).
<code>EXTRA_CC</code>	Mảng chuỗi lưu các địa chỉ email sẽ dùng trong CC (carbon copied).
<code>EXTRA_CHANGED_COMPONENT_NAME_LIST</code>	Extras của intent <code>ACTION_PACKAGE_CHANGED</code> , chứa mảng chuỗi của các component đã thay đổi.
<code>EXTRA_DATA_REMOVED</code>	Extras kiểu <code>boolean</code> dùng với intent <code>ACTION_PACKAGE_REMOVED</code> cho biết gói được gỡ cài đặt một phần hay hoàn toàn.

EXTRA_DOCK_STATE	Extras kiểu int dùng với intent ACTION_DOCK_EVENT yêu cầu trạng thái dock (gắn vào đế).
EXTRA_DOCK_STATE_CAR	Extras kiểu int dùng với intent EXTRA_DOCK_STATE để thể hiện phone đang trong trạng thái car dock (để gắn trên xe).
EXTRA_DOCK_STATE_DESK	Extras kiểu int dùng với intent EXTRA_DOCK_STATE để thể hiện phone đang trong trạng thái desk dock (để gắn đế bàn).
EXTRA_EMAIL	Mảng chuỗi lưu các địa chỉ email đích.
EXTRA_HTML_TEXT	Chuỗi hằng liên kết với intent ACTION_SEND để cung cấp văn bản định dạng HTML đến EXTRA_TEXT.
EXTRA_INTENT	Intent mô tả lựa chọn bạn muốn trình bày với ACTION_PICK_ACTIVITY.
EXTRA_KEY_EVENT	Đối tượng KeyEvent chứa sự kiện kích hoạt tạo intent mà nó ở trong.
EXTRA_LOCAL_ONLY	Chỉ rằng intent ACTION_GET_CONTENT chỉ trả về dữ liệu trên thiết bị cục bộ.
EXTRA_ORIGINATING_URI	Extras là URI dùng với intent ACTION_INSTALL_PACKAGE và ACTION_VIEW để chỉ định URI của local APK trong Data của intent.
EXTRA_PHONE_NUMBER	Chuỗi lưu số điện thoại được nhập trong ACTION_NEW_OUTGOING_CALL, hoặc số điện thoại thực sự được gọi trong ACTION_CALL.
EXTRA_SHORTCUT_ICON	Tên của extras được dùng để định nghĩa icon, như một Bitmap, của một shortcut.
EXTRA_SHORTCUT_INTENT	Tên của extras được dùng để định nghĩa intent của một shortcut.
EXTRA_SHORTCUT_NAME	Tên của extras được dùng để định nghĩa tên của một shortcut.
EXTRA_STREAM	URI lưu một stream dữ liệu liên kết với intent, dùng với ACTION_SEND để cung cấp dữ liệu đang được gửi.
EXTRA_SUBJECT	Một chuỗi hằng chứa trong dòng subject của thông điệp.
EXTRA_TEMPLATE	Dữ liệu khởi tạo được đặt trong một record vừa tạo, dùng với intent ACTION_INSERT.
EXTRA_TEXT	Chuỗi hằng liên kết với intent, dùng với ACTION_SEND hỗ trợ gửi dữ liệu chuỗi.
EXTRA_TITLE	Tiêu đề của hộp thoại cung cấp cho người dùng khi dùng với intent ACTION_CHOOSER.
EXTRA_UID	Extras kiểu int trong các intent ACTION_UID_REMOVED, cung cấp uid của gói được gán.

Extras thường kèm theo intent không tường minh.

Ví dụ: thực hiện tìm kiếm trên web với từ khóa "android intents".

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
intent.putExtra(SearchManager.QUERY, "android intents");
startActivity(intent);
```

gửi SMS có nội dung "Remember to buy Nexus" đến 555-1234.

```
Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.parse("sms:5551234"));
intent.putExtra("sms_body", "Remember to buy Nexus");
startActivity(intent);
```

hiển thị Pictures.

```
Intent intent = new Intent();
intent.setType("image/pictures/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivity(intent);
```

e) Flags

Khi activity được khởi động bởi intent:

- Một thực thể của activity được tạo.
- Thực thể này được đặt trên đỉnh back stack.
- Activity được bắt đầu và đưa lên nền.

Các bước được mô tả ở trên có thể được thay đổi tùy biến bằng cách dùng Flags (tùy chọn) cho intent. Chỉ định Flags thích hợp trên đối tượng intent trước khi truyền intent đến phương thức startActivity().

```
intent.addFlag(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```

Flags chỉ cho hệ thống Android cách tùy biến khởi động activity và cách xử lý nó sau khi khởi động.

Flags	Mô tả
FLAG_ACTIVITY_REORDER_TO_FRONT	Đẩy một thực thể của activity được gọi (nếu có) trong stack lên bề mặt, thay vì tạo mới.
FLAG_ACTIVITY_BROUGHT_TO_FRONT	Cờ được thiết lập bởi hệ thống cho thấy activity tồn tại với chế độ khởi động singleTask đã được gọi và dùng lại.
FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET	Đánh dấu activity được gọi và tất cả các activity trên nó đều bị xóa (loại bỏ khỏi stack) khi tác vụ thiết lập lại, thiết lập lại này được thực hiện ngay sau khi tác vụ được đưa lên bề mặt.
FLAG_ACTIVITY_RESET_TASK_IF_NEEDED	Cờ được thiết lập bởi hệ thống, khi người dùng lại tiếp tục một tác vụ từ màn hình Home, menu chính hoặc từ danh sách các activity thực hiện gần nhất (recently launched).
FLAG_ACTIVITY_CLEAR_TOP	Đẩy một thực thể của activity được gọi (nếu có) trong stack lên bề mặt, thay vì tạo mới. Ngoài ra, các activity nằm trên nó trong stack đều bị xóa khỏi stack.

FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS	Khi chạy một activity trong một tác vụ mới, activity đó sẽ có mặt trong danh sách các activity thực hiện gần nhất (truy cập bằng nhấn giữ nút Home). Cờ này dùng ngăn chặn việc đưa activity vào danh sách đó.
FLAG_ACTIVITY_FORWARD_RESULT	A gọi B bằng <code>startActivityForResult()</code> , thay vì B trả kết quả bằng <code>setResult()</code> , B chuyển tiếp kết quả cho C bằng <code>startActivity()</code> kèm theo cờ này rồi <code>finish()</code> . C dùng <code>setResult()</code> và <code>finish()</code> để trả kết quả về cho <code>onActivityResult()</code> của A.
FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY	Cờ này được thiết lập bởi hệ thống, chỉ rằng activity đã được khởi động từ danh sách các activity thực hiện gần nhất.
FLAG_ACTIVITY_MULTIPLE_TASK	Cờ này không có hiệu lực, trừ phi kết hợp với cờ FLAG_ACTIVITY_NEW_TASK, nếu dùng kết hợp, một tác vụ mới được tạo ra và một thực thể mới của activity được gọi trở thành thành phần đầu tiên của tác vụ đó.
FLAG_ACTIVITY_NEW_TASK	Nếu dùng riêng, một tác vụ mới chỉ được tạo ra nếu không có sẵn. Nếu có sẵn, activity được đưa vào tác vụ đó thay vì tạo mới.
FLAG_ACTIVITY_NO_ANIMATION	Cờ này vô hiệu hóa hình ảnh động trong quá trình chuyển tiếp giữa hai activity. Có từ Android 2+.
FLAG_ACTIVITY_NO_HISTORY	Cờ này chỉ rằng activity được gọi không lưu lại trong stack, nghĩa là sau khi đi khỏi nó người dùng không thể quay lại nó.
FLAG_ACTIVITY_NO_USER_ACTION	Cờ này chỉ rằng việc chuyển tiếp activity là tự động, không phải do người dùng trực tiếp thực hiện. Ví dụ call-in activity khởi chạy tự động khi có cuộc gọi.
FLAG_ACTIVITY_REORDER_TO_FRONT	Đẩy một thực thể của activity được gọi (nếu có) trong stack lên bề mặt, trong khi phần còn lại được giữ nguyên. Kết quả là stack được sắp xếp lại.
FLAG_ACTIVITY_RESET_TASK_IF_NEEDED	Cờ này không hiệu lực trừ khi intent tạo ra một tác vụ mới hoặc đẩy một tác vụ mới lên bề mặt. Khi đó, tác vụ được thiết lập lại.
FLAG_ACTIVITY_SINGLE_TOP	Nếu thực thể của activity mục tiêu đã có sẵn trên đỉnh stack, không tạo thực thể mới nữa.

f) Component

Trường tùy chọn này là tên tường minh của một đơn thể sẽ xử lý intent. Ví dụ: `com.twe.examples.app.MyActivity`.

3. Filter của intent

Intent filter (bộ lọc intent) định nghĩa mối quan hệ giữa intent và ứng dụng. Android OS dùng filter để xác định tập các Activity, Service, và BroadcastReceiver có thể nhận intent với các action, category, extras liên kết với intent này, từ đó hướng intent đến đơn thể nhận thích hợp nhất.

Element `<intent-filter>` trong tập tin manifest liệt kê các action, category và data liên kết với intent.

Ví dụ sau chỉ định CustomActivity có thể được triệu gọi với một trong hai action, một category hoặc một dữ liệu.

```
<activity android:name=".CustomActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="com.twe.examples.intentdemo.LAUNCH" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

Intent filter cũng có thể tạo thông qua code bằng cách dùng lớp `android.content.IntentFilter`.

```
BookBroadcastReceiver receiver = new BookBroadcastReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction("com.twe.bookreceiver.action.BOOK_NEW");
filter.addCategory(Intent.CATEGORY_DEFAULT);
registerReceiver(receiver, filter);
```

Nếu một activity không có sẵn cho một intent cụ thể, ứng dụng thường bị dừng. Bạn có thể xác định một activity có sẵn hay không cho một intent cụ thể, trước khi gửi intent đi:

```
public static boolean isIntentAvailable(Context context, String action) {
    final PackageManager manager = context.getPackageManager();
    final Intent intent = new Intent(action);
    List<ResolveInfo> list = manager.queryIntentActivities(intent, PackageManager.MATCH_DEFAULT_ONLY);
    return list.size() > 0;
}
```

4. PendingIntent

Mặc dù intent được sử dụng chủ yếu để khởi chạy một tác vụ, chúng cũng được dùng để đăng ký các callback, bằng cách tạo `android.app.PendingIntent`.

Pending intent là một loại *intent chờ được xử lý*. Một ứng dụng có thể cung cấp một pending intent đến một ứng dụng khác, pending intent này chỉ được xử lý khi một điều kiện nhất định nào đó được đáp ứng hoặc một tác vụ nào đó được hoàn thành. Ví dụ, một ứng dụng đặt notification lên thanh notification, nó cung cấp cho thanh notification một pending intent. Khi người dùng tap lên notification đó, pending intent chờ sẵn sẽ báo ngược trở lại ứng dụng.

Lớp PendingIntent cung cấp bốn phương thức static để tạo các pending intent:

Phương thức	Mô tả
<code>getActivity()</code>	Được dùng khi tạo một pending intent để khởi chạy một activity.
<code>getBroadcast()</code>	Được dùng khi tạo một pending intent để gửi một intent quảng bá.
<code>getService()</code>	Được dùng khi tạo một pending intent để khởi chạy một service.
<code>getActivities()</code>	Được dùng khi tạo một pending intent để khởi chạy nhiều activity.

Ví dụ, tạo một intent dùng khởi chạy Contacts List, gói trong pending intent:

```
Intent intent = new Intent();
intent.setAction(Intent.ACTION_VIEW);
intent.setData(Uri.parse("content://contacts/people"));
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

Khi pending intent trên được gửi đến ứng dụng khác, nó có thể được xử lý bất cứ lúc nào bằng cách gọi phương thức `send()` của nó:

```
try {
    pendingIntent.send();
} catch (PendingIntent.CanceledException e) {
    e.printStackTrace();
}
```

Service

Service là đơn thể được thiết kế để chạy các tác vụ mất nhiều thời gian dưới nền. Ví dụ, một service có thể chơi nhạc dưới nền trong khi người dùng đang sử dụng một ứng dụng khác, hoặc một service tải dữ liệu qua mạng mà vẫn không ngăn người dùng cùng lúc tương tác với một ứng dụng khác.

Service không có bất kỳ một UI nào, từ activity hoàn toàn có thể kết nối với một service và trao đổi dữ liệu với nó.

Một service cơ bản có hai trạng thái:

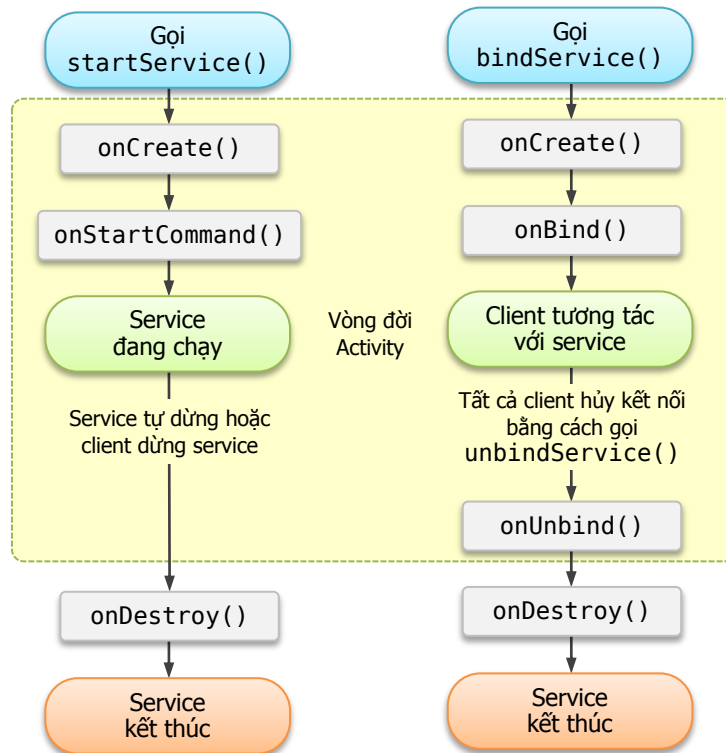
Trạng thái	Mô tả
Started	Service có trạng thái Started khi một đơn thể ứng dụng khác, chẳng hạn activity, khởi động nó bằng lời gọi <code>startService()</code> kèm theo đối tượng Intent. Nói cách khác, activity gửi intent để khởi động service. Khi đã khởi động, service có thể chạy dưới nền vô thời hạn, ngay cả khi activity khởi động nó đã bị hủy. Nó được dùng bằng phương thức <code>stopService()</code> hoặc tự dừng bằng cách gọi phương thức <code>stopSelf()</code> , hoặc bị hủy bởi Android OS để lấy tài nguyên (service có ưu tiên cao và thường bị hủy cuối cùng). Service có trạng thái Started thường không trả về kết quả hoặc cho phép tương tác với đơn thể khởi chạy nó.
Bound	Service có trạng thái Bound khi một đơn thể ứng dụng khác liên kết với nó bằng lời gọi <code>bindService()</code> . Service ở trạng thái Bound cung cấp giao diện IBinder cho phép đơn thể liên kết tương tác với service, gửi yêu cầu, nhận kết quả. Đơn thể liên kết đến nó hủy liên kết bằng cách gọi phương thức <code>unbindService()</code> . Service không thể dùng cho đến khi các đơn thể khác hủy kết nối hoàn toàn với nó. Dùng service có trạng thái Bound khi muốn nhận kết quả và tương tác với service.

1. Vòng đời của service

Để tạo một service, bạn thừa kế lớp cơ sở `android.app.Service` hoặc một trong những lớp con của nó. Lớp Service định nghĩa một số phương thức callback quan trọng, bạn không cần cài đặt tất cả các phương thức callback, tuy nhiên bạn cần hiểu rõ từng phương thức và cài đặt chúng sao cho ứng dụng của bạn có hành vi theo cách bạn mong đợi.

Bạn có thể lựa chọn cài đặt cho các phương thức callback trong vòng đời của một service để theo dõi các thay đổi trạng thái của service và thực hiện các công việc ở giai đoạn thích hợp:

Phương thức callback	Mô tả
<code>onStartCommand()</code>	Hệ thống gọi phương thức này khi một đơn thể, chẳng hạn activity, khởi động service tường minh bằng cách gửi intent. Trong khi service chạy, phương thức này có thể được gọi nhiều lần bởi hệ thống. Bạn có trách nhiệm dừng service khi service thực hiện xong, bằng cách gọi phương thức <code>stopService()</code> hoặc tự dừng service bằng <code>stopSelf()</code> . Không cần cài đặt cho service kiểu Bound.
<code>onBind()</code>	Hệ thống gọi phương thức này khi một đơn thể khác muốn kết nối với service bằng cách gọi <code>bindService()</code> . Phương thức này được khai báo là abstract, nên bạn phải cung cấp một cài đặt cho nó. Với service kiểu Started, trả về null. Với service kiểu Bound, phải cung cấp một giao diện để client liên lạc với service bằng cách trả về một đối tượng IBinder.
<code>onUnbind()</code>	Hệ thống gọi phương thức này khi tất cả các client ngắt kết nối khỏi giao diện do service đưa ra.
<code>onRebind()</code>	Hệ thống gọi phương thức này khi các client mới kết nối đến service, trước đó các client này báo ngắt kết nối với <code>onUnbind()</code> của service.
<code>onCreate()</code>	Hệ thống gọi phương thức này khi service được tạo lần đầu, bằng cách dùng <code>onStartCommand()</code> hoặc <code>onBind()</code> . Lời gọi này thực hiện chỉ một lần.
<code>onDestroy()</code>	Hệ thống gọi phương thức này khi service bị dừng bằng cách gọi phương thức <code>stopService()</code> . Trong <code>onDestroy()</code> thường cài đặt các tác vụ dọn dẹp tài nguyên như thread, listener, receiver.



Vòng đời của hai loại service

Android làm bất kỳ điều gì cần thiết để sử dụng hiệu quả tài nguyên hệ thống, nhằm cung cấp trải nghiệm mượt mà cho người dùng. Vì vậy, bất kỳ ứng dụng nào mà người dùng không nhìn thấy hoặc không tương tác được xem là ít quan trọng hơn ứng dụng chạy bề mặt. Khi cần thêm tài nguyên, Android sẽ quyết định chấm dứt ứng dụng chạy nền ít quan trọng nhất. Service đang chạy ở chế độ nền cũng không được xem là quan trọng và có thể bị Android chấm dứt (kill) tại bất kỳ thời điểm nào. Chiến lược khởi động lại service có thể được điều chỉnh thông qua *trị trả về* của phương thức `onStartCommand()`:

- `START_NOT_STICKY` chỉ định rằng service sẽ không khởi động lại sau khi bị chấm dứt bởi Android, trừ phi một lệnh khởi động service tường minh được gửi.
- `START_STICKY` chỉ định service cần khởi động lại sau khi bị chấm dứt bởi Android, intent ban đầu bị loại bỏ.
- `START_REDELIVERY_INTENT` chỉ định service cần khởi động lại và cũng cần intent ban đầu.

Một số service khởi chạy khi Android OS khởi động. Điều này được thực hiện bằng cách dùng broadcast receiver có intent filter chặn bắt intent hệ thống `android.intent.action.BOOT_COMPLETED`, khi broadcast receiver bắt được intent này, nó sẽ triệu gọi các service cần thiết. Để hoạt động, Broadcast receiver như vậy phải được cấp quyền `RECEIVE_BOOT_COMPLETED`.

2. Liên lạc với service

Trong service, lớp `MyService`, cài đặt các phương thức callback:

- `onBind()`, cho phép kết nối một activity đến service. Nó trả về một đối tượng `IBinder`, cho phép service trực tiếp truy cập các phương thức và các thành viên. Trong ví dụ dưới, ta dùng service loại `Started` nên phương thức này trả về `null`.
- `onStartCommand()`, chạy tác vụ của service, thường đặt trong thread do chiếm nhiều thời gian. Trong ví dụ, ta gửi các intent quảng bá về cho activity chặn bắt bằng broadcast receiver mà nó đăng ký.
- `onDestroy()`, sẽ chạy khi service bị activity dừng tường minh bằng `stopService()`. Nếu không dừng tường minh, bạn có thể gọi `stopSelf()` sau khi thực hiện xong tác vụ của service để service tự dừng.

```

public class MyService extends Service {
    boolean isRunning = true;

    @Override public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override public void onCreate() {
        super.onCreate();
    }

    @Override public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d("MyService", "I am alive!");
        new Thread(new Runnable() {
            long startingTime = System.currentTimeMillis();
            public void run() {
                for (int i = 0; (i < 120) && isRunning; ++i) {
                    try {
                        Intent intent = new Intent("com.twe.actions.GOSERVICE");
                        String message = i + " value: " + System.currentTimeMillis() - startingTime;
                        intent.putExtra("serviceData", message);
                    }
                }
            }
        }).start();
    }
}

```

```

        sendBroadcast(intent);
        Thread.sleep(1000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
} // run
}).start();
return super.onStartCommand(intent, flags, startId);
}

@Override public void onDestroy() {
    super.onDestroy();
    Log.d("MyService", "I'm dead!");
    isRunning = false;
}
}

```

Chú ý là với service chạy cùng tiến trình với activity, nếu tác vụ của service chiếm nhiều thời gian, nên đặt nó vào một thread riêng. Tóm lại, thực hiện tác vụ trong Thread hoặc AsyncTask, rồi đưa nó vào onStartCommand() của service.

Service liên lạc với activity bằng cách quảng bá các intent chứa dữ liệu. Activity sẽ định nghĩa một intent filter, tạo một đối tượng broadcast receiver và đăng ký đối tượng receiver với filter. Activity dùng intent filter này để lọc các intent do service quảng bá và nhận dữ liệu kèm theo intent.

Trong activity:

- Để khởi chạy một service ta gọi phương thức startService() với tham số là đối tượng Intent. Bạn có thể gửi dữ liệu khởi tạo cho service bằng cách kèm thêm extras vào đối tượng intent này.
- Tạo đối tượng receiver thuộc lớp BroadcastReceiver, định nghĩa intent filter và đăng ký receiver với filter này. Activity dùng đối tượng receiver chặn bắt các intent quảng bá từ service.

```

public class MainActivity extends Activity {
    private TextView mTextView;
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override public void onReceive(Context context, Intent intent) {
            String serviceData = intent.getStringExtra("serviceData");
            String now = "\n" + serviceData + " --- " + new Date().toString();
            mTextView.append(now);
        }
    };

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.text_id);
    }

    @Override public void onResume() {
        super.onResume();
        // định nghĩa filter và đăng ký receiver
        registerReceiver(receiver, new IntentFilter("com.twe.actions.GOSERVICE"));
    }

    @Override protected void onPause() {
        super.onPause();
        stopService(new Intent(getBaseContext(), MyService.class));
        unregisterReceiver(receiver);
    }

    // xử lý của nút Start Service, gọi từ layout XML
    public startService(View view) {
        startService(new Intent(getBaseContext(), MyService.class));
        mTextView.setText("MyService started - (see ADM Log)");
    }

    // xử lý của nút Stop Service, gọi từ layout XML
    public stopService(View view) {
        stopService(new Intent(getBaseContext(), MyService.class));
        mTextView.setText("MyService Destroyed");
    }
}

```

Thông tin do service chuyển đến được cập nhật vào EditText. Xem tập tin layout XML:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"

```

```

android:layout_height="match_parent"
android:orientation="vertical" >
<EditText android:id="@+id/text_id"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:inputType="textMultiLine"
    android:textSize="12sp" />
<Button android:id="@+id/button_start_id"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:text="Start Service"
    android:onClick="startService" />
<Button android:id="@+id/button_top_id"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:text="Stop Service"
    android:onClick="stopService" />
</LinearLayout>

```

Tạo lớp MyService vẫn chưa đủ, Android yêu cầu phải khai báo service MyService trong tập tin manifest, dùng element <service>:

```

<service android:name=".MyService"
    android:enabled="true"
    android:exported="false" />

```

- Nếu khai báo không dùng intent filter như trên, service chỉ khởi động với intent tường minh từ đơn thể khác trong cùng ứng dụng, gọi là *local service*. Nếu bạn muốn service được truy cập bởi các ứng dụng khác trên cùng thiết bị, thêm một intent filter vào service, đồng thời đặt thuộc tính android:exported là true (mặc định). Khi đó, với ứng dụng khác, service gọi là *remote service*.

```

<service android:name=".MyService">
    <intent-filter>
        <action android:name="com.twe.servicedemo.MyService" />
    </intent-filter>
</service>

```

- Nếu bạn quyết định "export" service của bạn cho các ứng dụng khác sử dụng, Android hỗ trợ thiết lập quyền truy cập cho service đó bằng cách dùng thuộc tính android:permission của element <service>:

```

<service android:name=".MyService"
    android:enabled="true"
    android:exported="true"
    android:permission="com.twe.permission.MY_SERVICE">
    <intent-filter>
        <action android:name="com.twe.servicedemo.MyService" />
    </intent-filter>
</service>

```

Như vậy, các ứng dụng khác sẽ khai báo quyền truy cập service trong tập tin manifest của ứng dụng đó; người sử dụng sẽ xem và phê duyệt các quyền này khi cài đặt ứng dụng.

```

<uses-permission android:name="com.twe.permission.MY_SERVICE" />

```

- Mặc định, service chạy trong cùng tiến trình với đơn thể gọi nó. Để ép service chạy trong tiến trình của riêng nó, thêm thuộc tính android:process, khai báo tên của tiến trình phía trước có (:) với local service và ký tự thường với remote service.

3. Kết nối activity đến service

Khi code gọi service (thường từ activity) và service cần kết nối, ta dùng service kiểu Bound. Kênh liên lạc giữa code gọi service và service phụ thuộc rất nhiều vào loại service: local service hoặc remote service. Tùy mỗi loại, phương thức onBind() trả về đối tượng khác nhau.

a) Local Service

Kênh liên lạc dễ dàng được cung cấp bằng cách truyền thực thể android.os.Binder, trả về trực tiếp thực thể service. Trong lớp service:

- Chuẩn bị dữ liệu thành viên nhận dữ liệu do activity truyền đến, trong ví dụ là mảng movies.
- Tạo thực thể LocalBinder thừa kế Binder. Phương thức getService() của nó trả về thực thể của chính service.
- Phương thức onBind() bây giờ trả về thực thể LocalBinder tạo ở trên.
- Khi gọi tác vụ chạy nền trong onStartCommand(), truyền mảng dữ liệu movies để tác vụ chạy nền xử lý.

```

public class MyService extends Service {
    String[] movies; // dữ liệu cần nhận
    private final IBinder mLocalBinder = new LocalBinder();

    public class LocalBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
}

```

```

}

@Override public IBinder onBind(Intent arg0) {
    return mLocalBinder;
}

@Override public int onStartCommand(Intent intent, int flags, int startId) {
    // tác vụ chạy nền
    new DoBackgroundTask().execute(movies);
    return START_STICKY;
}

private class DoBackgroundTask extends AsyncTask<String, String, String> { ... }
// ...
}

```

Trong activity:

- Tạo đối tượng thuộc lớp ServiceConnection để theo dõi trạng thái của service. Cài đặt:
 - + `onServiceConnected()`, được gọi khi kết nối được tạo, nghĩa là sau khi gọi `bindService()`.
 - + `onServiceDisconnected()`, được gọi khi service ngắt kết nối với activity.

Từ tham số kiểu IBinder của phương thức `onServiceConnected()` ta lấy được đối tượng Binder, rồi gọi `getService()` để lấy tham chiếu đến service, từ đó bạn có thể truy cập thành viên của service một cách trực tiếp.

Trong phương thức `onServiceConnected()`, bạn cũng khởi động service bằng `startService()`.

- Trước khi bạn khởi động service bằng ServiceConnection, phải kết nối activity với service bằng `bindService()`. Phương thức này nhận tham số với đối tượng ServiceConnection để cập ở trên.

```

public class MainActivity extends Activity {
    MyService mServiceBinder;
    Intent intent;

    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            // được gọi khi connection được tạo
            mServiceBinder = ((MyService.LocalBinder)service).getService();
            String[] movies = { "Howl's Moving Castle", "Spirited Away" };
            // chuyển dữ liệu đến service thông qua đối tượng serviceBinder
            mServiceBinder.movies = movies;
            startService(intent);
        }

        public void onServiceDisconnected(ComponentName className) {
            // được gọi khi service ngắt kết nối
            mServiceBinder = null;
        }
    };

    // xử lý sự kiện nhấn nút Start Service
    public void startService(View view) {
        intent = new Intent(this, MyService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override protected void onCreate(Bundle savedInstanceState) {
        // ...
    }
    // ...
}

```

b) Remote Service

Ứng dụng gọi và service thuộc về hai tiến trình khác nhau, liên lạc thông qua IBinder được thực hiện bằng cơ chế IPC (Inter-Process Communication, giao tiếp liên tiến trình). Tùy theo mục tiêu của service, hoặc định nghĩa một giao diện service bằng cách dùng AIDL (Android Interface Definition Language), hoặc dùng một hàng đợi thông điệp đơn giản để chứa dữ liệu trao đổi giữa hai tiến trình.

- Dùng AIDL

Giống triệu gọi phương thức từ xa, trước hết cần định nghĩa một giao diện cho service. Do giao diện này dùng cho code gọi thuộc tiến trình khác, nên ta định nghĩa dưới dạng AIDL, trong tập tin `IMyServiceInterface.aidl`:

```

interface IMyServiceInterface {
    int add(in int a, in int b);
}

```

Trong giai đoạn biên dịch, các tập tin class sẽ được sinh ra từ tập tin giao diện AIDL này. Code gọi service và service cần các tập tin class này để liên lạc thông qua kênh IBinder.

Service cần cài đặt lớp Stub để cung cấp cài đặt cho các phương thức của giao diện `IMyServiceInterface`.

```
public class MyService extends Service {
    @Override public IBinder onBind(Intent intent) {
        return new IMyServiceInterface.Stub() {
            @Override public int add(int a, int b) throws RemoteException {
                return a + b;
            }
        };
    }
}
```

Code gọi service cũng dùng giao diện `IMyServiceInterface`:

```
public class MainActivity extends Activity {
    private IMyServiceInterface mServiceInterface;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent intent = new Intent();
        intent.setComponent(new ComponentName("com.twe.remote", "com.twe.remote.MyService"));
        bindService(intent, new ServiceConnection() {
            @Override public void onServiceConnected(ComponentName className, IBinder binder) {
                mServiceInterface = IMyServiceInterface.Stub.asInterface(binder);
                try {
                    int total = mServiceInterface.add(10, 20);
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }

            @Override public void onServiceDisconnected(ComponentName className) {
                mServiceInterface = null;
            }
        }, Context.BIND_AUTO_CREATE);
    }
}
```

- Dùng hàng đợi thông điệp (message queue)

Nếu dữ liệu trao đổi đơn giản, chỉ có kiểu nguyên thủy, Android cung cấp lớp `android.os.Message` để tạo một hàng đợi thông điệp đơn giản giữa code gọi và service.

Service cần cài đặt hàng đợi thông điệp:

```
public class MyService extends Service {
    public static final int MESSAGE_DOWNLOAD = 1;
    private final Messenger mMessenger = new Messenger(new RequestHandler());

    private class RequestHandler extends Handler {
        @Override public void handleMessage(Message message) {
            switch (message.what) {
                case MESSAGE_DOWNLOAD:
                    // bắt đầu download
                    break;
                default: super.handleMessage(message);
            }
        }
    }

    @Override public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
}
```

Hàng đợi thông điệp nhận và xử lý các thực thể `android.os.Message`. Để gửi các thông điệp này, code gọi service cũng dựa trên thực thể `Messenger`:

```
public class MainActivity extends Activity {
    public static final int MESSAGE_DOWNLOAD = 1;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
        Intent intent = new Intent();
        intent.setComponent(new ComponentName("com.twe.remote", "com.twe.remote.MyService"));
        bindService(intent, new ServiceConnection() {
            @Override public void onServiceConnected(ComponentName className, IBinder binder) {

```



```

        Messenger service = new Messenger(binder);
        Message downloadMessage = Message.obtain(null, MESSAGE_DOWNLOAD);
        try {
            service.send(downloadMessage);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @Override public void onServiceDisconnected(ComponentName className) { }
}, Context.BIND_AUTO_CREATE);
}
}

```

4. IntentService

Khi service thực hiện xong tác vụ chạy nền, service phải tự dừng càng sớm càng tốt để giải phóng tài nguyên. Bạn thường dùng phương thức tự dừng `stopSelf()` để thực hiện việc này.

Để dễ dàng tạo service chạy một tác vụ nền trong một thread riêng và tự động kết thúc (auto-stop) khi hoàn thành xong tác vụ nền, Android cung cấp lớp `android.app.IntentService`.

Phương thức `onHandleIntent()` là nơi bạn đặt code của tác vụ cần thực hiện trong service, tác vụ này sẽ chạy trong một thread tách biệt. Khi tác vụ kết thúc, thread kết thúc và service sẽ tự động dừng.

```

public class MyIntentService extends IntentService {
    public MyIntentService() {
        super(MyService.class.getName());
    }

    protected void onHandleIntent(Intent intent) {
        // code thực hiện tác vụ chạy nền với service tự động dừng
        // trong đó xử lý intent gửi đến để khởi động service
    }
}

```

Chú ý, các phương thức callback `onStartCommand()` và `onBind()` không cần cài đặt trong lớp con của `IntentService`.

Tuy nhiên, khi dùng `IntentService`, service và code gọi không có bất kỳ kết nối nào trong vòng đời của service. Bạn chỉ nên dùng với các service không yêu cầu xử lý đồng bộ các yêu cầu.

5. Service hệ thống

Khác với các service do ứng dụng cung cấp, giao diện service hệ thống có thể lấy đơn giản bằng cách truyền tên service (hằng của lớp `Context`) cho phương thức `Context.getSystemService()`:

```

PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);

```

Danh sách sau là các service hệ thống đáng chú ý của Android:

`android.app.ActivityManager`

Cho phép tương tác với trạng thái activity toàn cục của hệ thống (ví dụ lấy activity người dùng tương tác gần nhất).

`android.os.PowerManager`

Cho phép điều khiển cấu hình nguồn điện của thiết bị (ví dụ, ngăn chặn thiết bị chuyển sang chế độ sleep).

`android.app.AlarmManager`

Cho phép ứng dụng sắp xếp một pending intent sẽ thực hiện tại một thời điểm tương lai.

`android.app.NotificationManager`

Cho phép ứng dụng thông báo người dùng sự kiện trong nền, bằng cách hiển thị notification trên thanh notification.

`android.location.LocationManager`

Cho phép ứng dụng nhận được thông tin vị trí từ các nguồn cung cấp vị trí cho thiết bị (ví dụ, GPS).

`android.app.DownloadManager`

Cho phép ứng dụng yêu cầu tác vụ download HTTP chiếm nhiều thời gian. Service xử lý download trong nền và thông báo cho hệ thống khi download chấm dứt.

`android.net.ConnectivityManager`

Cho phép ứng dụng truy vấn trạng thái kết nối của thiết bị (ví dụ, mạng di động hoặc kết nối WiFi).

`android.net.wifi.WifiManager`

Cho phép ứng dụng tương tác với mạng WiFi (ví dụ, tìm kiếm mạng WiFi và thêm cấu hình mạng WiFi mới).

`android.app.UiModeManager`

Cho phép ứng dụng truy vấn và thao tác chế độ UI của thiết bị.

`android.view.inputmethod.InputMethodManager`

Cho phép ứng dụng điều khiển các phương thức nhập (ví dụ, hiển thị và ẩn bàn phím ảo).

`android.app.KeyguardManager`

Cho phép ứng dụng khóa và mở khóa màn hình khóa phím.

`android.app.SearchManager`

Cung cấp quyền truy cập đến chức năng tìm kiếm của Android.

`android.view.WindowManager`

Cung cấp truy cập window manager cấp cao nhất, cho phép đặt cửa sổ tùy chỉnh.

`android.view.LayoutInflater`

Cho phép "inflate" tài nguyên layout từ nguồn tài nguyên chỉ định trong context.

Cho phép ứng dụng điều khiển độ rung trên thiết bị để thông báo cho người dùng trong chế độ im lặng.

a) Timer

```
static final int UPDATE_INTERVAL= 1000;
private Timer timer = new Timer();
```

```
private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate(new TimerTask() {
        @Override public void run() {
            // code thực hiện tác vụ lặp định kỳ
        }
    }, 0, UPDATE_INTERVAL);
}
```

```
@Override public void onDestroy() {
    super.onDestroy();
    if (timer != null) timer.cancel();
}
```

b) Thread và các đơn thể

Thread trong Android nói chung giống thread của Java. Từ quan điểm ứng dụng, Android có ba loại thread:

The diagram illustrates the internal structure of an Android Process. It is divided into two main sections: 'Đơn thể bên trong' (Internal Components) and 'Đơn thể bên ngoài' (External Components).

Internal Components (Đơn thể bên trong):

- Main Thread:** A vertical blue bar representing the primary thread of execution. It receives calls from both internal and external components.
- Thread Pool:** A vertical blue bar representing a pool of threads used for background tasks.
- Activity:** A green rounded rectangle that starts the process and calls the Main Thread.
- Service:** A light blue rounded rectangle that can be called by the Main Thread or the Thread Pool.
- Content Provider:** An orange rounded rectangle that can be called by the Main Thread or the Thread Pool.
- Broadcast Receiver:** A purple rounded rectangle that can be called by the Main Thread.

External Components (Đơn thể bên ngoài):

- Đơn thể bên ngoài:** An orange rounded rectangle representing external components that can call the Thread Pool via 'Gọi Service/Content Provider'.

Call Flow (Gọi):

- 'Gọi tất cả các đơn thể' (Call all components) from the internal components to the Main Thread.
- 'Gọi Activity/Broadcast Receiver' from the internal components to the Main Thread.
- 'Gọi Service/Content Provider' from the external components to the Thread Pool.

+ Activity: ví dụ bạn nhấn một nút của activity, thiết bị sẽ chuyển hành động này thành thông điệp và đặt vào hàng đợi chính của tiến trình đang quan tâm. Thread chính xử lý từng thông điệp trong vòng lặp. Nếu thông điệp mất hơn 5 giây để xử lý, Android ném ra thông điệp ANR (Application Not Responsding).

+ Service: khi bạn khởi động một service bằng phương thức `startService()`, một thông điệp được đặt vào hàng đợi chính và thread chính sẽ xử lý chúng bằng code của service.

Nói cách khác, Android áp đặt một môi trường single-thread cho các thành phần UI.

Binder thread được dùng để liên lạc giữa các thread trong các tiến trình khác nhau. Mỗi tiến trình duy trì một tập hợp các thread, gọi là thread pool. Các thread này xử lý các thông điệp đến từ các tiến trình khác, ví dụ service hệ thống, content provider, service khác. Khi cần, một binder thread được tạo ra để xử lý yêu cầu gửi đến. Đa số trường hợp, ứng dụng không

phải quan tâm binder thread, ngoại trừ ứng dụng cung cấp service kiểu Bounded, có thể ràng buộc với tiến trình khác thông qua giao diện AIDL.

- Thread chạy nền

Nếu phải thực hiện tác vụ nào có thể chiếm hơn 5 giây, như tác vụ mạng, truy cập cơ sở dữ liệu, tính toán phức tạp, ... bạn phải đặt nó vào một thread chạy nền để tránh ANR. Một tiến trình mới tạo không chứa thread chạy nền nào, tiến trình sẽ tạo nó khi cần. Tất cả các thread mà một ứng dụng tạo tưởng mình là thread chạy nền (background thread).

Cơ bản, có hai cách tạo một thread chạy nền: tạo một lớp thừa kế lớp Thread và viết lại phương thức run(); hoặc tạo một thực thể lớp Thread và truyền cho nó đối tượng cài đặt giao diện Runnable, bạn phải cài đặt phương thức run() cho đối tượng này.

Sau đó, với cả hai cách, gọi phương thức start() của thread để thực sự chạy thread mới (không gọi phương thức run()).

Như vậy, mỗi thực thể máy ảo có ít nhất một UI thread chạy khi nó khởi động, hiển thị giao diện đồ họa tương tác với người dùng. Ứng dụng có thể quyết định sinh các thread chạy nền bổ sung cho những mục đích cụ thể.

c) Vấn đề cập nhật UI thread

Tác vụ chạy nền (worker thread) vẫn có nhu cầu cập nhật UI thread, ví dụ hiển thị tiến độ chạy. Tuy nhiên, Android không cho phép các thread chạy nền tương tác với UI thread vì nó không an toàn thread (not thread-safe). Nói cách khác, Android chỉ cho phép cập nhật view từ UI thread.

Thread chính của ứng dụng sẽ chạy một hàng đợi thông điệp (message queue) dùng quản lý các đơn thể quan trọng của ứng dụng: activity, intent, ... Bạn có thể tạo thread chạy nền và giao tiếp với thread chính bằng cách dùng đối tượng Handler của thread chính.

Khi một Handler được tạo ra, nó liên kết với hàng đợi thông điệp của thread chính tạo ra nó. Từ thời điểm đó, Handler có thể chuyển các đối tượng Message và Runnable đến hàng đợi thông điệp và lần lượt xử lý chúng.

Dùng Handler, ta cập nhật UI thread với các giải pháp:

- Cài đặt code cập nhật UI thread trong phương thức handleMessage() của đối tượng Handler. Trong thread chạy nền, dùng phương thức sendMessage() của đối tượng Handler gửi đối tượng Message lên hàng đợi thông điệp. Đối tượng Message này chứa các thông tin sẽ được trích xuất để cập nhật UI.

Thread chạy nền sẽ yêu cầu một message token bằng phương thức obtainMessage(). Sau khi có message token (đối tượng Message), thread chạy nền nạp dữ liệu vào message token và dùng phương thức sendMessage() của đối tượng Handler gửi nó lên hàng đợi thông điệp.

```
Handler mHandler = new Handler() {
    @Override public void handleMessage(Message message) {
        String value = (String) message.obj;
        // code cập nhật UI với thông tin từ msg nhận được
    }
};

// thread chạy nền
Thread mBackgroundJob = new Thread (new Runnable () {
    @Override public void run() {
        // thực thi tác vụ yêu cầu thời gian nhiều
        // lấy message token
        Message message = mHandler.obtainMessage();
        // chuyển message token đến hàng đợi thông điệp
        mHandler.sendMessage(message);
    }
});

@Override public void onStart() {
    super.onStart();
    mBackgroundJob.start();
}
```

- Cài đặt code cập nhật UI thread trong phương thức run() của một đối tượng Runnable riêng, gọi là tác vụ chạy bề mặt. Trong thread chạy nền, dùng các phương thức post() của đối tượng Handler gửi đối tượng Runnable này lên hàng đợi thông điệp. Ngoài post(), còn có các phương thức postAtFrontOfQueue(), postAtTime(), postDelayed().

```
Handler mHandler = new Handler();
// task nền sẽ dùng post() gửi runnable chạy bề mặt
private Runnable mBackgroundTask = new Runnable() {
    @Override public void run() {
        // thực thi tác vụ yêu cầu thời gian nhiều
        mHandler.post(mForegroundTask);
    }
};

// runnable chạy bề mặt, cập nhật UI
private Runnable mForegroundTask = new Runnable() {
    @Override public void run() {
        // code cập nhật UI
    }
}
```

```
@Override public void onStart() {
    super.onStart();
    new Thread(mBackgroundTask).start();
}
```

7. AsyncTask

Nếu bạn thấy cập nhật UI thread bằng cách dùng Handler phức tạp, bạn có thể dùng AsyncTask. AsyncTask là một cơ chế Android tạo ra để giúp xử lý tác vụ chiếm nhiều thời gian mà vẫn liên lạc với UI thread một cách an toàn. Để sử dụng lớp này, bạn cần thừa kế lớp AsyncTask<Params, Progress, Result> và cài đặt các phương thức:

onPreExecute() công việc thực hiện trước khi chạy tác vụ nền.
doInBackground() công việc phải làm trong nền, thực hiện tác vụ chiếm nhiều thời gian. Kích hoạt khi gọi phương thức execute() của AsyncTask.
onProgressUpdate() công việc phải làm khi đang chạy nền, định kỳ cập nhật UI thread. Nếu cài đặt phương thức này phải cài đặt thêm phương thức publishProcess().
onPostExecute() công việc phải làm khi tác vụ nền hoàn thành, thường là cập nhật UI thread.

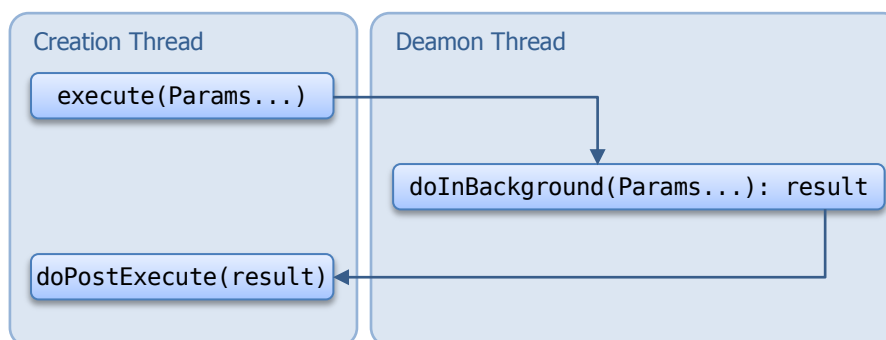
Các kiểu generic dùng trong AsyncTask:

Kiểu tham số	Mô tả
Params	Kiểu của mảng tham số được truyền đến tác vụ nền để xử lý.
Progress	Kiểu của mảng tham số mà tác vụ nền dùng trong báo cáo định kỳ cho UI thread.
Result	Kiểu của kết quả trả về sau khi tác vụ nền kết thúc.

Không bắt buộc phải dùng các kiểu generic trên, nếu không quan tâm đến kiểu, dùng kiểu Void.

Chi tiết về các phương thức của AsyncTask:

Phương thức	Mục đích
onPreExecute()	Triệu gọi trên UI thread ngay khi AsyncTask chạy. Thường dùng để khởi gán cho tác vụ nền, khởi tạo cho cập nhật trên UI thread. Ví dụ: hiển thị progress bar trên UI để theo dõi tác vụ nền.
doInBackground(Params...)	Triệu gọi trên thread chạy nền ngay sau khi onPreExecute() kết thúc thực hiện. Đây là nơi thực hiện tác vụ yêu cầu nhiều thời gian. Các tham số Params... được truyền từ phương thức execute() đến phương thức này. Trong phương thức, có thể dùng publishProgress(Progress...) để định kỳ trả về các kết quả tính toán. Các kết quả này sẽ được chuyển đến UI thread bằng phương thức onProgressUpdate().
onProgressUpdate(Progress...)	Triệu gọi trên UI thread mỗi khi gọi publishProgress(Progress...). Thường dùng để cập nhật hoặc báo cho cho UI thread kết quả tác vụ nền đang chạy. khởi gán cho tác vụ nền, khởi tạo cho cập nhật trên UI thread. Ví dụ: cập nhật progress bar cho thấy tiến độ chạy.
onPostExecute(Result)	Triệu gọi trên UI thread sau khi tác vụ nền kết thúc. Kết quả chạy được truyền đến tham số Result của phương thức này.
onCancelled(Result)	Nếu AsyncTask bị hủy, phương thức này lấy kết quả thay cho onPostExecute().



Luồng dữ liệu trong AsyncTask

Cơ chế cập nhật khi đang thực hiện tác vụ chạy nền được dùng để báo cho người dùng tiến độ thực hiện và trả về kết quả từng phần, ví dụ tải nhiều ảnh từ mạng, publishProgress() sẽ cập nhật UI thread với từng ảnh tải được.

Lớp con của AsyncTask thường được viết như lớp nội của activity để dễ cập nhật UI thread. Nếu viết tách biệt, đối tượng Context của activity sẽ được truyền như tham số đến constructor của lớp thừa kế AsyncTask.

Activity gọi AsyncTask, chú ý chuỗi thực thi:

```
public class MainActivity extends ActionBarActivity {
    private static final String[] DOWNLOAD_URLS = {
        "http://developer.android.com/design/media/devices_displays_density@2x.png",
        "http://developer.android.com/design/media/iconography_launcher_example2.png",
        "http://developer.android.com/design/media/iconography_actionbar_focal.png",
        "http://developer.android.com/design/media/iconography_actionbar_colors.png"
    };
    ProgressBar mProgressBar;
    LinearLayout mLayoutImages;
    DownloadTask task;
```

```

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mProgressBar = (ProgressBar) findViewById(R.id.progress_bar_id);
    mProgressBar.setMax(DOWNLOAD_URLS.length);
    mLayoutImages = (LinearLayout) findViewById(R.id.layout_image_id);
    task = new DownloadTask();
    task.execute(DOWNLOAD_URLS); // 1. tạo một AsyncTask và gọi execute() để thực thi
}

@Override protected void onDestroy() {
    super.onDestroy();
    task.cancel(true);
}

private class DownloadTask extends AsyncTask<String, Bitmap, Void> {
    private int count = 0;
    // 2. trước khi bắt đầu AsyncTask, chuẩn bị UI, ví dụ hiển thị ProgressBar
    @Override protected void onPreExecute() {
        super.onPreExecute();
        mProgressBar.setVisibility(View.VISIBLE);
        mProgressBar.setProgress(0);
    }

    // 3. thực hiện công việc yêu cầu nhiều thời gian. Không cập nhật trực tiếp UI thread ở đây
    @Override protected Void doInBackground(String... urls) {
        for (String url : urls) {
            if (!isCancelled()) {
                Bitmap bitmap = downloadFile(url);
                // 4. phương thức hỗ trợ, liên tục lấy kết quả trả về
                publishProgress(bitmap);
            }
        }
        return null;
    }

    // 5. kích hoạt bởi publishProgress(), định kỳ cập nhật UI thread
    @Override protected void onProgressUpdate(Bitmap... bitmaps) {
        super.onProgressUpdate(bitmaps);
        mProgressBar.setProgress(++count);
        ImageView mImageView = new ImageView(MainActivity.this);
        mImageView.setImageBitmap(bitmaps[0]);
        mLayoutImages.addView(mImageView);
    }

    // 6. hiển thị kết quả, dọn dẹp sau khi kết thúc, có thể dùng UI thread ở đây
    @Override protected void onPostExecute(Void unused) {
        super.onPostExecute(unused);
        mProgressBar.setVisibility(View.GONE);
    }

    @Override protected void onCancelled() {
        super.onCancelled();
        mProgressBar.setVisibility(View.GONE);
    }

    private Bitmap downloadFile(String url) {
        Bitmap bitmap = null;
        try {
            bitmap = BitmapFactory.decodeStream((InputStream) new URL(url).getContent());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return bitmap;
    }
}
}

```

Kể từ với API Level 11, bạn có thể chạy nhiều AsyncTask trên nhiều bộ xử lý có nhiều lõi. Ngoài execute(Params...), bạn có thể gọi:

- execute(Runnable) dùng thực thi Runnable thay cho viết lại doInBackground. Runnable được xử lý trong môi trường thực thi AsyncTask, không gọi onPreExecute(), onPostExecute() và onCancelled().

- executeOnExecutor(Executor, Params...) cấu hình môi trường thực thi thực mà trên đó tác vụ được xử lý. Executor có thể là AsyncTask.THREAD_POOL_EXECUTOR hoặc AsyncTask.SERIAL_EXECUTOR. Nhiều AsyncTask có thể được thực hiện cùng một lúc tùy thuộc vào số lượng lõi của bộ xử lý.

Broadcast receiver

Android cung cấp khả năng phát thông điệp toàn hệ thống, gọi là intent quảng bá (broadcast). Intent quảng bá thông báo các sự kiện của hệ thống, như thay đổi kết nối đến Internet, mức độ sạc pin, SMS gửi đến, điện thoại gọi đến.

Ứng dụng trên Android phải có khả năng xử lý hướng sự kiện dựa trên intent quảng bá. Nghĩa là xử lý các sự kiện (event) phát từ hệ thống hoặc từ các ứng dụng khác. Để thực hiện điều này, ứng dụng dùng đơn thể broadcast receiver đón nhận và đáp ứng các intent quảng bá đó.

Nói rõ hơn, phương thức onReceive() của broadcast receiver sẽ được kích hoạt nếu intent quảng bá gửi đến so trùng với loại intent quảng bá mà nó đăng ký. Giống như service, broadcast receiver không có UI, nhưng onReceive() của nó có thể gọi một activity để đáp ứng thông tin nhận được.

Ví dụ, nếu một ứng dụng muốn nhận và đáp ứng một event toàn cục như chuông điện thoại hoặc một SMS gửi đến, nó phải đăng ký một broadcast receiver bằng code hoặc bằng khai báo trong tập tin manifest của ứng dụng, chặn bắt event này.

Broadcast receiver thường được dùng để cập nhật nội dung, khởi động service, cập nhật UI của activity hoặc cảnh báo cho người dùng, dựa trên sự kiện nó nhận được.

Có hai bước tạo và dùng broadcast receiver:

- Tạo broadcast receiver.
- Đăng ký broadcast receiver.

Nếu bạn cài đặt một intent tùy biến riêng, sau đó tạo và quảng bá intent này, sẽ phải thêm một số bước bổ sung.

1. Tạo broadcast receiver

Một broadcast receiver được cài đặt như lớp con của lớp BroadcastReceiver, trong đó ta cài đặt phương thức callback chủ yếu onReceive(), tham số kiểu đối tượng Intent của phương thức này chính là thông điệp quảng bá nhận được.

```
public class MyReceiver extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected", Toast.LENGTH_LONG).show();
    }
}
```

Khi một intent quảng bá đến broadcast receiver, Android gọi phương thức callback onReceive() của broadcast receiver và truyền cho phương thức này đối tượng Intent chứa trong thông điệp.

Một broadcast receiver xem như hoạt động khi phương thức onReceive() của nó hoạt động. Nếu xử lý thông điệp cần nhiều thời gian, broadcast receiver phải khởi động một service để xử lý thông điệp dưới nền.

2. Đăng ký broadcast receiver

Để lắng nghe một loại intent quảng bá cụ thể, ứng dụng phải đăng ký broadcast receiver chặn bắt intent này trong tập tin manifest AndroidManifest.xml. Bằng cách này, ứng dụng báo cho Android biết loại intent quảng bá nó quan tâm.

Giả sử ta muốn đăng ký broadcast receiver tên MyReceiver ở trên, để chặn intent báo sự kiện ACTION_BOOT_COMPLETED do hệ thống phát ra khi Android hoàn tất quá trình khởi động, phải khai báo trong tập tin manifest:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name=".MyReceiver"
        android:enabled="true"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </receiver>
</application>
```

Như vậy, khi Android khởi động, intent báo sự kiện ACTION_BOOT_COMPLETED sẽ bị chặn bởi broadcast receiver MyReceiver, code xử lý cài đặt trong phương thức onReceive() của lớp MyReceiver sẽ chạy để đáp ứng sự kiện đó.

Bạn cũng có thể đăng ký động (dùng code) một broadcast receiver bằng phương thức Context.registerReceiver().

```
MyReceiver receiver = new MyReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction(BOOT_COMPLETED);
filter.addCategory(Intent.CATEGORY_DEFAULT);
registerReceiver(receiver, filter);
```

Một số sự kiện của hệ thống được định nghĩa như các trường final static trong lớp Intent:

Hằng sự kiện	Mô tả
android.intent.action.BATTERY_CHANGED	Intent sticky chứa trạng thái sạc, mức độ pin và các thông tin khác về pin.
android.intent.action.BATTERY_LOW	Báo tình trạng pin yếu trên thiết bị.
android.intent.action.BATTERY_OKAY	Báo tình trạng pin đủ sau tình trạng pin yếu trước đó.

android.intent.action.BOOT_COMPLETED	Quảng bá một lần sau khi hệ thống khởi động (boot).
android.intent.action.BUG_REPORT	Hiển thị activity thông báo có lỗi.
android.intent.action.CALL	Thực hiện cuộc gọi, chỉ định bởi phần Data của intent.
android.intent.action.CALL_BUTTON	Người dùng nhấn nút Call hoặc giao diện tương đương để đặt cuộc gọi.
android.intent.action.DATE_CHANGED	Bảo ngày có thay đổi.
android.intent.action.REBOOT	Báo thiết bị khởi động lại.

3. Tùy biến intent quảng bá

Broadcast receiver nhận các intent quảng bá. Một intent bình thường trở thành intent quảng bá khi bạn dùng các phương thức quảng bá để phát chúng đi. Có hai phương thức quảng bá:

- Quảng bá thông thường, gửi với phương thức `Context.sendBroadcast()`. Gửi nhận không đồng bộ, các receiver chạy không theo thứ tự (thường là cùng lúc). Nếu bạn chỉ gửi intent quảng bá đến các đơn thể trong ứng dụng, để tối ưu việc phát intent và bảo đảm an toàn cho thông điệp gửi, dùng `sendBroadcast()` của `LocalBroadcastManager`.

- Quảng bá có thứ tự, gửi với phương thức `Context.sendOrderedBroadcast()`, truyền đến một receiver tại một thời điểm, khi receiver nhận, nó có thể truyền kết quả đến receiver kế tiếp hoặc hủy bỏ quảng bá (bằng `abortBroadcast()`) và không truyền tiếp. Nhận theo thứ tự có thể điều khiển với thuộc tính `android:priority` của `intent-filter` so trùng, các receiver cùng độ ưu tiên sẽ chạy theo thứ tự tùy ý. Bạn xem thêm phần gửi/nhận SMS.

Nếu bạn muốn ứng dụng của bạn tạo và gửi các intent tùy biến riêng, bạn phải tạo và gửi các intent tùy biến này bằng cách dùng phương thức `sendBroadcast()` từ lớp `activity` của bạn.

```
public void broadcastIntent(View view) {
    Intent intent = new Intent();
    intent.setAction("com.twe.examples.CUSTOM_INTENT");
    intent.addCategory(Intent.CATEGORY_DEFAULT);
    sendBroadcast(intent);
}
```

Intent quảng bá `com.twe.examples.CUSTOM_INTENT` phải được đăng ký giống như cách đăng ký các intent do hệ thống sinh ra:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">
        <intent-filter>
            <action android:name="com.twe.examples.CUSTOM_INTENT" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </receiver>
</application>
```

Nếu bạn dùng phương thức `sendStickyBroadcast(Intent)`, intent sẽ được gọi là *sticky*. Một intent sau khi gửi và được xử lý bởi hệ thống sẽ không còn tồn tại, trong khi intent sticky vẫn còn "đỉnh" lại sau khi hoàn tất quảng bá. Android sử dụng intent sticky cho một số thông tin hệ thống, ví dụ tình trạng pin được gửi như một intent sticky, được "đỉnh" lại để có thể nhận bất kỳ lúc nào. Xem ví dụ:

```
// đăng ký sự kiện thay đổi tình trạng pin
IntentFilter filter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
// intent sticky thường dùng với BroadcastReceiver null trả về trạng thái pin
Intent batteryStatus = this.registerReceiver(null, filter);
// pin đang sạc hay đã đầy?
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = (status == BatteryManager.BATTERY_STATUS_CHARGING ||
    status == BatteryManager.BATTERY_STATUS_FULL);
boolean isFull = status == BatteryManager.BATTERY_STATUS_FULL;
// cách sạc?
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;
```

Trong ví dụ sau, tạo một intent tùy biến rồi tạo một `BroadcastReceiver` để chặn intent đó. Khi quen với việc chặn intent tùy biến, bạn có thể lập trình để chặn intent bất kỳ do hệ thống sinh ra.

Theo các bước sau:

- Trong lớp `activity`, tạo và quảng bá intent tùy biến.

```
// quảng bá một intent tùy biến
public void broadcastIntent(View view) {
    Intent intent = new Intent();
    intent.setAction("com.twe.examples.CUSTOM_INTENT");
    sendBroadcast(intent);
}
```

- Tạo một broadcast receiver tên `MyReceiver`.

```
public class MyReceiver extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
```

```

    Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
}
}

```

- Đăng ký MyReceiver trong tập tin AndroidManifest.xml.

```

<application android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity"
        android:label="@string/title_activity_main">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <receiver android:name="MyReceiver">
        <intent-filter>
            <action android:name="com.twe.examples.CUSTOM_INTENT"/>
        </intent-filter>
    </receiver>
</application>

```

- Phần giao diện, thêm một nút để quảng bá intent tùy biến của bạn:

Trong /res/layout/activity_main.xml:

```

<Button android:id="@+id/btnSendIntent"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/broadcast_intent"
    android:onClick="broadcastIntent"/>

```

Trong /res/values/strings.xml, định nghĩa chuỗi sẽ là nhãn nút.

```

<string name="broadcast_intent">Broadcast Intent</string>

```

Content Provider

Content provider quản lý việc truy cập đến nguồn dữ liệu có cấu trúc. Nó cung cấp giao diện chuẩn để kết nối đến dữ liệu của một ứng dụng từ code chạy trên một ứng dụng khác. Sơ đồ sử dụng dữ liệu thông qua content provider như sau:

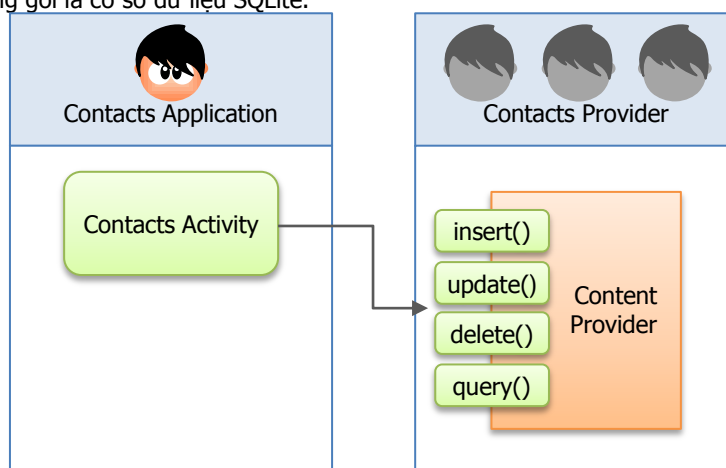
Ứng dụng sử dụng dữ liệu → Content Resolver → Content Provider → Repository của ứng dụng chia sẻ dữ liệu

Các yêu cầu truy cập dữ liệu được thực hiện bởi các phương thức của lớp android.content.ContentResolver.

Nguồn dữ liệu của content provider có thể là:

- Dữ liệu chia sẻ của một ứng dụng, lưu trữ bằng nhiều cách: trong cơ sở dữ liệu, trong tập tin và lưu trữ trên mạng. Nếu ứng dụng của bạn có một content provider, nó sẽ giúp việc truy cập dữ liệu của bạn trở nên dễ dàng hơn cho người phát triển ứng dụng khác.
- Dữ liệu từ các nguồn dữ liệu chuẩn toàn cục như hình ảnh, âm thanh, video, thông tin cá nhân trong contact.

Content provider hoạt động giống như một lớp dữ liệu (data layer) trừu tượng, giúp bạn truy vấn, biên soạn nội dung, thêm và xóa nội dung nguồn dữ liệu do nó cung cấp. Ý tưởng chính là định nghĩa các nguồn cung cấp dữ liệu theo kiểu REST, đóng gói nguồn dữ liệu vào content provider và truy cập bằng *giao diện chuẩn* với các phương thức CRUD. Trong hầu hết các trường hợp, nguồn dữ liệu được đóng gói là cơ sở dữ liệu SQLite.



1. Content URI

Giống như REST, để lấy một content provider, bạn phải chỉ định chuỗi truy vấn dưới dạng một URI theo định dạng sau:

```

<prefix>://<authority>/<path>/<id>

```

Ví dụ:

```

content://com.twe.examples.provider.College/students/12

```

```

content://browser/bookmarks // bookmarks, cần quyền com.android.browser.permission.READ_HISTORY_BOOKMARKS

```

```

content://contacts/people/5 // danh bạ trong điện thoại, cần cấp quyền android.permission.READ_CONTACTS

```

```
content://media/external/images
```

Các thành phần của URI:

Thành phần	Mô tả
prefix	Luôn là content://
authority	Chỉ định tên duy nhất của content provider, ví dụ contacts, browser, v.v... Với content provider do bên thứ ba cung cấp, dùng tên đầy đủ, ví dụ: com.twe.examples.statusprovider.
path	Chỉ định đường dẫn đến dữ liệu được cung cấp. Ví dụ, nếu bạn lấy tất cả contact từ content provider Contacts, thì đường dẫn đến dữ liệu là people và URI sẽ là content://contacts/people.
id	Số dùng chỉ định bản ghi cụ thể được yêu cầu. Ví dụ, nếu bạn muốn xem contact số 5 trong content provider Contacts thì URI sẽ là: content://contacts/people/5.

Có hai kiểu URI:

- URI dựa trên thư mục (directory-based URI) dùng truy cập nhiều bản ghi cùng kiểu.
- URI dựa trên định danh (id-based URI) dùng truy cập một bản ghi chỉ định

URI của các content provider hệ thống, dùng truy cập các nguồn dữ liệu toàn cục:

```
content://media/internal/images
content://media/external/images
content://com.android.contacts/contacts/
```

cũng được định nghĩa trong các lớp hỗ trợ của Android SDK:

```
MediaStore.Images.Media.INTERNAL_CONTENT_URI
MediaStore.Images.Media.EXTERNAL_CONTENT_URI
ContactsContract.Contacts.CONTENT_URI
```

Nghĩa là, thay vì dùng chuỗi truy vấn content://contacts/people/5, bạn có thể dùng:

```
Uri mPeopleBaseUri = ContactsContract.Contacts.CONTENT_URI;
Uri mPersonUri = Uri.withAppendedPath(mPeopleBaseUri, "5");
Cursor cursor = managedQuery(mPersonUri, null, null, null);
```

2. Content provider hệ thống

Android cung cấp một tập các content provider dùng truy cập các nguồn dữ liệu chuẩn toàn cục, gọi là các content provider hệ thống. Danh sách sau là các content provider hệ thống đáng chú ý của Android:

Content Provider	Mô tả
Alarm Clock Provider	Cung cấp truy cập đến cảnh báo cho người dùng và cho phép ứng dụng thao tác cảnh báo hiện có.
Browser Provider	Cho phép truy cập bookmarks và history của người dùng.
Calendar Provider	Cho phép truy cập lịch của người dùng và cho phép ứng dụng chỉnh sửa nó.
Contacts Provider	Cho phép truy cập contact list của người dùng. Hiện dùng lớp mới, ContactsContract.
Call Log Provider	Cung cấp truy cập danh sách cuộc gọi đi và đến của người dùng.
Document Provider	Cung cấp một giao diện chung để truy cập tài liệu của người dùng, như tài liệu lưu giữ trên cloud.
Media Store Provider	Cung cấp truy cập dữ liệu meta của các tập tin media trên thiết bị, như âm nhạc, video và hình ảnh.
Settings Provider	Cung cấp truy cập các preference của thiết bị cấp hệ thống.
Telephony Provider	Cung cấp truy cập dữ liệu điện thoại, như tin nhắn SMS (Short Message Service), MMS (Multimedia Messaging Service) và danh sách APN (Access Point Name).
User Dictionary Provider	Cung cấp truy cập tự điển do người dùng định nghĩa, dùng cho phương thức nhập có gợi ý trước.
Voicemail Provider	Cung cấp truy cập thư thoại của người dùng.

3. Sử dụng content provider

Để dùng một content provider, bạn cần:

- Xác định URI tham chiếu đến content provider và dữ liệu nó chứa.
- Lấy đối tượng ContentResolver bằng phương thức Context.getContentResolver(). Đối tượng này có trách nhiệm tìm ContentProvider chỉ định.
- Gọi các phương thức CRUD của đối tượng ContentResolver để truy cập dữ liệu, các phương thức này sẽ gọi các phương thức chuẩn tương ứng của ContentProvider.

Khi gọi các phương thức của ContentResolver, nếu bạn muốn áp dụng nhiều thao tác CRUD cùng lúc, có thể dùng:

```
applyBatch()    Thực hiện một danh sách các ContentProviderOperation, mỗi đối tượng có URI và kiểu tác vụ riêng.
bulkInsert()    Cho phép chèn một mảng ContentValues cho một directory-based URI. Chỉ cần chỉ định một URI cho tất cả đối tượng muốn thêm vào.
```

Ví dụ, truy cập User Dictionary Provider. Nếu chưa có từ nào trong tự điển, vào Settings > Language & input > Personal Dictionaries rồi thêm từ vào tự điển.

- Lấy các từ trong tự điển:

```
ContentResolver resolver = getContentResolver();
String[] projection = new String[]{ BaseColumns._ID, UserDictionary.Words.WORD };
Cursor cursor = resolver.query(UserDictionary.Words.CONTENT_URI, projection, null, null, null);
if (cursor.moveToFirst()) {
    do {
        long id = cursor.getLong(0);
        String word = cursor.getString(1);
        // xử lý từ lấy được
    } while cursor.moveToNext();
}
```

- Chèn từ mới vào từ điển, sử dụng đối tượng ContentValues chứa các cặp key/value để chèn:

```
ContentValues values = new ContentValues();
values.put(Words.WORD, "computer");
resolver.insert(UserDictionary.Words.CONTENT_URI, values);
```

- Cập nhật từ trong từ điển, cũng dùng đối tượng ContentValues.

```
values.clear();
values.put(Words.WORD, "abacus");
Uri mUri = ContentUris.withAppendedId(Words.CONTENT_URI, id);
long numUpdated = resolver.update(mUri, values, null, null);
```

- Xóa từ trong từ điển.

```
long numDeleted = resolver.delete(Words.CONTENT_URI, Words.WORD + " = ? ", new String[]{ "abacus" });
```

4. Tạo content provider tùy biến

Để tạo content provider tùy biến, chia sẻ nguồn dữ liệu thuộc ứng dụng của bạn, thực hiện các bước đơn giản sau:

- Tạo nguồn dữ liệu để lưu giữ nội dung. Thông thường, Android dùng nguồn dữ liệu là cơ sở dữ liệu SQLite.
- Tạo một lớp content provider, thừa kế từ lớp abstract android.content.ContentProvider:
 - + Định nghĩa URI mà thông qua đó content provider và dữ liệu chứa trong nó được tham chiếu đến.
 - + Cài đặt (override) các phương thức trong giao diện chuẩn của content provider bằng cách thực hiện các truy vấn tương ứng trên cơ sở dữ liệu SQLite. Trong phương thức onCreate(), bạn dùng lớp hỗ trợ DatabaseHelper để tạo hoặc mở cơ sở dữ liệu SQLite.

Các phương thức trong giao diện chuẩn của content provider:

Phương thức	Mô tả
onCreate()	Được gọi bởi Android khi content provider khởi động.
query()	Nhận một yêu cầu từ client. Kết quả trả về như một đối tượng Cursor.
insert()	Chèn nội dung mới vào trong content provider.
delete()	Xóa nội dung trong content provider.
update()	Cập nhật nội dung trong content provider với nội dung mới.
getType()	Trả về kiểu MIME của nội dung tại URI chỉ định.

Các phương thức, trừ onCreate(), có thể gọi nhiều lần từ các thread khác nhau, vì vậy chúng cần cài đặt an toàn thread.

Phương thức getType() có thể trả về:

- + kiểu MIME chuẩn: định dạng type/subtype, ví dụ image/png
- + kiểu MIME tùy biến: không chuẩn, dùng riêng (vendor-specific). Type có hai dạng: vnd.android.cursor.item cho một bản ghi (dòng) và vnd.android.cursor.dir cho nhiều bản ghi (nhiều dòng hoặc một bảng). Subtype chỉ định bởi nhà cung cấp (provider-specific), thường tạo từ authority của content provider với tên bảng. Ví dụ:
vnd.android.cursor.item/vnd.yourcompanyname.contenttype
vnd.android.cursor.dir/vnd.yourcompanyname.contenttype

Trong các phương thức CRUD, phương thức quan trọng là query(), tương tự SQL query, có các tham số sau:

Tham số của query()	Từ khóa/tham số SELECT	Ghi chú
Uri	FROM <i>table_name</i>	Uri ánh xạ bảng trong provider.
projection	<i>col, col, col, ...</i>	projection là mảng các cột trong dòng nhận được.
selection	WHERE <i>col=value</i>	selection chỉ định điều kiện chọn các hàng.
selectionArgs		Chọn các tham số thay thế đặt chỗ ? trong truy vấn SELECT.
sortOrder	ORDER BY <i>col, col, ...</i>	sortOrder chỉ định thứ tự các dòng kết quả trong Cursor trả về.

Với truy vấn đơn giản, các tham số projection, selection và selectionArgs đặt bằng null.

Ví dụ về một truy vấn chi tiết:

```
import ContactsContract.Contacts;
// ...
String[] projection = new String[] {
    Contacts._ID,
    Contacts.DISPLAY_NAME_PRIMARY
};
Uri mContactsUri = ContactsContract.Contacts.CONTENT_URI;
Cursor cursor = managedQuery( mContactsUri,
    projection, // mảng các cột trả về
    "_id=?", // phát biểu WHERE có tham số
    new String[] {12}, // cung cấp trị cho tham số
    Contacts.DISPLAY_NAME_PRIMARY + " ASC"); // phát biểu ORDER BY
```

Bạn nên định nghĩa một interface chứa các hằng số giúp ứng dụng dễ làm việc với content provider:

```
public interface StudentContract {
    String AUTHORITY = "com.twe.studentprovider";
    Uri AUTHORITY_URI = Uri.parse("content://" + AUTHORITY);
    String CONTENT_PATH = "students";
    Uri CONTENT_URI = Uri.withAppendedPath(AUTHORITY_URI, CONTENT_PATH);
    String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.com.twe.studentprovider.students";
    String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.com.twe.studentprovider.students";

    interface StudentColumns extends BaseColumns {
```



```

    String ID = "_id";
    String NAME = "name";
    String GRADE = "grade";
}
}

```

Khi các phương thức của content provider được gọi, một tham số Uri được truyền đến chúng để chỉ định dữ liệu tác vụ sẽ thực hiện trên đó. Uri này có thể tham chiếu đến một dòng cụ thể trong bảng chỉ định, hoặc chỉ định toàn bảng. Phương thức của content provider sẽ xác định loại Uri và có hành động phù hợp.

Để hỗ trợ xác định Uri, thực thể UriMatcher được dùng. Khi khởi tạo UriMatcher, nó được cấu hình để trả về một số nguyên chỉ định tương ứng với loại Uri mà nó so trùng. Trong ví dụ sau, nó trả về QUERY_ALL_STUDENTS (1) nếu Uri tham chiếu toàn bảng và trả về QUERY_BY_STUDENT_ID (2) nếu Uri tham chiếu một dòng của bảng.

Lớp content provider, thừa kế ContentProvider, cài đặt các phương thức chuẩn:

```

public class StudentProvider extends ContentProvider {
    private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

    // UriMatcher
    static final int QUERY_ALL_STUDENTS = 1;
    static final int QUERY_BY_STUDENT_ID = 2;
    static final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
    static {
        matcher.addURI(StudentContract.AUTHORITY, StudentContract.CONTENT_PATH, QUERY_ALL_STUDENTS);
        matcher.addURI(StudentContract.AUTHORITY, StudentContract.CONTENT_PATH + "/#", QUERY_BY_STUDENT_ID);
    }

    // khai báo cho database. Nguồn dữ liệu của content provider
    private SQLiteDatabase database;
    static final String DATABASE_NAME = "College";
    static final String STUDENTS_TABLE_NAME = "students";
    static final int DATABASE_VERSION = 1;
    static final String CREATE_DB_TABLE =
        "CREATE TABLE " + STUDENTS_TABLE_NAME + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        " name TEXT NOT NULL, " + " grade TEXT NOT NULL);";

    // lớp nội hỗ trợ, thực hiện truy cập database SQLite cho content provider
    private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override public void onCreate(SQLiteDatabase db) {
            db.execSQL(CREATE_DB_TABLE);
        }

        @Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS " + STUDENTS_TABLE_NAME);
            onCreate(db);
        }
    }

    @Override public boolean onCreate() {
        Context context = getContext();
        database = new DatabaseHelper(context).getWritableDatabase();
        return (database != null);
    }

    @Override public Cursor query(Uri uri, String[] projection, String selection,
                                String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
        qb.setTables(STUDENTS_TABLE_NAME);
        switch (matcher.match(uri)) {
            case QUERY_ALL_STUDENTS:
                qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
                break;
            case QUERY_BY_STUDENT_ID:
                qb.appendWhere(StudentContract.StudentColumns.ID + "=" + uri.getPathSegments().get(1));
                break;
            default: throw new IllegalArgumentException("Unknown URI " + uri);
        }
        if (sortOrder == null || sortOrder.isEmpty()) {
            sortOrder = StudentContract.StudentColumns.NAME;
        }
    }
}

```

```

    }
    Cursor cursor = qb.query(database, projection, selection, selectionArgs, null, null, sortOrder);
    // đăng ký để theo dõi content URI thay đổi
    cursor.setNotificationUri(getContext().getContentResolver(), uri);
    return cursor;
}

@Override public Uri insert(Uri uri, ContentValues values) {
    long rowID = database.insert(STUDENTS_TABLE_NAME, "", values);
    if (rowID > 0) {
        Uri mUri = ContentUris.withAppendedId(StudentContract.CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(mUri, null);
        return mUri;
    }
    throw new SQLException("Failed to add a record into " + mUri);
}

@Override public int delete(Uri uri, String selection, String[] selectionArgs) {
    int count = 0;
    switch (matcher.match(uri)) {
        case QUERY_ALL_STUDENTS:
            count = database.delete(STUDENTS_TABLE_NAME, selection, selectionArgs);
            break;
        case QUERY_BY_STUDENT_ID:
            count = database.delete(STUDENTS_TABLE_NAME,
                StudentContract.StudentColumns.ID + " = " + uri.getPathSegments().get(1) +
                (!TextUtils.isEmpty(selection) ? " AND (" + selection + ')' : ""), selectionArgs);
            break;
        default: throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}

@Override public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    int count = 0;
    switch (matcher.match(uri)) {
        case QUERY_ALL_STUDENTS:
            count = database.update(STUDENTS_TABLE_NAME, values, selection, selectionArgs);
            break;
        case QUERY_BY_STUDENT_ID:
            count = database.update(STUDENTS_TABLE_NAME, values,
                StudentContract.StudentColumns.ID + " = " + uri.getPathSegments().get(1) +
                (!TextUtils.isEmpty(selection) ? " AND (" + selection + ')' : ""), selectionArgs);
            break;
        default: throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}

@Override public String getType(Uri uri) {
    switch (matcher.match(uri)) {
        case QUERY_ALL_STUDENTS:
            return StudentContract.CONTENT_TYPE;
        case QUERY_BY_STUDENT_ID:
            return StudentContract.CONTENT_ITEM_TYPE;
        default: throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
}

```

- Cuối cùng, *đăng ký* content provider trong tập tin AndroidManifest.xml bằng cách dùng element <provider>.

```

<application android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <provider android:name=".StudentProvider"
        android:authorities="com.twe.studentprovider">
        android:enabled="true"
        android:exported="true" />
</application>

```

Thuộc tính `android:authorities` là quan trọng nhất, nó định nghĩa tên duy nhất của content provider. Android sẽ lưu một tham chiếu đến content provider theo tên này. Nó được dùng như một phần của URI mà ứng dụng khác dùng truy cập content provider: `content://com.twe.studentprovider/student/1`

Giống như service và các đơn thể dùng chung khác, bạn có thể thiết lập quyền truy cập cho content provider của bạn. Content provider cho phép khai báo tách biệt hai quyền đọc và ghi, chỉ định bởi các thuộc tính:

`android:readPermission` chỉ ứng dụng cho phép mới có thể gọi phương thức query.

`android:writePermission` chỉ ứng dụng cho phép mới có thể gọi phương thức insert, update, delete.

Nếu không cần khai báo tách biệt, thuộc tính `android:permission` dùng chỉ định quyền truy cập đến content provider.

- Từ activity, trong cùng ứng dụng hoặc thuộc ứng dụng khác, dùng `ContentResolver` truy cập đến content provider:

```
public class MainActivity extends Activity {
    EditText mName;
    EditText mGrade;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mName = ((EditText)findViewById(R.id.txtName));
        mGrade = ((EditText)findViewById(R.id.txtGrade));
    }

    // xử lý sự kiện tap nút Add
    public void onClickAddName(View view) {
        // thêm một record student mới
        ContentValues values = new ContentValues();
        values.put(StudentContract.StudentColumns.NAME, mName.getText().toString());
        values.put(StudentContract.StudentColumns.GRADE, mGrade.getText().toString());
        Uri mUri = getContentResolver().insert(StudentContract.CONTENT_URI, values);
        Toast.makeText(getBaseContext(), mUri.toString(), Toast.LENGTH_LONG).show();
    }

    // xử lý sự kiện tap nút Retrieve
    public void onClickRetrieveStudents(View view) {
        // lấy các record student
        Cursor cursor = getContentResolver().query(StudentContract.CONTENT_URI, null, null, null, "name");
        if (cursor.moveToFirst()) {
            do {
                Toast.makeText(this,
                    cursor.getString(cursor.getColumnIndex(StudentContract.StudentColumns.ID))
                    + ", " + cursor.getString(cursor.getColumnIndex(StudentContract.StudentColumns.NAME))
                    + ", " + cursor.getString(cursor.getColumnIndex(StudentContract.StudentColumns.GRADE)),
                    Toast.LENGTH_SHORT).show();
            } while (cursor.moveToNext());
        }
        cursor.close();
    }
}
```

Layout và View

Khối xây dựng cơ bản cho giao diện người dùng là một đối tượng view, được tạo từ lớp View, chiếm một vùng hình chữ nhật trên màn hình ứng dụng của bạn, nó chịu trách nhiệm vẽ và xử lý sự kiện.

View là lớp cơ sở được sử dụng để tạo các thành phần giao diện tương tác được, như TextView, EditText, ListView, hoặc ImageView, v.v... chúng sẽ được đặt vào layout của bạn.

ViewGroup là lớp con của View, cung cấp một container vô hình chứa các View hoặc ViewGroup khác, đồng thời định nghĩa các thuộc tính layout cho chúng.

Một số view phức tạp được gọi là widget, ví dụ android.widget.TextView, android.widget.TimePicker, v.v... Bạn cần phân biệt chúng với home-screen widget, là một nhóm các View và ViewGroup mà bạn có thể đặt trên màn hình Home và được cập nhật thường xuyên.

Layout

Các layout là lớp con của ViewGroup, định nghĩa bố trí trực quan cho giao diện người dùng. Một layout có thể chứa nhiều view hoặc các layout khác, tạo thành một *cây view phân cấp*. Kiến trúc các layout lồng nhau này cho phép bạn thiết kế nên các giao diện phức tạp hơn.

Android sẽ duyệt cây view phân cấp theo thứ tự pre-order rồi vẽ lần lượt các view trên cây view phân cấp ra màn hình.

Android cung cấp hai cách tiếp cận để khai báo layout và các thành phần của nó:

- Khai báo tĩnh trong tập tin XML tại thư mục /res/layout của project. Điều này cho phép tách biệt phần presentation và phần business của ứng dụng.
- Khai báo động bằng code trong thời gian chạy, ví dụ trong game.

1. Tập tin layout XML

a) Đơn vị đo

Khi thiết lập các thuộc tính cho layout, Android cho phép dùng các đơn vị đo lường sau:

dp *density-independent pixel*. 1 dp tương đương với một điểm ảnh trên một màn hình 160 dpi. Đây là đơn vị được đề nghị khi chỉ định kích thước của view trong một layout. Cách ghi chú khác: **dip**.

sp *scale-independent pixel*. Tương tự như dp và được đề nghị dùng chỉ định cho kích cỡ font chữ.

pt *point*. Một pt được định nghĩa bằng 1/72 của một inch, dựa trên kích thước vật lý của màn hình.

px *pixel*. Tương ứng với điểm ảnh thực trên màn hình. Đơn vị đo này, cùng với **mm** (milimeters) và **in** (inches) không được khuyến khích, do UI của bạn có thể không chính xác trên các thiết bị với độ phân giải màn hình khác nhau, nên dùng dp.

Khi thiết lập giao diện bằng code, một số phương thức nhận đối số kích thước là pixel. Bạn buộc phải chuyển trị x_dip tính bằng dip thành x_px tính bằng pixel:

```
int x_px = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, x_dip,
                                           getResources().getDisplayMetrics());
mEditText.setWidth(x_px);
```

Android định nghĩa bốn mật độ màn hình:

- mật độ thấp (ldpi) - 120 dpi
- mật độ trung bình (mdpi) - 160 dpi
- mật độ cao (hdpi) - 240 dpi
- mật độ cực cao (xhdpi) - 320 dpi

Điện thoại của bạn sẽ rơi vào một trong những mật độ được xác định trong danh sách trên. Ví dụ, Nexus S được coi là một thiết bị hdpi, do mật độ điểm ảnh của nó gần 240 dpi. HTC Hero, có (đường chéo) màn hình 3.2 inch và độ phân giải 320 - 480 mật độ điểm ảnh của nó khoảng 180 dpi, do đó nó sẽ được coi là một thiết bị mdpi.

b) Thuộc tính của layout

Mỗi layout có một tập định nghĩa các thuộc tính trực quan của layout đó.

Có nhiều thuộc tính dùng chung cho các loại layout khác nhau:

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:layout_width	Chiều rộng của khung vô hình bao quanh layout.
android:layout_height	Chiều cao của khung vô hình bao quanh layout.
android:layout_marginTop	Lề trên của layout.
android:layout_marginBottom	Lề dưới của layout.
android:layout_marginLeft	Lề trái của layout.
android:layout_marginRight	Lề phải của layout.
android:layout_gravity	Định vị của view trong cha.
android:gravity	Định vị nội dung/view con bên trong view.
android:layout_weight	Trọng số chỉ định tỷ lệ không gian view sẽ chiếm, mặc định là 0. Phần không gian sẽ chiếm bằng trọng số của view chia cho tổng các trọng số.
android:layout_x	Tọa độ x của layout.
android:layout_y	Tọa độ y của layout.
android:paddingLeft	Đệm trái của layout.
android:paddingRight	Đệm phải của layout.
android:paddingTop	Đệm trên của layout.
android:paddingBottom	Đệm dưới của layout.

Hình bên mô tả trực quan các thuộc tính margin và padding:



Ở đây, width và height là các chiều của layout/view chỉ định bằng dp, sp, pt, px, mm và in.

Bạn có thể chỉ định width và height với kích thước chính xác, nhưng nên dùng một trong các hằng được thiết lập cho width hoặc height:

wrap_content bảo rằng view sẽ có kích thước theo kích thước nội dung chứa trong nó. Ví dụ, `LinearLayout`, orientation là `vertical`, width của view là `wrap_content`, thì chiều ngang view sẽ chiếm toàn cột.

fill_parent bảo rằng view sẽ có kích thước giãn ra lớn bằng view cha đang chứa nó.

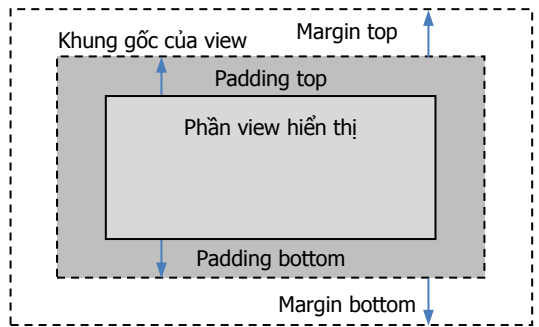
match_parent từ API Level 8+, `fill_parent` gọi là `match_parent` để mô tả tốt hơn: kích thước so bằng cha chứa nó.

Designer Tool của Android Studio hỗ trợ thiết lập các trị này cho view rất thuận tiện, dùng các icon  và  trên toolbar.

Thuộc tính `layout_weight` được dùng như sau: phần không gian sẽ chiếm bằng weight của view chỉ định chia cho tổng các weight. Nếu không chỉ định, weight mặc định là 0.

Thuộc tính `layout_gravity` và gravity rất quan trọng khi định vị view, có thể nhận nhiều trị như bảng sau, kết hợp bởi dấu '|':

Biên của view với các view khác



Hằng	Trị	Mô tả
top	0x30	Đặt đối tượng sát trên cùng trong container, không thay đổi kích thước.
bottom	0x50	Đặt đối tượng sát dưới cùng trong container, không thay đổi kích thước.
left	0x03	Đặt đối tượng sát bên trái trong container, không thay đổi kích thước.
right	0x05	Đặt đối tượng sát bên phải trong container, không thay đổi kích thước.
center_vertical	0x10	Đặt đối tượng canh giữa theo chiều dọc trong container, không thay đổi kích thước.
fill_vertical	0x70	Kích thước theo chiều dọc của đối tượng giãn ra theo kích thước container chứa nó.
center_horizontal	0x01	Đặt đối tượng canh giữa theo chiều ngang trong container, không thay đổi kích thước.
fill_horizontal	0x07	Kích thước theo chiều ngang của đối tượng giãn ra theo kích thước container chứa nó.
center	0x11	Đặt đối tượng canh giữa theo cả hai chiều, không thay đổi kích thước.
fill	0x77	Kích thước theo cả hai chiều của đối tượng giãn ra theo kích thước container chứa nó.
clip_vertical	0x80	Hiệu chỉnh kích thước chiều dọc của đối tượng để không làm giãn container chứa nó.
clip_horizontal	0x08	Hiệu chỉnh kích thước chiều ngang của đối tượng để không làm giãn container chứa nó.
start	0x00800003	Đẩy đối tượng đến nơi bắt đầu của container chứa nó, không thay đổi kích thước.
end	0x00800005	Đẩy đối tượng đến cuối container chứa nó, không thay đổi kích thước.

c) Định danh của View

Một view có một định danh (ID) duy nhất gán cho nó, dùng để xác định một nút duy nhất trên cây view phân cấp. Cú pháp cho nó trong tập tin layout XML là:

```
android:id="@+id/button_id"
```

Cú pháp tham chiếu tài nguyên như sau: @[package:]type/name

- Ký tự (@) bắt đầu chuỗi chỉ định rằng XML parser sẽ phân tích và dùng phần còn lại như định danh *tham chiếu* đến nguồn tài nguyên (resource ID).

- Ký tự (+) nghĩa là đây là một định danh tài nguyên mới được *tạo ra* và thêm vào tài nguyên của project. Để tạo một thực thể của đối tượng view và lấy nó từ tập tin XML layout, ví dụ `activity_main.xml`:

```
setContentView(R.layout.activity_main);
Button myButton = (Button) findViewById(R.id.button_id);
```

Như vậy, `R.id` là tập hợp các tài nguyên được định danh định nghĩa trong tập tin layout XML.

Trong tập tin layout XML, các view có resource ID dùng để tham chiếu trong cây view, được thiết lập bằng thuộc tính `android:id` của view. Tuy nhiên, nếu view được tạo bằng code mà bạn cần đến resource ID của nó, bạn phải định nghĩa riêng:

- Định nghĩa một resource ID cho view, ví dụ đặt tên ID là `pager_id`, trong tập tin `/res/values/ids.xml`:

```
<resources>
    <item type="id" name="pager_id" />
</resources>
```

- Sau đó, ID đó sẽ có mặt trong tập tin tài nguyên `R.java` và bạn có thể gán ID cho view, bằng code:

```
ViewPager view = new ViewPager(this);
view.setId(R.id.pager_id);
```

API Level 17 cung cấp phương thức `View.generateViewId()` dùng sinh ID duy nhất cho view.

d) include và merge

Đa số UI của ứng dụng Android được định nghĩa bằng cách dùng tập tin XML gọi là tập tin layout. Bạn có thể nghĩ layout như một template mà bạn lắp vào đó các view khác nhau hoặc các layout con. Các view lại tham chiếu đến các tài nguyên khác như chuỗi, màu sắc, các đối tượng vẽ được (drawable).

Mỗi tập tin layout XML có thể định nghĩa chỉ một phần của UI, nên có thể mở rộng thành tập tin layout phức tạp hơn:

- Tag `<include>` dùng để thêm vào layout bạn muốn dùng lại, ví dụ `/res/layout/title.xml`, bên trong layout chính:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```



```

android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center_horizontal">
<include android:id="@+id/news_title"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    layout="@layout/title"/>
<!-- các view khác -->
</LinearLayout>

```

Các thuộc tính android:layout_* trong tag <include> sẽ ghi đè lên thuộc tính của layout gốc được dùng lại.

- Tag <merge>, chỉ áp dụng như tag root (tag ngoài cùng) của một tập tin layout XML, dùng để tối ưu layout của Android bằng cách thu giảm số cấp trên cây view phân cấp.

Ví dụ: activity chính "inflate" từ activity_main.xml. Tập tin layout XML này dùng tag <merge> thay thế tag <FrameLayout>, và có một view tùy biến OkCancelButton, thừa kế từ LinearLayout:

```

<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:okCancelButton="http://schemas.android.com/apk/res/com.twe.examples.merge">
    <ImageView android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="center"
        android:src="@drawable/micheal" />
    <com.twe.examples.merge.OkCancelButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:paddingTop="8dip"
        android:gravity="center_horizontal"
        android:background="#AA000000"
        okCancelButton:okLabel="Save"
        okCancelButton:cancelLabel="Don't save" />
</merge>

```

Layout XML okcancelbar.xml của view tùy biến cũng dùng tag <merge> thay cho <LinearLayout>:

```

<merge xmlns:android="http://schemas.android.com/apk/res/android">
    <include layout="@layout/okcancelbar_button"
        android:id="@+id/okcancelbar_ok" />
    <include layout="@layout/okcancelbar_button"
        android:id="@+id/okcancelbar_cancel" />
</merge>

```

View tùy biến OkCancelButton "inflate" từ okcancelbar.xml.

```

public class OkCancelButton extends LinearLayout {
    public OkCancelButton(Context context, AttributeSet attrs) {
        super(context, attrs);
        setOrientation(HORIZONTAL);
        setGravity(Gravity.CENTER);
        setWeightSum(1.0f);

        LayoutInflater.from(context).inflate(R.layout.okcancelbar, this, true);
        TypedArray array = context.obtainStyledAttributes(attrs, R.styleable.OkCancelButton, 0, 0);

        String text = array.getString(R.styleable.OkCancelButton_okLabel);
        if (text == null) text = "Ok";
        ((Button) findViewById(R.id.okcancelbar_ok)).setText(text);
        text = array.getString(R.styleable.OkCancelButton_cancelLabel);
        if (text == null) text = "Cancel";
        ((Button) findViewById(R.id.okcancelbar_cancel)).setText(text);
        array.recycle();
    }
}






```


Nếu dùng Hierarchy Viewer để xem hai phiên bản có dùng và không dùng tag <merge>, bạn sẽ thấy các view con trong tag <merge> gần trực tiếp view cha, kết quả là giảm bớt một số cấp trên cây view phân cấp.

Tạo layout có thể thực hiện thủ công trong tập tin layout XML, tuy nhiên nên kết hợp sử dụng các công cụ trực quan sau:

- Designer Tool: công cụ của Android Studio là tập hợp một số view của IDE giúp thiết kế trực quan (WYSIWYG).

+ Palette: panel bên trái, chứa tất cả widget được tổ chức thành nhóm. Bạn có thể kéo thả chúng vào layout, hoặc chọn widget rồi click vào vị trí trên layout để đặt chúng.

+ Device Screen: cung cấp hiển thị trực quan của giao diện đang thiết kế. Cho phép bạn thao tác trực tiếp các widget trong thiết kế (chọn, xóa, di chuyển, thay đổi kích thước). Mô hình thiết bị được thể hiện có thể được thay đổi nhiều cách bởi toolbar ngay trên Device Screen. Toolbar có các công cụ cho phép phóng to , thu nhỏ , phóng vừa kích thước , thiết lập layout_width  và layout_height , chuyển từ portrait sang landscape, chuyển hiển thị theo API Level, ...

Màn hình hiển thị thường có khung thiết bị (device frame) bao quanh. Bạn có thể tắt bằng cách chọn icon  ngoài cùng bên phải của toolbar và bỏ chọn Include Device Frames (if available).

View kèm theo dấu 💡 (light bulb) cho thấy có vấn đề khi thiết kế, click lên nó sẽ nhận được giải pháp đề nghị.

Một widget trong thiết kế được hỗ trợ tùy chỉnh mạnh. Bạn có thể chuyển nó thành loại widget khác bằng cách chọn Morphing trong menu tắt, đặt id và thiết lập text trong tài nguyên cho nó bằng cách double-click vào widget.

+ Preview: Designer Tool có hai chế độ thể hiện trong hai tab ngay dưới Device Screen: Design và Text. Tab Text cho phép xem source của layout, khi chuyển qua chế độ này, panel Preview bên phải cho phép xem kết quả thay đổi trực tiếp trong source của layout.

+ Component Tree: trình bày cách widget được tổ chức trong layout như một cây phân cấp trực quan. Các nút trong cây phân cấp của nó có thể kéo thả giúp thay đổi linh hoạt kiến trúc, tổ chức lại layout. Chọn một widget trong thiết kế tương đương chọn một nút trong cây và ngược lại.

+ Properties: widget trong thiết kế liên kết với một tập các thuộc tính. Khi chọn widget, bạn có thể xem và hiệu chỉnh thuộc tính của widget đó trong Properties.

- Hierarchy Viewer là công cụ dùng nhận dạng mối quan hệ phân cấp trong layout. Nó hiển thị cấu trúc UI của màn hình hiện hành trên AVD hoặc trên thiết bị thật, thành một cây phân cấp.

Hierarchy Viewer hỗ trợ thiết kế, dò lỗi, kiểm tra, tạo tài liệu cho giao diện người dùng. Có thể gọi Hierarchy Viewer bằng cách chạy phiên bản độc lập tại sdk/tools/hierarchyviewer.bat, phiên bản này đã lạc hậu, thay thế bằng monitor.bat.

Trong Android Studio, click icon Android Device Monitor trên toolbar để mở cửa sổ Android Device Monitor, trong cửa sổ này chọn Window > Open Perspective... và chọn Hierarchy View.

2. Layout tĩnh

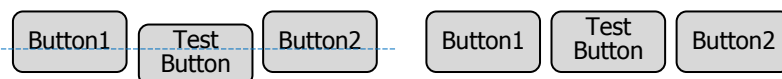
Android cung cấp một số kiểu layout, được sử dụng phối hợp với nhau trong khi tạo UI cho các ứng dụng Android.

Layout	Mô tả
LinearLayout	LinearLayout (box model) là một ViewGroup, sắp xếp các View con theo một hướng, ngang hoặc dọc.
RelativeLayout	RelativeLayout (rule-based model) là một ViewGroup, sắp xếp các View con với vị trí liên quan đến cha hoặc View con khác.
TableLayout	TableLayout (grid model) nhóm các View thành hàng và cột.
AbsoluteLayout	AbsoluteLayout cho phép chỉ định vị trí chính xác cho các con của nó.
FrameLayout	FrameLayout cho phép các view con có thể chồng lên nhau.
ListView	ListView là một ViewGroup hiển thị như một danh sách chứa các mục có thể cuộn được.
GridView	GridView là một ViewGroup hiển thị như một lưới hai chiều có thể cuộn được.
ScrollView	ScrollView là một ViewGroup thiết kế để hỗ trợ cuộn được nội dung chứa trong nó.

a) LinearLayout

LinearLayout là một ViewGroup, sắp xếp các view con một cách tuyến tính theo một hướng, horizontal (ngang, mặc định) hoặc vertical (dọc), tùy theo thuộc tính quan trọng android:orientation.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:baselineAligned	Trị boolean, true hoặc false. Đôi khi đặt false để ngăn canh hàng đường nền của nội dung. (xem hình dưới)
android:divider	Một drawable dùng như ngăn cách dọc giữa các nút. Dùng một trị màu có định dạng #rgb, #argb, #rrgb hoặc #aarrgbb.
android:gravity	Chỉ định cách định vị nội dung bên trong, trên cả hai trục X và Y. Trị định vị là top, bottom, left, right, center, center_vertical, center_horizontal, .v.v...
android:orientation	Thuộc tính quan trọng, chỉ định hướng sắp xếp của các view con, dùng horizontal cho sắp xếp thành hàng (mặc định), vertical cho sắp xếp thành cột.



Trái: layout bị phá vỡ do canh hàng đường nền

Phải: đặt android:baselineAligned="false" để ngăn canh hàng đường nền

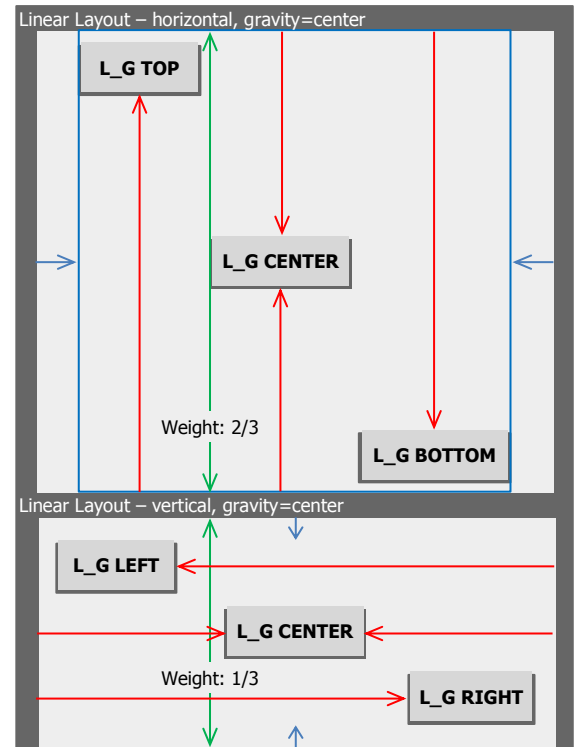
Ví dụ dưới cho thấy tác động các thuộc tính trong LinearLayout: android:gravity (xanh), android:layout_gravity (đỏ) và android:layout_weight (xanh lá).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#666666"
    android:orientation="vertical">
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Linear Layout - horizontal, gravity=center"
        android:textColor="#FFFFFF"/>
    <LinearLayout android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="2"
        android:background="#EEEEEE"
        android:gravity="center"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:orientation="horizontal">
```

```

<Button android:id="@+id/button_01_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:text="l_g top"/>
<Button android:id="@+id/button_02_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="l_g center"/>
<Button android:id="@+id/button_03_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:text="l_g bottom"/>
</LinearLayout>
<TextView android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="2dip"
    android:text=
    "Linear Layout - vertical, gravity=center"
    android:textColor="#FFFFFF"/>
<LinearLayout android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="1"
    android:background="#CCCCCC"
    android:layout_marginLeft="15dp"
    android:layout_marginRight="15dp"
    android:gravity="center"
    android:orientation="vertical">
    <Button android:id="@+id/button_04_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:text="l_g left"/>
    <Button android:id="@+id/button_05_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="l_g center"/>
    <Button android:id="@+id/button_06_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="l_g right"/>
</LinearLayout>
</LinearLayout>

```



b) RelativeLayout

Đa số UI có thể định nghĩa đơn giản bằng cách lồng nhiều LinearLayout, tuy nhiên tập tin layout sẽ trở nên phức tạp, khó bảo trì, khó hiệu chỉnh. Để thiết kế UI phức tạp, Android cung cấp RelativeLayout, cho phép tổ chức các view theo mối liên quan giữa view đó với view khác. Vị trí của mỗi view sẽ tham chiếu đến view bên cạnh hoặc cha của nó.

id=F toLeftOf E above D	id=E center_horizontal ParentTop	id=G toRightOf E above B
id=D center_vertical ParentLeft	id=A Center	id=B center_vertical ParentRight
id=I toLeftOf C below D	id=C center_horizontal ParentBottom	id=H toRightOf C below B

Khi thiết lập quan hệ, các view dùng tham chiếu được chỉ định bằng android:id, vì vậy các view này phải được liệt kê trong tập tin layout XML trước để có thể tham chiếu từ chúng được.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:gravity	Chỉ định cách định vị nội dung bên trong view, trên cả hai trục X và Y. Trị định vị là top, bottom, left, right, center, center_vertical, center_horizontal, .v.v...

`android:ignoreGravity` Chỉ định view không chịu ảnh hưởng của gravity.

Một view mốc (anchor) được chỉ định bằng id. Bạn định nghĩa vị trí view của bạn bằng cách dùng các thuộc tính của `RelativeLayout.LayoutParams`, để thể hiện mối liên quan giữa view của bạn với view mốc.

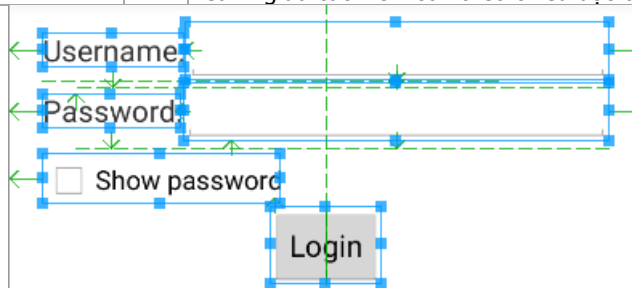
@Override

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    RelativeLayout mLayout = new RelativeLayout(this);
    // EditText
    RelativeLayout.LayoutParams paramsEditText = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.MATCH_PARENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
    paramsEditText.addRule(RelativeLayout.ALIGN_PARENT_TOP, RelativeLayout.TRUE);
    paramsEditText.addRule(RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.TRUE);
    paramsEditText.setMargins(10, 10, 0, 10);
    EditText mEditText = new EditText(this);
    mEditText.setId(text.generateViewId()); // API Level 17
    // Button
    RelativeLayout.LayoutParams paramsButton = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.WRAP_CONTENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
    paramsButton.addRule(RelativeLayout.BELOW, text.getId());
    paramsButton.addRule(RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.TRUE);
    Button mButton = new Button(this);
    mButton.setText(getString(R.string.button_label));

    mLayout.addView(mEditText, paramsEditText);
    mLayout.addView(mButton, paramsButton);
    setContentView(mLayout);
}
```

Trong trường hợp view mốc không hiển thị, thuộc tính `android:layout_alignWithParentIfMissing` được đặt bằng true để chỉ rằng view cha sẽ được dùng như view mốc.

Thuộc tính	Mô tả
<code>android:layout_above</code>	Định vị cạnh dưới của view này nằm trên view mốc.
<code>android:layout_alignBaseline</code>	Canh hàng đường nền của view này với đường nền của view mốc.
<code>android:layout_alignBottom</code>	Canh hàng cạnh dưới của view này với cạnh dưới của view mốc.
<code>android:layout_alignEnd</code>	Canh hàng cạnh sau của view này với cạnh sau của view mốc.
<code>android:layout_alignLeft</code>	Canh hàng cạnh trái của view này với cạnh trái của view mốc.
<code>android:layout_alignRight</code>	Canh hàng cạnh phải của view này với cạnh phải của view mốc.
<code>android:layout_alignStart</code>	Canh hàng cạnh trước của view này với cạnh trước của view mốc.
<code>android:layout_alignTop</code>	Canh hàng cạnh trên của view này với cạnh trên của view mốc.
<code>android:layout_below</code>	Định vị cạnh trên của view này nằm dưới view mốc.
<code>android:layout_toEndOf</code>	Định vị cạnh trước của view này so trùng với cạnh sau của view mốc.
<code>android:layout_toLeftOf</code>	Định vị cạnh phải của view này so trùng với cạnh trái của view mốc.
<code>android:layout_toRightOf</code>	Định vị cạnh trái của view này so trùng với cạnh phải của view mốc.
<code>android:layout_toStartOf</code>	Định vị cạnh sau của view này so trùng với cạnh trước của view mốc.
<code>android:layout_alignParentBottom</code>	Canh hàng cạnh dưới của view này với cạnh dưới của view cha.
<code>android:layout_alignParentEnd</code>	Canh hàng cạnh sau của view này với cạnh sau của view cha.
<code>android:layout_alignParentLeft</code>	Canh hàng cạnh trái của view này với cạnh trái của view cha.
<code>android:layout_alignParentRight</code>	Canh hàng cạnh phải của view này với cạnh phải của view cha.
<code>android:layout_alignParentStart</code>	Canh hàng cạnh trước của view này với cạnh trước của view cha.
<code>android:layout_alignParentTop</code>	Canh hàng cạnh trên của view này với cạnh trên của view cha.
<code>android:layout_centerHorizontal</code>	Canh giữa các view con theo chiều ngang bên trong view cha của chúng.
<code>android:layout_centerInParent</code>	Canh giữa các view con theo chiều ngang hoặc chiều dọc bên trong view cha của chúng.
<code>android:layout_centerVertical</code>	Canh giữa các view con theo chiều dọc bên trong view cha của chúng.



RelativeLayout trong Designer Tool

Các mũi tên màu xanh mô tả quan hệ giữa các view. Các đường đứt màu xanh lá là các đường mốc.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```

        android:padding="20dp">
<TextView android:id="@+id/label_username_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:textSize="18sp"
    android:text="Username: " />
<EditText android:id="@+id/edit_username_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignBaseline="@id/label_username_id"
    android:layout_toRightOf="@id/label_username_id"
    android:inputType="text" />
<TextView android:id="@+id/label_password_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@id/edit_username_id"
    android:textSize="18sp"
    android:text="Password: " />
<EditText android:id="@+id/edit_password_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/edit_username_id"
    android:layout_alignParentRight="true"
    android:layout_alignBaseline="@id/label_password_id"
    android:layout_toRightOf="@id/label_password_id"
    android:inputType="textPassword" />
<CheckBox android:id="@+id/show_password_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show password"
    android:layout_alignParentLeft="true"
    android:layout_below="@id/edit_password_id" />
<Button android:id="@+id/button_login_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Login"
    android:layout_below="@id/show_password_id"
    android:layout_centerHorizontal="true" />
</RelativeLayout>

```

c) TableLayout

TableLayout nhóm các view vào các hàng và cột, không hiển thị biên (border line). Ta dùng element <TableRow> để tạo một dòng trong bảng. Mỗi dòng có 0 đến nhiều cell, mỗi cell giữ một View hoặc ViewGroup. Chiều rộng của mỗi cột xác định bởi chiều rộng của cell rộng nhất.

Để view mở rộng trên nhiều cột, dùng thuộc tính android:layout_span cho view. Không mở rộng trên nhiều dòng được.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:collapseColumns	Chỉ định chỉ số (tính từ 0) các cột cần ẩn. Danh sách các cột tách bởi dấu phẩy, ví dụ 1, 2, 5.
android:shrinkColumns	Chỉ định chỉ số (tính từ 0) các cột cần co lại để phù hợp với màn hình, do view có nội dung làm kích thước vượt ra khỏi màn hình. Danh sách các cột tách bởi dấu phẩy, ví dụ 1, 2, 5.
android:stretchColumns	Chỉ định chỉ số (tính từ 0) các cột cần giãn. Danh sách các cột tách bởi dấu phẩy, ví dụ 1, 2, 5. Thường dùng kết hợp với shrinkColumns.

Form login.

```

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TableRow android:gravity="center" android:paddingTop="10dp" >
        <TextView android:id="@+id/title_id"
            android:layout_width="wrap_content"
            android:layout_span="2"
            android:gravity="center"
            android:text="Login"
            android:textSize="24sp" />
    </TableRow>
    <TableRow android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginTop="10dp" >

```



```

<TextView android:id="@+id/label_username_id"
    android:layout_width="wrap_content"
    android:text="Username: "
    android:textSize="18sp" >
</TextView>
<EditText android:id="@+id/username_id"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:inputType="text" >
</EditText>
</TableRow>
<TableRow android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="10dp" >
    <TextView android:id="@+id/label_password_id"
        android:layout_width="wrap_content"
        android:text="Password: "
        android:textSize="18sp" />
    <EditText android:id="@+id/password_id"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:inputType="textPassword" >
        <requestFocus />
    </EditText>
</TableRow>
<TableRow android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:gravity="left" >
    <CheckBox android:id="@+id/show_password_id"
        android:layout_span="2"
        android:layout_weight="1"
        android:text="Show password" />
</TableRow>
<LinearLayout android:gravity="center" android:layout_marginTop="10dp">
    <Button android:id="@+id/button_login_id"
        style="?android:attr/buttonStyle"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:text="Login" >
    </Button>
    <Button android:id="@+id/button_cancel_id"
        style="?android:attr/buttonStyle"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:text="Cancel" >
    </Button>
</LinearLayout>
</TableLayout>

```

d) AbsoluteLayout

AbsoluteLayout giúp ta chỉ định chính xác vị trí (tọa độ x, y) cho các con của nó. AbsoluteLayout kém linh động và khó bảo trì hơn các loại layout khác, hiện đã lạc hậu. Thường sử dụng trong trường hợp định vị lại view khi xoay màn hình.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:layout_x	Chỉ định tọa độ x của view.
android:layout_y	Chỉ định tọa độ y của view.

```

<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="OK"
        android:layout_x="50px"
        android:layout_y="361px" />
    <Button android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_x="225px"
        android:layout_y="361px" />
</AbsoluteLayout>

```

e) FrameLayout

FrameLayout là kiểu layout đơn giản nhất, các view con bố trí với mốc là góc trên trái màn hình, vì vậy chúng có thể chồng lên nhau, một phần hoặc hoàn toàn. View được khai báo trước bị chồng bởi view được khai báo sau.

Thường dùng khi đặt các view nhỏ như dòng bản quyền, ngày giờ hiện tại (DigitalClock), ... chồng lên vị trí nào đó trên view con. Đầu tiên, Bạn thêm nhiều view con đến FrameLayout, sau đó điều chỉnh vị trí của từng view con bên trong FrameLayout bằng cách dùng thuộc tính đa trị android:layout_gravity.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:foreground	Định nghĩa một drawable là trị màu có định dạng #rgb, #argb, #rrgbv hoặc #aarrggb.
android:foregroundGravity	Định nghĩa gravity áp dụng cho drawable trên. Mặc định là fill, các trị có thể là: top, bottom, left, right, center, center_vertical, center_horizontal, v.v...
android:measureAllChildren	Xác định view con hiển thị hoặc ẩn.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView android:src="@mipmap/ic_launcher"
        android:scaleType="fitCenter"
        android:layout_height="250px"
        android:layout_width="250px"/>
    <TextView android:text="Frame Demo"
        android:textSize="30px"
        android:textStyle="bold"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:gravity="center"/>
</FrameLayout>
```

FrameLayout thường dùng trong tab selection widget. Khi nhiều phần của thông tin (ví dụ thông tin khách hàng) cần hiển thị, có thể chia chúng vào các tab của tab selection widget. Nó bao gồm:

- TabHost là container chính chứa các tab button và các tab content.
- TabWidget cài đặt dãy các tab button, chứa nhãn và icon (tùy chọn).
- FrameLayout là container chứa tab content.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TabHost android:id="@+id/tabhost"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TabWidget android:id="@android:id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        <FrameLayout android:id="@android:id/tabcontent"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:paddingTop="62px">
            <!-- layout của tab 1 lưu trong /res/layout/screen1.xml -->
            <include layout="@layout/screen1" />
            <!-- layout của tab 2 lưu trong /res/layout/screen2.xml -->
            <include layout="@layout/screen2" />
        </FrameLayout>
    </TabHost>
</FrameLayout>
```

Trong onCreate() của activity:

```
final TabHost tabs = (TabHost)findViewById(R.id.tabhost);
tabs.setup();
TabHost.TabSpec spec;

spec = tabs.newTabSpec("tag1");
spec.setContent(R.id.tab1);
spec.setIndicator("Personal");
tabs.addTab(spec);

spec = tabs.newTabSpec("tag2");
spec.setContent(R.id.tab2);
spec.setIndicator("Family");
tabs.addTab(spec);
// thiết lập tab mặc định được chọn
tabs.setCurrentTab(0); // hoặc theo tag: tabs.setCurrentTabByTag("tag1");
// lắng nghe sự kiện chọn tab bằng đối tượng OnTabChangeListener
```

```

tabs.setOnTabChangeListener(new OnTabChangeListener() {
    @Override public void onTabChanged(String tagId) {
        String text = "Im currently in: " + tagId + "\nindex: " + tabs.getCurrentTab();
        Toast.makeText(getApplicationContext(), text, Toast.LENGTH_SHORT).show();
    }
});

```

f) GridLayout

API Level 14 giới thiệu thêm GridLayout, mục đích dùng giải quyết vấn đề canh hàng và hiệu suất gấp trong các layout khác.

- Vấn đề canh hàng: để canh hàng phức tạp cần tạo các layout lồng nhau, làm cây view trở nên phức tạp.

- Vấn đề hiệu suất: cây view phức tạp, nhiều cấp sẽ làm giảm hiệu suất khi hiển thị.

GridLayout dùng một mạng lưới vô hình chia view cha thành dòng, cột và các cell. Nó hỗ trợ cả giãn (spanning) dòng lẫn giãn cột. GridLayout rất linh động, nhất là khi bạn tạo các form nhập. Designer Tool của Anroid Studio hỗ trợ rất tốt khi thiết kế có dùng GridLayout.

Ví dụ sau là Calculator tạo với GridLayout.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:columnCount="4"
        android:layout_centerHorizontal="true">
        <EditText android:layout_gravity="fill"
            android:inputType="number"
            android:layout_columnSpan="4" />

        <Button android:text="/" />
        <Button android:text="*" />
        <Button android:text="-" />
        <Button android:text="+" />

        <Button android:text="7" />
        <Button android:text="8" />
        <Button android:text="9" />
        <Button android:text="Bksp" />

        <Button android:text="4" />
        <Button android:text="5" />
        <Button android:text="6" />
        <Button android:text="C" />

        <Button android:text="1" />
        <Button android:text="2" />
        <Button android:text="3" />
        <Button android:layout_gravity="fill" android:layout_rowSpan="2" android:text="=" />

        <Button android:text="." />
        <Button android:layout_gravity="fill" android:layout_columnSpan="2" android:text="0" />
    </GridLayout>
</RelativeLayout>

```



3. Layout động

Trong trường hợp nội dung của UI là không xác định trước, Android cung cấp layout AdapterView để hỗ trợ phát triển UI động. Adapter giữ vai trò như một cầu nối giữa nguồn dữ liệu và layout AdapterView, nó lấy dữ liệu và chuyển mỗi thực thể dữ liệu vào một view để đưa vào layout AdapterView. Android cung cấp vài lớp con của Adapter để lấy dữ liệu thuận tiện từ các nguồn dữ liệu phổ biến như mảng hoặc cơ sở dữ liệu.

Adapter cũng có thể được dùng để cung cấp dữ liệu cho spinner, list view, grid view, .v.v...

a) ArrayAdapter

Bạn dùng loại adapter này khi nguồn dữ liệu của bạn là một mảng. Mặc định, ArrayAdapter tạo một view cho mỗi mục của mảng bằng cách gọi toString() cho mỗi mục và đặt nội dung lấy được vào một TextView. Giả sử bạn có một mảng các chuỗi và bạn muốn hiển thị trong một ListView (một loại layout AdapterView), khởi tạo một ArrayAdapter mới bằng cách dùng một constructor để chỉ định layout cho mỗi chuỗi và mảng chuỗi.

```

String[] fruits = {"StarApple", "Rambutan", "Guava", "Gooseberry"};
ArrayAdapter adapter = new ArrayAdapter<String>(this, R.layout.simple_list_item_1, fruits);

```

Các tham số của constructor, theo thứ tự:

- Context của ứng dụng.

- Resource ID của *layout cho một mục của list*. Layout dùng cho ví dụ trên là layout built-in tham chiếu đến một mục của list đơn giản. Các layout built-in có trong `android.R.layout`. Bạn có thể định nghĩa tùy biến layout cho một mục của list trong tập tin layout XML riêng.

Ví dụ: giả sử định nghĩa layout một mục trong `/res/layout/list_item.xml`:

```
String[] fruits = {"StarApple", "Rambutan", "Guava", "Gooseberry"};
ArrayAdapter adapter = new ArrayAdapter<String>(this, R.layout.list_item, fruits);
```

Điều này cho phép viết các adapter tùy biến làm cho các mục của list trở nên phong phú hơn.

- Mảng các chuỗi, lưu nội dung của mục sẽ hiển thị trong danh sách.

Khi bạn tạo xong `ArrayAdapter`, đơn giản gọi `setAdapter()` cho đối tượng `ListView`:

```
ListView mListView = (ListView) findViewById(R.id.listview);
mListView.setAdapter(adapter);
```

b) SimpleCursorAdapter

Khi dùng `ArrayAdapter` ta giả sử rằng toàn bộ dữ liệu đã được nạp vào bộ nhớ tại thời điểm Adapter khởi tạo. Trong đa số trường hợp, dữ liệu được lưu trữ trong cơ sở dữ liệu, cách tiếp cận này không thích hợp. Android cung cấp `SimpleCursorAdapter` nếu nguồn dữ liệu của bạn là tập kết quả `Cursor` lấy từ cơ sở dữ liệu.

Khi dùng `SimpleCursorAdapter`, bạn phải chỉ định một layout được dùng cho mỗi dòng của `Cursor` và cột trong `Cursor` sẽ được chèn vào với các View của layout đó.

Ví dụ, nếu bạn muốn tạo một danh sách liên lạc, mỗi mục có tên và số điện thoại. Bạn thực hiện một query trả về một `Cursor` mà một dòng chứa thông tin liên lạc cho một cá nhân và các cột của dòng tương ứng với tên và số điện thoại. Sau đó:

- Tạo một mảng chuỗi chỉ định cột nguồn nào trong `Cursor` bạn sẽ lấy tên.

- Tạo một mảng số nguyên chỉ định id của View đích trong layout bạn muốn đặt dữ liệu.

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

Khởi tạo `SimpleCursorAdapter`, truyền cho constructor các tham số: layout được dùng cho mỗi kết quả, `Cursor` chứa các kết quả, hai mảng nguồn và đích trên.

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.list_item,
                                                    cursor, fromColumns, toViews, 0);
ListView mListView = (ListView) findViewById(R.id.listview);
mListView.setAdapter(adapter);
```

`SimpleCursorAdapter` sẽ tạo một view cho mỗi dòng trong `Cursor` bằng cách dùng layout được cung cấp, chèn từng mục của `fromColumns` vào View `toViews` tương ứng.

c) Adapter tùy biến

Layout chỉ định trong constructor của adapter, `android.R.layout.simple_list_item_1`, là layout định nghĩa trước trong tài nguyên được cung cấp bởi Android SDK, nó chỉ có một `TextView` dùng như root element. Vì vậy, một mục (item) của `ListView` chỉ hiển thị như một chuỗi. Nếu bạn muốn tùy biến một mục để có nhiều chi tiết hơn như thêm chuỗi với các định dạng khác nhau, thêm hình ảnh, checkbox, v.v... cần thực hiện như sau:

- Tạo một tập tin layout XML mới, định nghĩa view cho mục tùy biến. Ví dụ `/res/layout/list_item.xml`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <CheckBox android:id="@+id/enrolled"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_alignParentRight="true"
        android:focusable="false" />
    <TextView android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@id/enrolled"
        android:textStyle="bold" />
    <TextView android:id="@+id/date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/titleText"
        android:layout_toLeftOf="@id/enrolled" />
</RelativeLayout>
```

- Tạo một adapter tùy biến là lớp con của `BaseAdapter`, thường thừa kế từ `ArrayAdapter<T>`. Lớp này thường được cài đặt như lớp thành viên của `ListActivity` hoặc `FragmentActivity`. Các phương thức cần được cài đặt là:

Phương thức	Mô tả
getCount()	Trả về số item có trong tập dữ liệu.
getItem(int)	Trả về đối tượng item tại vị trí cho trước.
getItemId(int)	Trả về định danh của item (id của dòng) tại vị trí cho trước.
getView(int, View, ViewGroup)	Trả về đối tượng View có thể thêm vào layout AdapterView để hiển thị một item.

Phương thức getView() nhận tham số thứ hai là một thực thể View cũ để dùng lại. Nếu thực thể View này là null, nó được tạo bằng code hoặc đơn giản "inflate" từ tập tin tài nguyên layout XML. Đối tượng LayoutInflater lấy từ dịch vụ Context.LAYOUT_INFLATER_SERVICE:

```
LayoutInflater inflater =
    (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
convertView = inflater.inflate(R.layout.list_item, parent, false);
```

hoặc từ Activity.getLayoutInflater() nếu adapter là lớp nội của activity. Khi View dùng lại chuyển đổi thành layout mới, nạp dữ liệu của item tại vị trí chỉ định (tham số thứ nhất) vào view.

```
public class CustomAdapter extends ArrayAdapter<Student> {
    public CustomAdapter(ArrayList<Student> students) {
        super(getActivity(), 0, students);
    }

    @Override public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = getActivity().getLayoutInflater().inflate(R.layout.list_item, parent, false);
        }
        // Cấu hình view cho Student tại vị trí position
        Student student = getItem(position);
        TextView title = (TextView) convertView.findViewById(R.id.title);
        title.setText(student.getName());
        TextView date = (TextView) convertView.findViewById(R.id.date);
        date.setText(student.getBirthDate().toString());
        CheckBox enrolled = (CheckBox) convertView.findViewById(R.id.enrolled);
        enrolled.setChecked(student.isEnrolled());
        return convertView;
    }
}
```

Xem thêm phần design pattern View Holder.

- Sử dụng CustomAdapter trong ListActivity hoặc ListFragment:

Trong onCreate(), thiết lập adapter cho ListView bằng adapter tùy biến trên:

```
CustomAdapter adapter = new CustomAdapter(students);
setListAdapter(adapter);
```

Trong onItemClick(), lấy đối tượng tương ứng cho một mục được chọn:

```
Student student = ((CustomAdapter)getListAdapter()).getItem(position);
```

d) Cập nhật adapter

Đôi khi, bạn cần cập nhật dữ liệu của adapter, thực hiện như sau:

- Với ArrayAdapter, dùng các phương thức hỗ trợ add(), insert(), remove() và clear() của adapter để thay đổi tập dữ liệu. Nếu ứng dụng kích hoạt báo thay đổi bằng phương thức setNotifyOnChange(), adapter sẽ báo cho AdapterView biết dữ liệu đã thay đổi do các phương thức hỗ trợ.

```
arrayAdapter.setNotifyOnChange(true);
arrayAdapter.add("Item 5");
arrayAdapter.remove("Item 4");
```

- Trường hợp dùng loại adapter khác, hoặc chỉ cập nhật được list mà adapter tham chiếu, mỗi lần cập nhật bạn dùng phương thức notifyDataSetChanged() của adapter để báo cho AdapterView biết adapter đã thay đổi.

Vì bạn chỉ thay đổi UI, nên cũng phải tiến hành cập nhật đồng bộ cho nguồn dữ liệu của ứng dụng.

```
public class CustomAdapter extends ArrayAdapter<Student> {
    @Override public View getView(int position, View convertView, ViewGroup parent) {
        convertView = getActivity().getLayoutInflater().inflate(R.layout.list_item, parent, false);
        ImageView delete = (ImageView) convertView.findViewById(R.id.delete);
        delete.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                new AlertDialog.Builder(context)
                    .setMessage("Do you want to remove?")
                    .setCancelable(false)
                    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            itemList.remove(position);
                            notifyDataSetChanged();
                        }
                    })
                    .setNegativeButton("No", new DialogInterface.OnClickListener() {
```



```

        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    }).create().show();
});
}
return convertView;
}

```

Trong ví dụ trên, bạn cập nhật adapter trực tiếp từ ListView. Nhiều tình huống, list bên trong adapter được cập nhật từ view khác, lúc này bạn cần gọi `notifyDataSetChanged()` trong phương thức callback `onResume()` của activity/fragment chứa adapter đó. Ví dụ:

```

@Override public void onResume() {
    super.onResume();
    ((CustomAdapter) getListAdapter()).notifyDataSetChanged();
}

```

Chú ý là `notifyDataSetChanged()` chỉ được gọi trong UI thread. Nếu không, dùng `runOnUiThread()` của activity để gọi:

```

final ArrayAdapter adapter = ((ArrayAdapter) getListAdapter());
runOnUiThread(new Runnable() {
    public void run() {
        adapter.notifyDataSetChanged();
    }
});

```

4. Layout AdapterView

Android cung cấp vài loại layout AdapterView cho các trường hợp sử dụng khác nhau. Tất cả các lớp AdapterView này đều dùng một thực thể Adapter để đổ đầy dữ liệu vào các View của từng mục trong AdapterView.

a) ListView

ListView là một layout AdapterView, hiển thị các mục trong một danh sách cuộn được theo chiều dọc. Danh sách các mục được chèn tự động vào list bằng cách dùng Adapter để lấy nội dung từ nguồn như mảng hoặc cơ sở dữ liệu.

Có vài cách dùng ListView:

- Dùng ListView. Khai báo tĩnh ListView với id, ví dụ: `@+id/list_view` trong tập tin layout XML của activity:

```

<ListView android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

rồi cung cấp adapter cho nó:

```

public class MainActivity extends Activity {
    ListView myListView;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        myListView = (ListView) findViewById(R.id.list_view);
        final String[] items = { "Orange", "Rambutan", "Guava", "Starapple" };
        ArrayAdapter<String> ad = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items);
        myListView.setAdapter(ad);
        // xử lý sự kiện tap một mục trong list
        myListView.setOnItemClickListener(new OnItemClickListener() {
            @Override public void onItemClick(AdapterView<?> av, View v, int position, long id) {
                String text = "position: " + position + "\ndata: " + items[position];
                Toast.makeText(getApplicationContext(), text, Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

- Dùng ListActivity. Đây là lớp con của Activity, bạn viết lại phương thức `onListItemClick()` xử lý sự kiện tap lên một mục của list. Nếu ứng dụng sử dụng fragment, dùng `ListFragment` thay thế.

Nếu ListView chiếm cả activity, không cần khai báo tập tin layout XML. Nếu activity có layout riêng, ListView được khai báo với thuộc tính `android:id="@android:id/list`.

```

public class MainActivity extends ListActivity {
    final String[] items = { "Orange", "Rambutan", "Guava", "Starapple" };

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items));
    }

    @Override protected void onListItemClick(ListView l, View v, int position, long id) {

```

```

super.onListItemClick(l, v, position, id);
String text = "position: " + position + "\ndata: " + items[position];
Toast.makeText(getApplicationContext(), text, Toast.LENGTH_SHORT).show();
}
}

```

Ví dụ sau dùng ListView để giả lập một menu, điều hướng đến các activity bằng cách gửi intent tương ứng:

```

public class MenuActivity extends ListActivity {
    String[] classes = { "Task 1", "Task 2" };

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(MenuActivity.this,
            android.R.layout.simple_list_item_1, classes));
    }

    @Override protected void onListItemClick(ListView l, View v, int position, long id) {
        String[] classnames = {"Task1Activity", "Task2Activity" };
        super.onListItemClick(l, v, position, id);
        try {
            Class<?> c = Class.forName("com.twe.examples.menudemo." + classnames[position]);
            Intent intent = new Intent(MenuActivity.this, c);
            startActivity(intent);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

```

Các thuộc tính của ListView:

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:divider	Drawable hoặc màu được vẽ giữa các mục của danh sách thành đường phân cách.
android:dividerHeight	Chỉ định chiều cao của divider. Có thể tính bằng px, dp, sp, in, hoặc mm.
android:entries	Chỉ định tham chiếu đến một nguồn tài nguyên mảng chứa dữ liệu sẽ của các mục.
android:footerDividersEnabled	Nếu false, ListView không vẽ divider trước mỗi chân view. Mặc định là true.
android:headerDividersEnabled	Nếu false, ListView không vẽ divider sau mỗi đầu view. Mặc định là true.

b) GridView

GridView là một AdapterView, trình bày các mục trong một lưới hai chiều (row và column) cuộn được. Các mục của lưới không cần xác định trước, chúng có thể được chèn tự động vào layout bằng cách dùng ListAdapter.

Các thuộc tính của GridView

Thuộc tính	Mô tả
android:id	Định danh duy nhất của layout.
android:columnWidth	Chỉ định chiều rộng cố định của từng cột. Có thể tính bằng px, dp, sp, in, hoặc mm.
android:gravity	Chỉ định gravity của nội dung từng cell. Có thể là top, bottom, left, right, center, center_vertical, center_horizontal, v.v...
android:horizontalSpacing	Định nghĩa khoảng cách ngang mặc định giữa các cột. Có thể tính bằng px, dp, sp, in, hoặc mm.
android:numColumns	Số cột được hiển thị. Có thể là số nguyên hoặc auto_fit.
android:stretchMode	Định nghĩa cách các cột giãn ra để lấp đầy không gian trống, nếu có thể. none: bất hoạt chế độ stretch. spacingWidth: khoảng cách giữa các cột sẽ giãn. columnWidth: từng cột sẽ giãn bằng nhau. spacingWidthUniform: khoảng cách giữa các cột sẽ giãn đồng dạng.
android:verticalSpacing	Định nghĩa khoảng cách dọc mặc định giữa các hàng. Có thể tính bằng px, dp, sp, in, hoặc mm.

Ví dụ tạo một album chứa các thumbnail, dùng GridView trong /res/layout/activity_main.xml:

```

<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/grid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center" />

```

Chuẩn bị adapter tùy biến:

```

public class ImageAdapter extends BaseAdapter {

```

```

private Context context;
public ImageAdapter(Context context) {
    this.context = context;
}

public int getCount() {
    return mThumbIds.length;
}

public Object getItem(int position) {
    return null;
}

public long getItemId(int position) {
    return 0;
}

// tạo một ImageView cho mỗi item được tham chiếu bởi Adapter
public View getView(int position, View convertView, ViewGroup parent) {
    ImageView mImageView;
    if (convertView == null) {
        mImageView = new ImageView(mContext);
        mImageView.setLayoutParams(new GridView.LayoutParams(85, 85));
        mImageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        mImageView.setPadding(8, 8, 8, 8);
    } else {
        mImageView = (ImageView) convertView;
    }
    mImageView.setImageResource(mThumbIds[position]);
    return mImageView;
}

// mảng các ảnh
public Integer[] mThumbIds = {
    R.drawable.sample_0, R.drawable.sample_1, R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5, R.drawable.sample_6, R.drawable.sample_7 };
}

```

Activity với GridView:

```

public class MainActivity extends Activity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GridView mGridView = (GridView) findViewById(R.id.grid);
        mGridView.setAdapter(new ImageAdapter(this));
        mGridView.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
                // gửi intent (kèm theo id của ảnh) để mở ảnh trong activity khác
                Intent intent = new Intent(getApplicationContext(), SingleViewActivity.class);
                intent.putExtra("id", position);
                startActivity(intent);
            }
        });
    }
}

```

5. Spinner

Spinner là lớp con của AdapterView. Spinner giống như một danh sách thả xuống, chứa các trị cho phép người dùng lựa chọn. Spinner có chức năng giống ListView nhưng chiếm không gian nhỏ hơn.

Thêm Spinner đến tài nguyên layout XML.

```

<Spinner android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```

Một số thuộc tính đáng chú ý của Spinner:

Thuộc tính	Mô tả
android:spinnerMode	Chế độ hiển thị của Spinner: dialog: Spinner hiển thị như một cửa sổ. dropdown: Spinner hiển thị như một widget.
android:dropDownHorizontalOffset	Offset theo chiều ngang của drop-down tính bằng pixel.
android:dropDownVerticalOffset	Offset theo chiều dọc của drop-down tính bằng pixel.

android:gravity	Gravity của nội dung.
android:dropDownSelector	Selector tham chiếu đến một tài nguyên hoặc một tệp màu.
android:dropDownWidth	Chiều rộng của drop-down.
android:popupBackground	Drawable dùng như nền của drop-down.
android:prompt	Dấu nhắc hiển thị khi Spinner trong chế độ dialog.

Cũng như với ListView, bạn cung cấp adapter để liên kết đến các mục trong Spinner bằng phương thức `setAdapter()`. Sau đó, gán đối tượng lắng nghe sự kiện chọn mục của Spinner, đối tượng `OnItemSelectedListener`, bằng phương thức `setOnItemSelectedListener()`.

```
String[] fruits = { "Orange", "Rambutan", "Gooseberry" }
Spinner spinner = (Spinner) findViewById(R.id.spinner);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item, fruits);
spinner.setAdapter(adapter);
spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        int index = spinner.getSelectedPosition();
        Toast.makeText(getApplicationContext(), "Favorite fruit: " + fruits[index], Toast.LENGTH_SHORT).show();
    }
    public void onNothingSelected(AdapterView<?> parent) { }
});
```

6. ViewPager

Để xây dựng giao diện có thể lướt (swipe) qua lại giữa trang hiện hành và các trang hai bên, activity phải tạo và quản lý một ViewPager. Vì ViewPager sẽ là một fragment container, nó cần phải có resource ID.

Một AdapterView cần một Adapter để cung cấp đối tượng dữ liệu cho từng mục, một ViewPager cũng cần một PagerAdapter để cung cấp fragment cho từng mục. Nói chung, thường dùng `FragmentStatePagerAdapter`, lớp con của `PagerAdapter`. Bạn cần viết hai phương thức cho lớp này, `getCount()` và `getItem()`. Khi phương thức `getItem()` được gọi cho một vị trí của adapter, nó sẽ trả về một fragment được cấu hình với đối tượng lưu tại vị trí đó.

Khi sử dụng ViewPager, các sự kiện như chọn trang, cuộn trang được listener `ViewPager.OnPageChangeListener` nhận biết, bạn cần cài đặt cho các phương thức callback của lớp này để xử lý sự kiện nhận được.

```
public class PagerActivity extends FragmentActivity {
    private ViewPager pager;
    private ArrayList<Item> items;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        pager = new ViewPager(this);
        pager.setId(R.id.pager);
        setContentView(pager);
        // nạp dữ liệu cho items
        FragmentManager fm = getFragmentManager();
        pager.setAdapter(new FragmentStatePagerAdapter(fm) {
            @Override public int getCount() {
                return items.size();
            }
            @Override public Fragment getItem(int position) {
                Item item = items.get(position);
                return ItemFragment.newInstance(item.getId());
            }
        });
        pager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {
            public void onPageScrollStateChanged(int state) { }
            public void onPageScrolled(int position, float posOffset, int posOffsetPixels) { }
            public void onPageSelected(int position) {
                Item item = items.get(position);
                setTitle(item.getName());
            }
        });
        pager.setCurrentItem(0);
    }
}
```

7. ViewStub

Đôi khi không nhất thiết phải hiển thị toàn bộ layout của bạn, một số thành phần sẽ được nạp theo yêu cầu (on-demand) trong thời gian chạy. Khi đó, bạn tạo layout riêng cho thành phần đó rồi đưa vào ViewStub.

ViewStub là một view control nhẹ, không có kích thước và không vẽ ra layout. ViewStub có thuộc tính `android:layout`, báo layout sẽ được đưa vào ViewStub. Khi bạn cần nạp layout vào ViewStub, gọi `setVisibility(View.VISIBLE)` hoặc gọi `inflate()`.

- Layout có dùng ViewStub:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <ViewStub android:id="@+id/viewstub"
        android:layout="@layout/subtree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

- Chuẩn bị một layout riêng, sẽ đưa vào sau, trong /res/layout/subtree.xml.
- Khi cần nạp layout riêng (ViewStub), thực hiện:

```
((ViewStub) findViewById(R.id.viewstub)).setVisibility(View.VISIBLE);
```

Bạn cũng có thể thêm ViewStub vào cây view bằng code:

```
LinearLayout layout = (LinearLayout) findViewById(R.id.layout);
ViewStub stub = new ViewStub(MainActivity.this);
LinearLayout.LayoutParams params = new LinearLayout.LayoutParams
    ((int) LayoutParams.WRAP_CONTENT, (int) LayoutParams.WRAP_CONTENT);
stub.setLayoutParams(params);
layout.addView(stub);
stub.setLayoutResource(R.layout.subtree);
View inflated = stub.inflate();
```

View

Bạn vừa biết các layout dùng bố trí các view. Phần này giới thiệu chi tiết về các view đó.

Một View là một đối tượng được vẽ trên màn hình và người dùng có thể tương tác với nó. Một ViewGroup là đối tượng chứa các đối tượng View và ViewGroup khác.

Một đối tượng view có thể có một định danh (id) duy nhất gán cho nó định danh một View trong một cây. Cú pháp của định danh, dùng bên trong element XML là:

```
android:id="@+id/text_id"
```

Để tạo một view bạn định nghĩa chúng trong tập tin layout và gán nó một định danh duy nhất như sau:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
</LinearLayout>
```

Sau đó, tạo một thực thể của view này và lấy nó từ layout như sau:

```
TextView mTextView = (TextView) findViewById(R.id.text_id);
```

Android cung cấp một số view cho phép người dùng tạo nên UI cho ứng dụng.

View	Mô tả
TextView	Dùng để hiển thị văn bản cho người dùng.
EditText	Lớp con của TextView, cung cấp nhiều khả năng soạn thảo phong phú.
AutoCompleteTextView	Tương tự EditText, có thể đưa ra một danh sách gợi ý hoàn chỉnh nhập tự động khi nhập.
Button	Nút, có thể nhấn, tap bởi người dùng để thực hiện một hành động.
ImageButton	Nút, có hình ảnh. Thừa kế từ ImageView.
Checkbox	Hộp kiểm, thiết lập on/off bởi người dùng. Các tùy chọn được chọn không loại trừ nhau.
ToggleButton	Nút giữ nguyên trạng thái khi nhấn, nhấn lần nữa để chuyển qua trạng thái kia.
RadioButton	Người dùng chọn/bỏ chọn. Tùy chọn được chọn loại trừ các tùy chọn khác trong nhóm.
RadioGroup	Dùng nhóm một hay nhiều RadioButton với nhau.
ProgressBar	Cung cấp một cái nhìn trực quan về tiến độ của một số nhiệm vụ đang thực hiện.
Spinner	Danh sách thả xuống, cho phép người dùng chọn một trị từ danh sách.
TimePicker	Khi kích hoạt, cho phép người dùng chọn thời gian trong ngày. Chế độ 24-hour hoặc AM/PM.
DatePicker	Khi kích hoạt, cho phép người dùng chọn ngày tháng năm.

1. Output control

a) TextView

TextView hiển thị văn bản của người dùng và tùy chọn cho phép người dùng biên soạn văn bản đó. TextView là một đơn thể biên soạn văn bản, tuy nhiên lớp cơ bản được cấu hình không cho phép biên soạn.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của control.
android:capitalize	Nếu thiết lập, TextView có phương thức nhập, tự động viết hoa ký tự đầu (capitalize) văn bản nhập vào.

	0 – Không tự động capitalize 1 – Capitalize từ đầu của mỗi câu 2 – Capitalize từng từ của câu 3 – Capitalize mọi ký tự
android:autoLink	Nếu thiết lập, tự động chuyển URL hoặc địa chỉ email trong văn bản thành link.
android:cursorVisible	Hiển thị con trỏ văn bản (mặc định) hoặc ẩn. Mặc định là false.
android:editable	Nếu thiết lập true, cho phép TextView có phương thức nhập.
android:ellipsize	Nếu thiết lập: start, middle, end hoặc marquee; các từ vượt quá width sẽ thay bằng dấu (...). Thiết lập marquee, văn bản tự động trượt từ phải sang trái.
android:fontFamily	Tên font (dạng chuỗi) dùng cho văn bản.
android:gravity	Chỉ định cách căn chỉnh văn bản theo trục x hoặc y của view, khi văn bản nhỏ hơn view.
android:hint	Văn bản hướng dẫn hiển thị khi TextView trống.
android:inputType	Kiểu của dữ liệu nhập. Phone, Date, Time, Number, Password, .vv...
android:maxLength	Chiều cao tối đa của TextView, tính bằng pixel.
android:maxLength	Chiều rộng tối đa của TextView, tính bằng pixel.
android:minHeight	Chiều cao tối thiểu của TextView, tính bằng pixel.
android:minHeight	Chiều rộng tối thiểu của TextView, tính bằng pixel.
android:password	Trị true hoặc false. Cho phép hiển thị các ký tự nhập thành ký tự chấm (dot).
android:phoneNumber	Trị true hoặc false. Chỉ định phương thức nhập của TextView là phone number.
android:singleLine	Nếu thiết lập, văn bản sẽ không giãn ra thành nhiều dòng.
android:text	Văn bản hiển thị.
android:textAllCaps	Trị true hoặc false. Hiển thị văn bản chữ hoa.
android:textColor	Màu văn bản. Trị màu theo định dạng #rgb, #argb, #rrggbb hoặc #aarrggbb.
android:textColorHighlight	Màu của văn bản được chọn.
android:textColorHint	Màu văn bản hướng dẫn. Trị màu theo định dạng #rgb, #argb, #rrggbb hoặc #aarrggbb.
android:textIsSelectable	Trị true hoặc false. Chỉ định nội dung không biên soạn được có thể được chọn.
android:textSize	Kích thước văn bản, đơn vị đề nghị là "sp" (scaled-pixel).
android:textStyle	Kiểu văn bản (bold, italic, bolditalic). Danh sách phân cách bởi dấu ' '. 0 – normal 1 – bold 2 – italic
android:typeface	Kiểu font (normal, sans, serif, monospace) của văn bản. Danh sách phân cách bởi dấu ' '.

b) ImageView

Hiển thị ảnh cho người dùng.

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.
android:adjustViewBounds	Thiết lập true nếu bạn muốn ImageView hiệu chỉnh các biên để giữ tỷ lệ của các drawable của nó.
android:baseline	Offset của đường nền bên trong view này.
android:baselineAlignBottom	Nếu true, ImageView sẽ căn chỉnh đường nền với cạnh dưới của nó.
android:cropToPadding	Nếu true, ảnh sẽ bị xén để vừa với bên trong padding của nó.
android:scaleType	Điều khiển cách co giãn ảnh. Trị: matrix, fitXY, fitStart, fitCenter, fitEnd, ...
android:src	Thiết lập một drawable dùng làm nội dung của ImageView. Resource ID hoặc color.
android:background	Drawable được dùng như hình nền.

c) ProgressBar

ProgressBar cho phép ứng dụng báo cho người dùng một cách trực quan về tiến trình đang thực hiện. Có hai dạng:

- dạng thanh (bar) nếu thời gian thực hiện tiến trình xác định được.
- dạng vòng (ring) nếu thời gian thực hiện tiến trình không xác định được.

Thuộc tính	Mô tả
android:indeterminate	Nếu thiết lập, kích hoạt chế độ vòng (ring).
android:max	Trị tối đa được thiết lập cho progress bar.
android:progress	Trị cho mỗi bước tăng.
style	Kiểu progress bar, Android cung cấp: <div> <div>@android:style/Widget.ProgressBar.Horizontal</div> <div>thanh ngang</div> </div> <div> <div>@android:style/Widget.ProgressBar.Small</div> <div>vòng</div> </div> <div> <div>@android:style/Widget.ProgressBar.Large</div> <div>vòng lớn</div> </div> <div> <div>@android:style/Widget.ProgressBar.Inverse</div> <div>vòng xoay ngược</div> </div>

d) Space

View vô hình dùng chèn các khoảng trống (ngang hoặc dọc) giữa các đơn thể.

```
<Space android:layout_width="wrap_content"
    android:layout_height="20dp" />
```

2. Input control

a) EditText

EditText là lớp con của TextView, thừa kế từ android.widget.TextView, cho phép người dùng soạn thảo nội dung và cung cấp nhiều khả năng soạn thảo phong phú.

Thuộc tính	Mô tả
android:id	Định danh duy nhất của control.
android:autoText	Chỉ định EditText có phương thức nhập văn bản, tự động sửa lỗi chính tả.
android:cursorVisible	Nếu thiết lập, hiển thị con trỏ soạn thảo.
android:drawableBottom	Drawable được vẽ bên dưới văn bản.
android:drawableRight	Drawable được vẽ bên phải văn bản.
android:digits	Cấu hình trường chỉ chấp nhận số.
android:enabled	Chỉ định control được kích hoạt.
android:singleLine	Nếu thiết lập, văn bản sẽ không giãn ra thành nhiều dòng.
android:password	Trị true hoặc false. Chỉ định nội dung hiển thị hoặc thay bằng các dấu chấm.
android:text	Văn bản hiển thị.
android:background	Drawable được dùng làm nền.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung trong view.
android:onClick	Tên phương thức context của View sẽ triệu gọi khi click.
android:visibility	Cho phép view hiển thị.

b) AutoComplexTextView

AutoCompleteTextView là một view tương tự EditText, ngoại trừ việc nó hiển thị một danh sách gợi ý hoàn tất tự động khi người dùng nhập văn bản. Danh sách gợi ý này được hiển thị trong một list thả xuống, người dùng có thể chọn một mục trong danh sách này để thay thế (hoặc hoàn tất) nội dung đang nhập.

Thuộc tính	Mô tả
android:completionHint	Định nghĩa hướng dẫn được hiển thị trong danh sách thả xuống.
android:completionHintView	Định nghĩa view của hướng dẫn được hiển thị trong danh sách thả xuống.
android:completionThreshold	Định nghĩa số ký tự ít nhất người dùng phải nhập trước khi gợi ý hoàn tất hiển thị.
android:dropDownAnchor	Định nghĩa view làm mốc cho danh sách thả xuống.
android:dropDownHeight	Chỉ định chiều cao của danh sách thả xuống.
android:dropDownHorizontalOffset	Số pixel danh sách thả xuống dùng làm offset theo chiều ngang.
android:dropDownSelector	Bộ chọn trong danh sách thả xuống
android:dropDownVerticalOffset	Số pixel danh sách thả xuống dùng làm offset theo chiều dọc.
android:dropDownWidth	Chỉ định chiều ngang của danh sách thả xuống.
android:popupBackground	Dùng thiết lập nền.

c) Button

Button là một nút có thể nhấn hoặc tap bởi người dùng, để thực hiện một hành động.

Thuộc tính thừa kế lớp android.widget.TextView

Thuộc tính	Mô tả
android:autoText	Chỉ định EditText có phương thức nhập văn bản, tự động sửa lỗi chính tả.
android:drawableBottom	Drawable được vẽ bên dưới văn bản.
android:drawableRight	Drawable được vẽ bên phải văn bản.
android:editable	Nếu thiết lập true, cho phép EditText có một phương thức nhập.
android:text	Văn bản hiển thị.

Thuộc tính thừa kế lớp android.view.View

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.
android:background	Drawable được dùng để làm nền.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung của view.
android:onClick	Tên của phương thức được context của view triệu gọi
android:visibility	Điều khiển hiển thị ban đầu của view.

d) ImageButton

Không là lớp con của Button, mà là lớp con của ImageView. ImageButton hiển thị nút với một hình ảnh (thay vì văn bản), có thể nhấn hoặc tap bởi người dùng.

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.
android:onClick	Tên của phương thức được context của view triệu gọi
android:src	Thiết lập một drawable dùng làm nội dung của ImageView. Resource ID hoặc color.
android:visibility	Điều khiển hiển thị ban đầu của view.

e) ToggleButton

ToggleButton là nút có trạng thái checked/unchecked bật qua lại giữa hai trạng thái.

Thuộc tính	Mô tả
android:disabledAlpha	Alpha (kênh trong suốt) được áp dụng khi nút bất hoạt.
android:textOff	Văn bản cho nút khi unchecked.
android:textOn	Văn bản cho nút khi checked.

android:editable	Nếu thiết lập true, cho phép TextView có một phương thức nhập.
android:text	Văn bản hiển thị.

Thuộc tính thừa kế lớp android.widget.TextView

Thuộc tính	Mô tả
android:autoText	Chỉ định TextView có phương thức nhập văn bản, tự động sửa lỗi chính tả.
android:drawableBottom	Drawable được vẽ bên dưới văn bản.
android:drawableRight	Drawable được vẽ bên phải văn bản.
android:editable	Nếu thiết lập true, cho phép TextView có một phương thức nhập.
android:text	Văn bản hiển thị.

Thuộc tính thừa kế lớp android.view.View

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.
android:background	Drawable được dùng để làm nền.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung của view.
android:onClick	Tên của phương thức được context của view triệu gọi
android:visibility	Điều khiển hiển thị ban đầu của view.

Một control khá giống ToogleButton là Switch:

<pre><Switch android:id="@+id/toggle" android:layout_width="wrap_content" android:layout_height="wrap_content" android:textOff="Off" android:textOn="On" android:checked="true" android:showText="true" /></pre>
--

f) CheckBox

CheckBox sử dụng khi người dùng chọn một số tùy chọn trong một nhóm, các tùy chọn không loại trừ nhau.

Thuộc tính thừa kế lớp android.widget.TextView

Thuộc tính	Mô tả
android:autoText	Chỉ định TextView có phương thức nhập văn bản, tự động sửa lỗi chính tả.
android:drawableBottom	Drawable được vẽ bên dưới văn bản.
android:drawableRight	Drawable được vẽ bên phải văn bản.
android:editable	Nếu thiết lập true, cho phép TextView có một phương thức nhập.
android:text	Văn bản hiển thị.

Thuộc tính thừa kế lớp android.view.View

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.
android:background	Drawable được dùng để làm nền.
android:checked	Thiết lập true hoặc false, cho thấy đang check hoặc không.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung của view.
android:onClick	Tên của phương thức được context của view triệu gọi
android:visibility	Điều khiển hiển thị ban đầu của view.

Ứng dụng có thể đăng ký một listener để lắng nghe sự kiện thay đổi trạng thái check.

<pre>checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() { @Override public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) { if (isChecked) { // trạng thái ON } else { // trạng thái OFF } } });</pre>

h) RadioButton và RadioGroup

RadioButton có hai trạng thái, được chọn hoặc không. RadioButton cho phép người dùng chỉ chọn một tùy chọn trong một tập các tùy chọn.

Thuộc tính thừa kế lớp android.widget.TextView

Thuộc tính	Mô tả
android:autoText	Chỉ định TextView có phương thức nhập văn bản, tự động sửa lỗi chính tả.
android:drawableBottom	Drawable được vẽ bên dưới văn bản.
android:drawableRight	Drawable được vẽ bên phải văn bản.
android:editable	Nếu thiết lập true, cho phép TextView có một phương thức nhập.
android:text	Văn bản hiển thị.

Thuộc tính thừa kế lớp android.view.View

Thuộc tính	Mô tả
android:id	Cung cấp một định danh cho view này.

android:background	Drawable được dùng để làm nền.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung của view.
android:onClick	Tên của phương thức được context của view triệu gọi
android:visibility	Điều khiển hiển thị ban đầu của view.

Lớp `RadioGroup` dùng cho một tập các `RadioButton`. Nếu bạn chọn một radio button trong một radio group, bất kỳ radio button trong cùng nhóm được chọn trước đây sẽ tự động bỏ chọn.

Thuộc tính	Mô tả
android:checkedButton	ID của radio button con trong nhóm sẽ được chọn mặc định.

Thuộc tính thừa kế lớp `android.view.View`

Thuộc tính	Mô tả
android:background	Drawable được dùng để làm nền.
android:contentDescription	Định nghĩa văn bản mô tả nhanh nội dung của view.
android:id	Cung cấp một định danh cho view này.
android:onClick	Tên của phương thức được context của view triệu gọi
android:visibility	Điều khiển hiển thị ban đầu của view.

Ứng dụng có thể đăng ký một listener để lắng nghe sự kiện thay đổi trạng thái check của `RadioGroup`:

```
radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override public void onCheckedChanged(RadioGroup group, int checkedId) {
        switch (checkedId) {
            case R.id.email: break;
            case R.id.phone: break;
            case R.id.both: break;
            case R.id.none:
        }
    }
});
```

Ví dụ dùng `RadioButton` và `RadioGroup`:

```
public class AndDemoUI extends Activity {
    CheckBox chkCream, chkSugar;
    Button btnPay;
    RadioGroup radCoffeeType, radDecaf, radEspresso, radColombian;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        chkCream = (CheckBox) findViewById(R.id.chkCream);
        chkSugar = (CheckBox) findViewById(R.id.chkSugar);

        radCoffeeType = (RadioGroup) findViewById(R.id.radGroupCoffeeType);
        radDecaf = (RadioButton) findViewById(R.id.radDecaf);
        radEspresso = (RadioButton) findViewById(R.id.radEspresso);
        radColombian = (RadioButton) findViewById(R.id.radColombian);

        btnPay = (Button) findViewById(R.id.btnPay);
        btnPay.setOnClickListener(new OnClickListener() {
            @Override public void onClick(View v) {
                String msg = "Coffee ";
                if (chkCream.isChecked()) msg += " & cream";
                if (chkSugar.isChecked()) msg += " & Sugar";
                // lấy ID của radio button được chọn
                int radioId = radCoffeeType.getCheckedRadioButtonId();
                if (radColombian.getId() == radioId) msg = "Colombian " + msg;
                // bạn có thể dùng isChecked() với từng radio button
                if (radEspresso.isChecked()) msg = "Espresso " + msg;
                Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

3. Input Method Framework (IMF)

Android 1.5 đưa ra khái niệm IMF. Ý tưởng là để IMF làm trung gian tương tác giữa ứng dụng và phương pháp nhập của người dùng. IMF nhằm đáp ứng sự phát triển nhanh của các thiết bị phần cứng/phần mềm, các phương pháp nhập mới xuất hiện trong các ứng dụng của người dùng, như bàn phím ảo và thật, nhận diện tiếng nói, nhận diện chữ viết tay.

IMF nhận biết bàn phím cứng có sẵn và trạng thái hiện tại của nó. Nếu không có bàn phím cứng, bộ soạn thảo phương pháp nhập (IME – Input Method Editor) sẽ được cung cấp khi người nhập vào `EditText` đang kích hoạt.

`EditText` có các thuộc tính báo cho Android biết kiểu dữ liệu nhập dự kiến:

- Trong tập tin layout XML: `android:inputType="text|textCapWords"`
- Trong code: `editText.setRawInputType(android.text.InputType.TYPE_CLASS_PHONE);`

Do biết được kiểu dữ liệu nhập dự kiến Android hỗ trợ người dùng nhập liệu một cách hợp lý, ví dụ chỉ hiển thị bàn phím ảo với các phím thích hợp.

Hằng	Trị	Mô tả
none	0x00000000	Không có kiểu nội dung. Dùng bất hoạt EditText.
text	0x00000001	Văn bản.
textCapCharacters	0x00001001	Viết hoa các ký tự. Có thể kết hợp với text và các biến thể của nó.
textCapWords	0x00002001	Viết hoa ký tự đầu mỗi từ. Có thể kết hợp với text và các biến thể của nó.
textCapSentences	0x00004001	Viết hoa ký tự đầu mỗi câu. Có thể kết hợp với text và các biến thể của nó.
textAutoCorrect	0x00008001	Tự động sửa lỗi văn bản nhập. Có thể kết hợp với text và các biến thể của nó.
textAutoComplete	0x00010001	Hoàn thành tự động văn bản nhập. Có thể kết hợp với text và các biến thể của nó.
textMultiLine	0x00020001	Cho phép nhiều dòng trong trường. Có thể kết hợp với text và các biến thể của nó.
textImeMultiLine	0x00040001	Mặc dù văn bản nhập không nhiều dòng, IME nên cung cấp nhiều dòng có thể. Có thể kết hợp với text và các biến thể của nó.
textUri	0x00000011	Văn bản là URI.
textEmailAddress	0x00000021	Văn bản là địa chỉ e-mail. Bàn phím có ký tự @.
textEmailSubject	0x00000031	Văn bản được cung cấp như subject của e-mail.
textShortMessage	0x00000041	Văn bản là nội dung của thông điệp ngắn.
textLongMessage	0x00000051	Văn bản là nội dung của thông điệp dài.
textPersonName	0x00000061	Văn bản là tên của một cá nhân.
textPostalAddress	0x00000071	Văn bản được cung cấp như địa chỉ gửi thư bưu chính.
textPassword	0x00000081	Văn bản là password.
textVisiblePassword	0x00000091	Văn bản là password được hiển thị.
textWebEditText	0x000000a1	Văn bản được cung cấp như văn bản dạng web.
textFilter	0x000000b1	Văn bản được lọc một số dữ liệu.
textPhonetic	0x000000c1	Văn bản cho phát âm ngữ âm. Như trường tên phiên âm trong một mục contact.
number	0x00000002	Trường chỉ nhập số.
numberSigned	0x00001002	Số có dấu. Có thể kết hợp với number và các tùy chọn khác của nó.
numberDecimal	0x00002002	Số thập phân (phân số). Có thể kết hợp với number và các tùy chọn khác của nó.
phone	0x00000003	Để nhập số điện thoại. Bàn phím ngoài ký tự số, còn thêm () . / Pause Wait # - +
datetime	0x00000004	Để nhập ngày và giờ.
date	0x00000014	Để nhập một ngày. Dữ liệu hợp lệ: 12/31/2014, 12-31-2014, 12.31.2014
time	0x00000024	Để nhập một thời điểm. Bàn phím ngoài ký tự số, còn có : và ký tự để tạo AM, PM.

Trị của các thuộc tính thường dùng kết hợp:

android:inputType="number|numberSigned|numberDecimal"

Bàn phím ảo sẽ hiển thị số, các phím không phải số vẫn hiển thị nhưng không hoạt động, chỉ nhập được số. Chấp nhận số nguyên có dấu và số thực. Sau khi thiết lập phương pháp nhập cho EditText, để thấy kết quả bạn phải cấu hình bàn phím ảo cho AVD: vào Settings > Language & input, phần KEYBOARD & INPUT METHODS, click Default; trong hộp thoại Choose input method, phần Hardware Physical keyboard, chọn OFF.

4. View tùy biến

Android cung cấp một danh sách phong phú các view tạo sẵn như Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, AutoCompleteTextView, v.v... mà bạn có thể sử dụng trực tiếp khi phát triển ứng dụng của bạn. Tuy nhiên, trong nhiều tình huống, bạn muốn đóng gói các khối code tạo giao diện lặp đi lặp lại, tối ưu tốc độ hiển thị, điều khiển hoàn toàn về kích thước và việc vẽ lên view, thêm nhận biết gestures, v.v... Android sẽ cung cấp cho bạn khả năng tạo ra các view tùy biến phù hợp với nhu cầu của bạn.

a) Tạo view tùy biến

View tùy biến của bạn có thể thừa kế trực tiếp từ lớp View hoặc từ các lớp con có sẵn của nó, kể cả các loại layout. Các bước như sau:

- Tạo lớp view tùy biến thừa kế TextView:

```
public class DateView extends TextView {
    public DateView(Context context) {
        this(context, null);
    }

    public DateView(Context context, AttributeSet attrs) {
        super(context, attrs);
        // cài đặt thêm các chức năng tùy biến
        setDate();
    }

    private void setDate() {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
        setText(dateFormat.format(Calendar.getInstance().getTime()));
    }
}
```

Nếu bạn cần cài đặt tùy biến việc vẽ hay thay đổi kích thước của view, hãy xem xét viết lại các phương thức sau:

- `onDraw(Canvas)`, tham số cho phương thức này là đối tượng lớp `android.graphics.Canvas`, rất giống trong Java Swing, có các phương thức quen thuộc như `drawLine()`, `drawPath()`, `drawText()`, `drawRect()`, `drawBitmap()`, ... mà bạn có thể gọi để vẽ lên view của bạn.

- `onMeasure()` báo cho Android biết kích thước của view tùy biến, dựa trên các thuộc tính về kích thước đã thiết lập cho nó. Bạn có thể tùy biến lại kích thước nếu muốn kết quả khác với `match_parent` và `wrap_content`.

Hai tham số kiểu `int` truyền cho phương thức này thể hiện bộ (chế độ, kích thước) thiết lập cho `android:layout_width` và `android:layout_height` của view tùy biến trong tập tin layout XML. Ví dụ, bạn thiết lập thuộc tính cho view tùy biến: `android:layout_width="match_parent"`, parent có width là 400, thì `widthMode` là `View.MeasureSpec.EXACTLY` và `widthSize` là 400.

Các chế độ gồm:

- + `EXACTLY`: kích thước đã được thiết lập cụ thể, nên dùng kích thước này. Khi sử dụng `match_parent` và parent có kích thước cụ thể, cũng nhận được chế độ này.
- + `AT_MOST`: kích thước được thiết lập là `match_parent` hoặc `wrap_content`, với trị tối đa được thiết lập (`android:maxWidth` hoặc `android:maxHeight`), không nên vượt trị này.
- + `UNSPECIFIED`: kích thước được thiết lập là `wrap_content`, kích thước do bạn chọn.

Sau khi lấy kích thước và chế độ truyền vào, bạn tính toán kích thước view tùy biến và trả về kích thước view bạn muốn bằng phương thức `setMeasuredDimension()`.

- `onLayout()`, nếu view tùy biến của bạn phức tạp, có layout riêng cho các view con.

```
@Override protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    int desiredWidth = 100;
    int desiredHeight = 100;

    int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    int heightMode = MeasureSpec.getMode(heightMeasureSpec);
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);

    int width;
    int height;

    // đo width
    if (widthMode == MeasureSpec.EXACTLY) {
        width = widthSize;
    } else if (widthMode == MeasureSpec.AT_MOST) {
        width = Math.min(desiredWidth, widthSize); // không nên lớn hơn kích thước được cung cấp
    } else {
        width = desiredWidth; // tùy theo bạn muốn
    }

    // đo height
    if (heightMode == MeasureSpec.EXACTLY) {
        height = heightSize;
    } else if (heightMode == MeasureSpec.AT_MOST) {
        height = Math.min(desiredHeight, heightSize);
    } else {
        height = desiredHeight;
    }
    setMeasuredDimension(width, height);
}
```

- Tạo thực thể view tùy biến bằng code trong activity/fragment:

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // tạo một thực thể view tùy biến và đặt nó trong layout
    DateView dateView = new DateView(this);
    setContentView(dateView);
}
```

hoặc, bằng khai báo trong tập tin layout XML của activity/fragment:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <com.twe.examples.cviewdemo.DateView
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

b) Tạo thuộc tính tùy biến cho view

Các view tùy biến được cấu hình bằng các thuộc tính mặc định tương ứng với lớp cha của nó. Ngoài ra, Android cũng cho phép tạo và dùng các thuộc tính tùy biến, ví dụ:

```
<com.twe.examples.cviewdemo.DateView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    custom:delimiter="- " />
```

Trước tiên, định nghĩa các thuộc tính tùy biến này trong /res/values/attrs.xml:

```
<resources>
    <declare-styleable name="DateView">
        <attr name="delimiter" format="string"/>
    </declare-styleable>
</resources>
```

Đọc các thuộc tính này từ tập tin XML và thiết lập chúng cho view tùy biến.

```
public class DateView extends TextView {
    public String delimiter;

    public DateView(Context context) {
        this(context, null);
    }

    public DateView(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.DateView);
        final int n = a.getIndexCount();
        for (int i = 0; i < n; ++i) {
            int attr = a.getIndex(i);
            switch (attr) {
                case R.styleable.DateView_delimiter:
                    delimiter = a.getString(attr);
                    setDate();
                    break;
            }
        }
        a.recycle();
    }

    private void setDate() {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd" + delimiter + "MM" + delimiter + "yyyy");
        setText(dateFormat.format(Calendar.getInstance().getTime()));
    }
}
```

Fragment

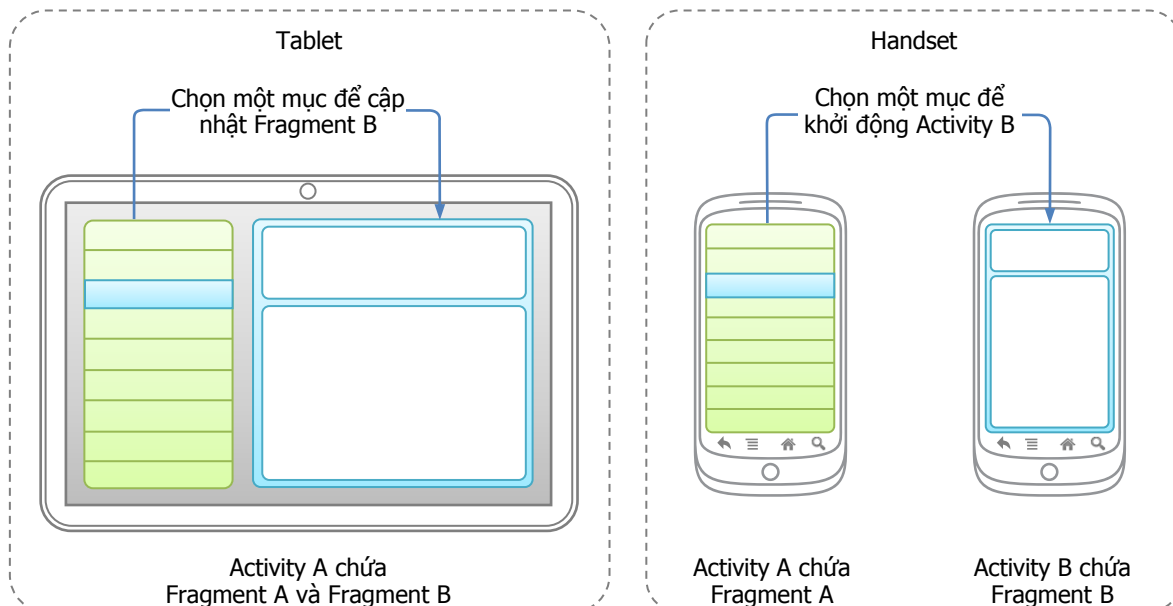
Fragment thay đổi cơ bản kiến trúc của ứng dụng, từ dựa trên activity thành dựa trên fragment, đặc biệt trên thiết bị Android có màn hình rộng như tablet. Ngoài ra, fragment cũng kết hợp với các UI widget khác như ActionBar, ViewPager, NavigationDrawer tạo nên khả năng điều hướng linh động cho ứng dụng.

Một fragment là một phần của giao diện người dùng đặt trong một activity, thực hiện một tác vụ con cho activity. Fragment được xem như một sub-activity. Trước đây, chúng ta có hạn chế là chỉ trình bày được một activity duy nhất trên màn hình tại một thời điểm. Vì thế, chúng ta không thể chia màn hình điện thoại thành nhiều phần và kiểm soát các phần một cách riêng biệt. Bây giờ, chúng ta giải quyết vấn đề đó với một activity chứa nhiều fragment, chúng có layout, sự kiện, và vòng đời riêng.

Sau đây là những điểm quan trọng về fragment:

- Một fragment có layout riêng, tác vụ riêng và vòng đời riêng.
- Có thể thêm, loại bỏ, thay thế các fragment của một activity trong lúc activity đó đang chạy.
- Có thể kết hợp nhiều fragment trong một activity duy nhất để xây dựng một activity có giao diện nhiều vùng (multi-pane UI).
- Một fragment có thể được dùng trong nhiều activity.
- Vòng đời của fragment liên quan chặt chẽ đến vòng đời activity chứa nó, gọi là host activity. Có nghĩa là khi activity dừng lại, tất cả các fragment có trong activity đó cũng dừng. Fragment không thể chạy độc lập ngoài activity.
- Một fragment thường có giao diện người dùng, gọi là UI fragment. Tuy nhiên, fragment có thể thực hiện hành vi mà không có thành phần giao diện người dùng, gọi là fragment không UI.

Hình dưới là ví dụ điển hình về tình huống sử dụng fragment. Ứng dụng có hai khối UI định nghĩa bởi các fragment. Hai fragment này có thể kết hợp vào một activity như thiết kế trên tablet, hoặc bố trí tách biệt như thiết kế trên điện thoại cầm tay.



Activity A, có thể chứa cả hai fragment khi chạy trên thiết bị màn hình rộng như tablet. Tuy nhiên, trên màn hình nhỏ của điện thoại cầm tay, không đủ chỗ cho cả hai fragment. Vì vậy, activity A chỉ chứa fragment A, liệt kê các mục, khi người dùng chọn một mục, sẽ khởi động activity B chứa fragment B, hiển thị nội dung chi tiết về mục đó.

Fragment được thêm vào Android API phiên bản Honeycomb (API level 11), trong gói `android.app`, với các lớp chính: `Fragment`, `FragmentManager` và `FragmentTransaction`.

Để tương thích với các phiên bản trước đó, bạn có thể dùng thư viện hỗ trợ cho fragment `android.support.v4.app` có các lớp chính: `FragmentActivity`, `Fragment`, `FragmentManager`, `FragmentTransaction`. Lúc này, activity của bạn phải thừa kế từ lớp cơ sở `android.support.v4.app.FragmentActivity`, thay vì `android.app.Activity`.

1. Vòng đời của fragment

Fragment có vòng đời tương tự activity, liên quan chặt chẽ với vòng đời của activity chứa nó. Ví dụ, khi activity nhận `onPause()`, từng fragment chứa trong nó cũng nhận `onPause()`. Điểm khác chủ yếu là các phương thức callback trong vòng đời fragment được gọi bởi `FragmentManager` của activity chứa nó, chứ không phải bởi Android OS như activity.

Vòng đời của fragment gồm các tầng sau:

Giai đoạn 1: khi fragment được thêm vào `FragmentManager`, các phương thức sau sẽ lần lượt được gọi:

`onAttach()`, gọi khi một fragment liên kết với activity chứa nó.

`onCreate()`, gọi khi khởi tạo fragment, khởi tạo các thành viên và tài nguyên của fragment tại đây. Đây cũng là nơi bạn phục hồi trạng thái của fragment mà bạn đã giữ lại khi fragment tạm dừng (pause) hoặc dừng lại (stop).

`onCreateView()`, gọi khi tạo cây view liên kết với fragment, "inflate" layout của fragment để vẽ nên UI của fragment lần đầu. Phương thức này trả về một đơn thể View, đây là gốc cây view của fragment. Với fragment không UI, không cần viết lại phương thức này.

Bạn có thể tạo layout bằng hai cách:

- trực tiếp bằng code, tạo thực thể và cấu hình các view.
- bằng khai báo, cung cấp tập tin layout XML và "inflate" layout từ đó. Để hỗ trợ "inflate", `onCreateView()` nhận tham số là `LayoutInflater` và `ViewGroup` từ layout của activity chứa fragment, phục vụ như cha của layout của fragment.

`onActivityCreated()`, gọi khi `onCreate()` của activity kết thúc.

Giai đoạn 2: khi fragment hiển thị, nó sẽ lần lượt qua các trạng thái: `onStart()`, `onResume()`.

Giai đoạn 3: khi fragment vào chế độ nền, nó sẽ lần lượt qua các trạng thái: `onPaused()`, `onStop()`.

Giai đoạn 4: khi fragment bị hủy, nó sẽ lần lượt qua các trạng thái:

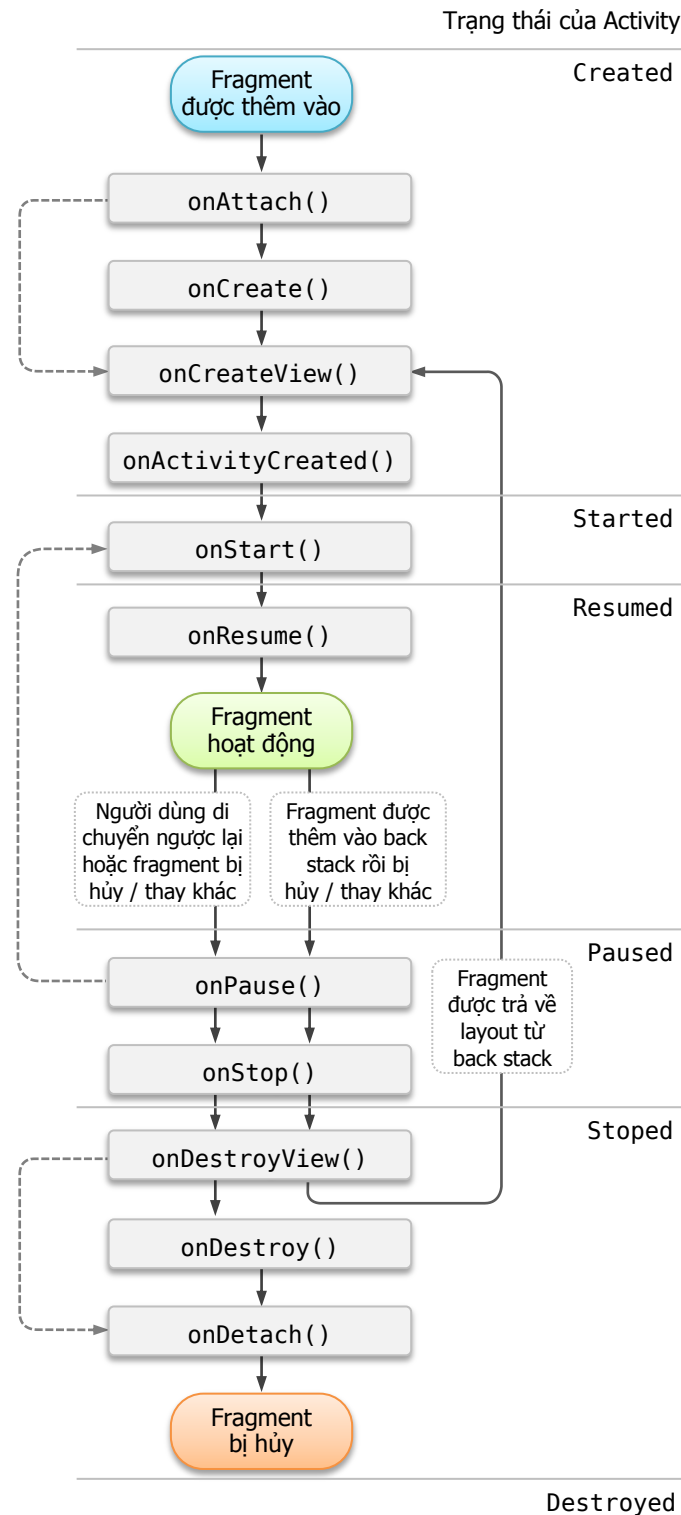
`onPaused()`, gọi khi người dùng rời khỏi fragment. Đây là nơi bạn lưu trữ trạng thái fragment nếu cần.

`onStop()`

`onDestroyView()`, gọi khi cây view liên kết với fragment bị hủy.

`onDestroy()`

`onDetach()`, gọi khi liên kết giữa activity và fragment bị hủy.



Vòng đời của fragment

Chú ý, khi viết lại (overriding) các phương thức callback, ngoại trừ onCreateView(), bạn phải gọi phương thức cài đặt tương ứng của lớp cha (super.onXxx()), nếu không sẽ nhận exception.

2. Làm việc với fragment

a) Thêm fragment đến activity

Có hai cách thêm fragment vào activity:

- Thêm fragment tĩnh bằng layout

Khai báo fragment bên trong tập tin layout XML của activity chứa nó. Fragment luôn hiển thị bên trong layout của activity, tuy nhiên không thể `remove()`, `replace()`, `detach()` hoặc `attach()` nó trong thời gian chạy được.

Khai báo này được thực hiện bằng element `<fragment>`, trong đó thuộc tính `android:name` cung cấp tên lớp của fragment.

Mỗi fragment cần một định danh duy nhất, có ba cách cung cấp định danh cho fragment:

- + Fragment cần một định danh duy nhất, có ba cách cấp định danh cho fragment.
- + Cung cấp thuộc tính android:id với định danh duy nhất.
- + Cung cấp thuộc tính android:tag với tag là chuỗi duy nhất (thường dùng với fragment không UI).
- + Nếu không dùng hai cách trên, hệ thống sử dụng ID của container chứa fragment.

Ví dụ:

Tạo fragment ToolBarFragment:

```
public class ToolBarFragment extends Fragment implements SeekBar.OnSeekBarChangeListener {
    @Override public View onCreateView(LayoutInflater inflater,
                                       ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_toolbar, container, false);
        SeekBar seekBar = (SeekBar) view.findViewById(R.id.seek_id);
        seekBar.setOnSeekBarChangeListener(this);
        return view;
    }

    @Override public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        ImageFragment fragment = (ImageFragment) getFragmentManager().findFragmentById(R.id.fragment_image);
        fragment.scaleImage(progress);
    }
    @Override public void onStartTrackingTouch(SeekBar seekBar) { }
    @Override public void onStopTrackingTouch(SeekBar seekBar) { }
}
```

có layout fragment_toolbar.xml chứa một SeekBar:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <SeekBar android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/seek_id"/>
</LinearLayout>
```

Tạo fragment ImageFragment:

```
public class ImagesFragment extends Fragment {
    ImageView mImageView;
    @Override public View onCreateView(LayoutInflater inflater,
                                       ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_images, container, false);
        mImageView = (ImageView) view.findViewById(R.id.image_id);
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.mini);
        mImageView.setImageBitmap(bitmap);
        return view;
    }

    public void scaleImage(int value) {
        Matrix matrix = new Matrix();
        matrix.setScale(value / 100f + 1, value / 100f + 1); // ánh xạ [0, 100] vào [1, 2]
        mImageView.setImageMatrix(matrix);
    }
}
```

có layout fragment_image.xml chứa một ImageView:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/image_id"
        android:scaleType="matrix" />
</LinearLayout>
```

Khi thiết kế layout cho activity_main.xml, trong Palette của Designer Tool, chọn Custom > <fragment>, rồi chọn hai fragment trên trong danh sách Fragments hiển thị, kéo thả vào layout của activity_main.xml.

- Thêm fragment động bằng code

Đầu tiên, trong tập tin layout XML của activity, tạo một điểm để đặt fragment trong cây phân cấp các view của activity, được xem như một view group dành chỗ (placeholder) cho fragment.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Với mỗi UI fragment, tạo tập tin layout XML riêng, định nghĩa layout cho từng fragment, ví dụ fragment_main.xml. Trong phương thức callback onCreateView() của lớp Fragment, tạo UI cho nó bằng cách dùng LayoutInflater được cung cấp "inflater" layout của nó từ tập tin layout XML tương ứng.

```
public class MainFragment extends Fragment {
    private EditText editText;

    @Override public View onCreateView(LayoutInflater inflater,
                                       ViewGroup container, Bundle savedInstanceState) {
```



```
View rootView = inflater.inflate(R.layout.fragment_main, container, false);
editText = (EditText) rootView.findViewById(R.id.title);
return rootView;
}
}
```

Trong onCreate() của lớp activity chứa fragment:

- + Lấy FragmentManager bằng getFragmentManager() hoặc getSupportFragmentManager().
- + Gọi beginTransaction() của FragmentManager để tạo và trả về một thực thể FragmentTransaction.
- + Dùng đối tượng FragmentTransaction thêm fragment bằng phương thức add(). Tham số thứ nhất của phương thức này là ViewGroup mà fragment sẽ được thêm vào, chỉ định bởi resource ID. Với fragment không UI, bạn có thể cung cấp thêm một chuỗi tag dùng định danh fragment đó.

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // kiểm tra xem hệ thống có đang tạo lại activity không. Nếu Bundle không null,
    // hệ thống đã tạo lại activity, activity sẽ tự động khởi tạo lại các fragment.
    if (savedInstanceState == null) {
        getFragmentManager()
            .beginTransaction()
            .add(R.id.container, new MainFragment())
            .commit();
    }
}
```

Mặc định, khi activity được tạo lại, các fragment của nó cũng bị hủy và tự động tạo lại. Bạn có thể yêu cầu hệ thống giữ lại thực thể fragment hiện hành nếu activity được tạo lại, bằng phương thức Fragment.setRetainInstance(boolean) với tham số true. Như vậy, khi activity tạo lại, onDestroy() và onCreate() của fragment sẽ không được gọi.

b) FragmentManager và FragmentTransaction

Mỗi activity có một FragmentManager, dùng để quản lý các fragment của activity đó. FragmentManager xử lý:

- Một danh sách các fragment, FragmentManager thêm view của fragment chúng vào cây view phân cấp của activity.
- Một back stack các fragment transaction.

Bạn lấy FragmentManager của activity bằng getFragmentManager() hoặc getSupportFragmentManager().

Sau đó, dùng FragmentManager để:

- Lấy một fragment đang tồn tại trong activity bằng findFragmentById() (với UI fragment) hoặc findFragmentByTag() (với fragment không UI).
- Đẩy (pop) fragment ra khỏi back stack (tương tự với nhấn nút Back), với popBackStack().
- Đăng ký một listener lắng nghe sự kiện thay đổi back stack, với addOnBackStackChangeListener().

Bạn có thể thực hiện nhiều tác vụ quan trọng trên fragment như thêm, xóa, gắn vào, gỡ ra hoặc thay thế. Muốn thực hiện, bạn phải gọi các phương thức tương ứng của FragmentTransaction. Nó giữ vai trò trung tâm khi sử dụng fragment. Các phương thức của nó trả về tham chiếu đến cùng thực thể FragmentTransaction, nên có thể gọi thành một chuỗi các phương thức.

```
getFragmentManager()
    .beginTransaction()
    .remove(fragment1)
    .add(R.id.fragment_container, fragment2)
    .show(fragment3)
    .hide(fragment4)
    .commit();
```

FragmentTransaction có phương thức addToBackStack(String) để đưa transaction vào back stack, trước khi "commit" nó. Tham số kiểu String là chuỗi tùy chọn để định danh trạng thái của back stack, hoặc null. Hệ thống sẽ triệu gọi onPause(), onStop() và onDestroyView() của fragment khi đặt nó vào back stack. Khi dùng dùng nhấn nút Back, hệ thống sẽ triệu gọi onCreateView(), onActivityCreated(), onStart() và onResume() của fragment.

c) Tích hợp action bar / options menu vào fragment

Fragment có thể thêm các item của nó vào action bar hoặc options menu của activity.

- Đầu tiên bạn phải gọi Fragment.setHasOptionsMenu() trong phương thức onCreate() của fragment.
- Viết lại các phương thức callback sau đây của fragment: onCreateOptionsMenu() và onOptionsItemSelected(). Bạn có thể tùy chọn cài đặt cho onPrepareOptionsMenu(), onOptionsItemSelected(), và onDestroyOptionsMenu().

Trong fragment:

```
public class MyFragment extends Fragment {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        inflater.inflate(R.menu.fragment_menu, menu);
    }
}
```

```

@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.fraction_menu_info:
            // xử lý item của fragment menu
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

Các phương thức tương ứng của activity chứa fragment sẽ tự động triệu gọi các phương thức cài đặt ở trên của fragment. Nếu bạn có viết lại các phương thức này của activity, nhớ gọi `super()` trong phương thức tương ứng của fragment.

Lớp activity chứa fragment:

```

public class MyActivity extends Activity {
    @Override public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        getMenuInflater().inflate(R.menu.activity_menu, menu);
        return true;
    }

    @Override public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.activity_menu_info:
                // xử lý item của activity menu
                return true;
            default:
                // xử lý item của fragment menu
                return super.onOptionsItemSelected(item);
        }
    }
}

```

d) Giao tiếp giữa fragment và activity

Fragment có thể giao tiếp trực tiếp với activity chứa nó:

- Activity có thể gọi các phương thức public của fragment thông qua tham chiếu đến đối tượng fragment. Nếu bạn không lưu trữ tham chiếu đến fragment nào trong activity, dùng `findFragmentById()` và `findFragmentByTag()` của đối tượng `FragmentManager` để lấy tham chiếu.

- Ngược lại, fragment truy cập đến thực thể activity chứa nó bằng phương thức `getActivity()`.

Ví dụ, từ fragment lấy tham chiếu đến view chỉ định thuộc activity chứa fragment đó:

```
View listView = getActivity().findViewById(R.id.list);
```

Fragment không thừa kế từ `Context`, muốn lấy `Context` phải thông qua activity hoặc gọi `getApplicationContext()`.

Ngay sau khi tạo một fragment, có thể cung cấp cho nó một tập các tham số:

- Tạo một đối tượng `Bundle`, đặt vào nó các trị bạn muốn cung cấp như tham số đến fragment.
- Triệu gọi phương thức `setArguments(Bundle)` của fragment, truyền cho fragment `Bundle` trên.
- Bên trong fragment, triệu gọi `getArguments()` khi bạn muốn lấy `Bundle`.

Trong fragment:

```

public static class DetailFragment extends Fragment {
    public static DetailFragment newInstance(int index) {
        DetailFragment fragment = new DetailFragment();
        Bundle args = new Bundle();
        args.putInt("index", index);
        fragment.setArguments(args);
        return fragment;
    }

    public int getShownIndex() {
        return getArguments().getInt("index", 0);
    }
    // ...
}

```

Trong activity:

```
DetailFragment detail = DetailFragment.newInstance(2);
```

e) Cài đặt dialog bằng fragment

Từ Android 3+, với khuyến khích thiết kế dùng fragment, các phương thức dùng quản lý dialog của activity trở nên lạc hậu. Lớp `DialogFragment` trở thành lớp cơ sở dùng quản lý các dialog dựa trên fragment.

- `DialogFragment` cài đặt một fragment hiển thị một cửa sổ dialog, nổi lên trên cửa sổ activity.

- Fragment này chứa một đối tượng Dialog. Bạn có thể điều khiển dialog, hiển thị hoặc hủy nó, thông qua các phương thức của DialogFragment, không cần gọi trực tiếp các phương thức của dialog.
 - Nếu bạn muốn dùng fragment chỉ như một dialog, viết lại phương thức onCreateDialog() và trả về một thực thể Dialog hoặc lớp con của nó.
 - DialogFragment có phương thức show() dùng thêm fragment và phương thức dismiss() dùng hủy bỏ fragment.
- Dialog dựa trên fragment:

```
public class ConfirmationDialogFragment extends DialogFragment
    implements DialogInterface.OnClickListener {
    private ConfirmationDialogFragmentListener listener;
    public static ConfirmationDialogFragment newInstance(int title) {
        ConfirmationDialogFragment fragment = new ConfirmationDialogFragment();
        Bundle args = new Bundle();
        args.putInt("title", title);
        fragment.setArguments(args);
        return fragment;
    }

    // listener sẽ được cài đặt rồi thiết lập bằng setter cho fragment sau khi tạo
    public interface ConfirmationDialogFragmentListener {
        public void onPositiveClick();
        public void onNegativeClick();
    }

    public void setListener(ConfirmationDialogFragmentListener listener) {
        this.listener = listener;
    }

    @Override public Dialog onCreateDialog(Bundle savedInstanceState) {
        int title = getArguments().getInt("title");
        return new AlertDialog.Builder(getActivity())
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setTitle(title)
            .setPositiveButton(android.R.string.ok, this)
            .setNegativeButton(android.R.string.cancel, this)
            .create();
    }

    @Override public void onClick(DialogInterface dialog, int which) {
        if (listener != null) {
            switch (which) {
                case DialogInterface.BUTTON_POSITIVE: listener.onPositiveClick();
                default: listener.onNegativeClick();
            }
        }
    }
}
```

Dùng dialog dựa trên fragment trong activity:

```
public class MainActivity extends FragmentActivity
    implements OnClickListener, ConfirmationDialogFragmentListener {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // gán listener cho nút
        Button buttonPostDialog = (Button) findViewById(R.id.button_post_dialog);
        buttonPostDialog.setOnClickListener(this);
    }

    // cài đặt cho OnClickListener, listener của nút trong activity
    @Override public void onClick(View v) {
        ConfirmationDialogFragment dialog =
            ConfirmationDialogFragment.newInstance(R.string.dialog_format_title);
        dialog.setListener(this);
        dialog.show(getFragmentManager(), null);
    }

    // cài đặt cho ConfirmationDialogFragmentListener, listener của các nút trong dialog
    @Override public void onPositiveClick() {
        Toast.makeText(this, android.R.string.ok, Toast.LENGTH_LONG).show();
    }
}
```

```
@Override public void onNegativeClick() {
    Toast.makeText(this, android.R.string.cancel, Toast.LENGTH_LONG).show();
}
}
```

Cũng theo khuôn hướng thiết kế dùng fragment, một số lớp con của Fragment được cung cấp:

ListFragment

Một fragment tự động quản lý một ListView. Giống với ListActivity.

PreferenceFragment

Một fragment tự động quản lý một tập các đối tượng Preference. Dùng tài nguyên XML preference, fragment có thể tự động tạo một giao diện để hiển thị và biên tập một tập các preference. Giống với PreferenceActivity.

WebFragment

Một fragment tự động tạo và quản lý một webView.

3. Kiến trúc ứng dụng với fragment

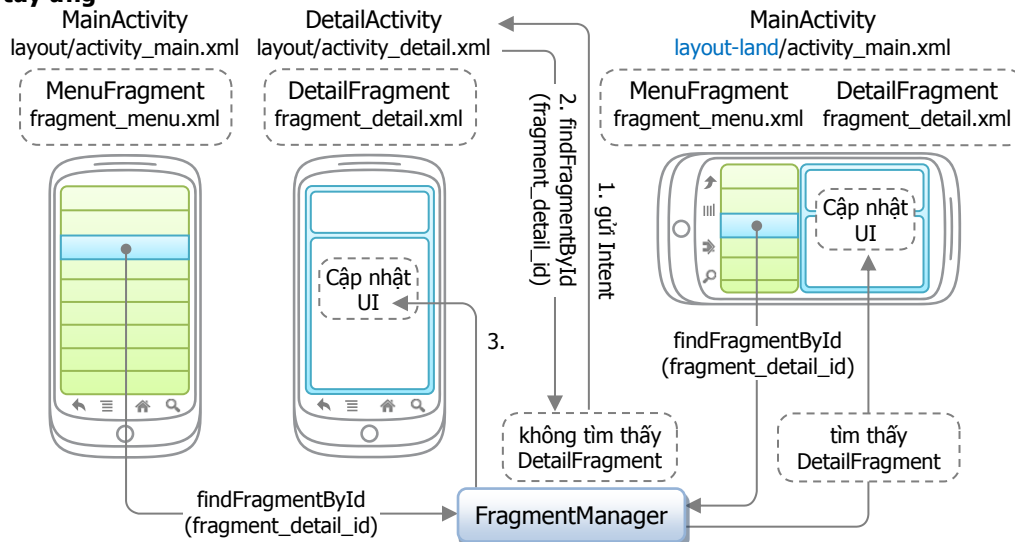
Các công cụ của SDK cung cấp một số application templates giúp bạn nhanh chóng tạo nên ứng dụng có dùng fragment với cấu trúc linh động. Bạn nên học tập các template này để nắm bắt cách sử dụng fragment trong ứng dụng.

Khi tạo project Android mới, bạn chọn một trong các template:

- Blank Activity with Fragment: cấu trúc ứng dụng dùng fragment cơ bản.
- Master/Detail Flow: lựa chọn các mục liệt kê trong một ListFragment, chi tiết về mục đó hiển thị trong một fragment khác.
- Navigation Drawer Activity: lựa chọn các mục liệt kê trong một drawer trượt từ bên trái vào, chi tiết về mục đó hiển thị trong fragment nền bên dưới.
- Tabbed Activity: bạn có thể thực hiện một số kiểu điều hướng như swipe views, action bar tabs, action bar spinner. Nội dung của các phần chứa trong các fragment.

Trong phần sau, chúng tôi trình bày một ví dụ tiếp cận dùng fragment, giúp bạn nắm bắt kỹ thuật thiết kế ứng dụng dùng fragment.

Master/Detail tùy ứng



Khi khởi chạy ở chế độ màn hình đứng (portrait), layout của MainActivity là layout/activity_main.xml, layout này chứa một fragment là MenuFragment. Nếu bạn xoay thiết bị, layout của MainActivity là layout-land/activity_main.xml, layout này chứa hai fragment: MenuFragment nằm bên trái và DetailFragment nằm bên phải.

Khi MenuFragment có mặt, nó "inflate" layout trong fragment_menu.xml. Layout này chứa một ListView nạp dữ liệu từ một adapter tùy biến, ví dụ CustomAdapter. Mỗi mục liên quan tới một đối tượng, ví dụ đối tượng thuộc lớp Item.

Khi tap một mục trong ListView, phương thức `findFragmentById()` của FragmentManager được triệu gọi với tham số là id của DetailFragment. Kết quả là một hai khả năng sau:

- Khi thiết bị trong chế độ landscape, DetailFragment được tìm thấy, cập nhật trực tiếp UI của nó với thông tin của Item tương ứng trong ListView.

- Khi thiết bị trong chế độ portrait, DetailFragment không được tìm thấy (bằng null), tiến hành:

+ Gửi Intent (kèm Bundle chứa vị trí của Item liên quan) để khởi chạy DetailActivity.

```
public class MenuFragment extends ListFragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_list, container, false);
        // CustomAdapter là adapter tùy biến, lớp thành viên của MenuFragment
        // ItemList là nguồn dữ liệu singleton (xem cuối chương Data Persistence – SQLite)
        setListAdapter(new CustomAdapter(ItemList.get(getActivity()).getList()));
        return view;
    }

    // gọi khi muốn cập nhật dữ liệu của ListView
    public void notifyChanged() {
```

```

    ((CustomAdapter) getListAdapter()).notifyDataSetChanged();
}

@Override public void onItemClick(ListView l, View v, int position, long id) {
    DetailFragment detailfragment = (DetailFragment) getFragmentManager()
        .findFragmentById(R.id.fragment_detail_id);
    if (detailfragment != null && detailfragment.isInLayout()) {
        Item item = ItemList.get(getActivity()).getItem(position);
        detailfragment.updateView(item);
    } else {
        Intent intent = new Intent(v.getContext(), DetailActivity.class);
        Bundle bundle = new Bundle();
        bundle.putInt("index", position);
        intent.putExtras(bundle);
        startActivity(intent);
    }
}

// cập nhật ListView trong chế độ landscape
@Override public void onResume() {
    super.onResume();
    notifyDataSetChanged();
}

class CustomAdapter extends ArrayAdapter<Item> {
    // ...
}
}

```

+ DetailActivity, có layout trong layout/activity_detail.xml. Layout này chứa DetailFragment, có layout là /layout/fragment_detail.xml.

+ DetailActivity cũng gọi findFragmentById() của FragmentManager để tìm DetailFragment và cập nhật UI cho DetailFragment với thông tin lấy được từ id trong Bundle nhận được.

```

public class DetailActivity extends FragmentActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE) {
            finish();
            return;
        }
        setContentView(R.layout.activity_detail);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            int position = extras.getInt("index");
            DetailFragment detailFragment = (DetailFragment) getFragmentManager()
                .findFragmentById(R.id.fragment_detail_id);
            Item item = ItemList.get(this).getItem(position);
            detailFragment.updateView(item);
        }
    }
}

```

Trong DetailFragment, nếu thay đổi thông tin của Item, ListView cần phải được cập nhật.

```

public class DetailFragment extends Fragment {
    EditText et;
    Item currentItem = null;
    MenuFragment menuFragment;

    @Override public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_detail, container, false);
        menuFragment = (MenuFragment) getFragmentManager().findFragmentById(R.id.fragment_menu);
        et = (EditText) view.findViewById(R.id.etInfo);
        et.addTextChangedListener(new TextWatcher() {
            @Override public void onTextChanged(CharSequence s, int start, int before, int count) {
                currentItem.setInfo(s.toString());
                if (menuFragment != null) menuFragment.notifyChanged();
            }

            @Override public void beforeTextChanged(CharSequence s, int start, int count, int after) {}
            @Override public void afterTextChanged(Editable s) {}
        });
    }
}

```



```

    return view;
}

public void updateView(Version version) {
    currentVersion = version;
    et.setText(currentItem.getInfo());
}
}

```

Phân biệt hai trường hợp cập nhật ListView trong MenuFragment:

- Nếu tìm thấy MenuFragment thì thiết bị đang trong chế độ landscape, cập nhật trực tiếp:

```

MenuFragment menuFragment = (MenuFragment) getFragmentManager().findFragmentById(R.id.fragment_menu);
menuFragment.notifyDataSetChanged();

```

- Nếu thiết bị trong chế độ portrait, việc cập nhật tiến hành khác, dùng một trong hai cách:

+ Trong MenuFragment, khi gửi intent, gọi startActivityForResult(intent, 0). Đồng thời, viết lại phương thức onActivityResult() của MenuFragment, trong đó cập nhật ListView của MenuFragment:

```

@Override public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    notifyDataSetChanged();
}

```

+ Trong MenuFragment, khi gửi intent, gọi startActivity(intent). Đồng thời, viết lại phương thức onResume() của MenuFragment, trong đó cập nhật ListView của MenuFragment:

```

@Override public void onResume() {
    super.onResume();
    notifyDataSetChanged();
}

```

Xử lý sự kiện

1. Xử lý sự kiện

Ứng dụng Android là ứng dụng hướng sự kiện (event-driven). Sự kiện (event) được phát ra khi người dùng tương tác với giao diện của ứng dụng, ví dụ nhấn một nút hoặc chạm trên màn hình sẽ sinh ra sự kiện. Android duy trì một hàng đợi sự kiện để lưu các sự kiện khi chúng xuất hiện. Bạn có thể lấy các sự kiện này vào chương trình và thực hiện hành động đáp lại sự kiện tương ứng tùy theo yêu cầu.

Các khái niệm sau liên quan với quản lý sự kiện của Android:

- Event Listener: đối tượng lắng nghe sự kiện được thiết lập cho view, gọi tắt là listener. Listener được kích hoạt khi loại sự kiện mà nó lắng nghe xuất hiện do người dùng tương tác với view.
- Đăng ký Event Listener: thiết lập listener phù hợp cho view để lắng nghe các sự kiện quan tâm và cài đặt các Event Handler của listener đó để xử lý sự kiện.
- Event Handler: phương thức xử lý sự kiện, cài đặt trong listener tương ứng.

Event Handler	Event Listener và mô tả
onClick()	OnClickListener Sẽ được gọi khi người dùng click, chạm hoặc chọn (focus) vào một view như button, text, image, ... Bạn dùng Event Handler onClick() để xử lý sự kiện này.
onLongClick()	OnLongClickListener Sẽ được gọi khi người dùng click, chạm hoặc chọn (focus) kéo dài vài giây vào một view như button, text, image, ... Bạn dùng Event Handler onLongClick() để xử lý sự kiện này.
onFocusChange()	OnFocusChangeListener Sẽ được gọi khi view mất focus, nghĩa là người dùng chuyển sang một mục view khác. Bạn dùng Event Handler onFocusChange() để xử lý sự kiện này.
onKey()	OnKeyListener Sẽ được gọi khi người dùng chọn một mục view rồi nhấn hoặc nhả phím cứng trên thiết bị. Bạn dùng Event Handler onKey() để xử lý sự kiện này.
onTouch()	OnTouchListener Sẽ được gọi khi người dùng nhấn/nhả một phím, hoặc thực hiện một thao tác (gesture) nào đó trên màn hình. Bạn dùng Event Handler onTouch() để xử lý sự kiện này.
onMenuClick()	OnMenuClickListener Sẽ được gọi khi người dùng chọn một mục menu. Bạn dùng Event Handler onMenuClick() để xử lý sự kiện này.

Còn khá nhiều Event Listener khác như OnHoverListener, OnDragListener, OnCreateContextMenuListener, v.v...

Tùy theo tình huống, có một số cách cài đặt listener, được trình bày sau đây:

- Event Listener được cài đặt như một lớp listener riêng. Như vậy, nó có thể áp dụng cho nhiều UI control và có thể thay đổi độc lập. Tuy nhiên, nó chỉ có thể thay đổi thành viên của activity thông qua các phương thức public.
- Event Listener được cài đặt như lớp nội (Named Inner Class) của activity. Cách này thường được sử dụng vì Event Handler có thể truy cập tất cả thành viên của activity.
- Event Listener được cài đặt như một lớp nội vô danh (Anonymous Inner Class). Phương thức xử lý sự kiện được cài đặt bên trong lớp nội đó. Đây là cách thường được sử dụng.

```

TextView mTextView;
Button mButton;
@Override protected void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
mTextView = (TextView) findViewById(R.id.text_id);
mButton = (Button) findViewById(R.id.button_id);
mButton.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
        mTextView.setText("Button clicked!");
    }
});
mButton.setOnLongClickListener(new View.OnLongClickListener() {
    @Override public boolean onLongClick(View v) {
        mTextView.setText("Long button click!");
        return true;
    }
});
}

```

Chú ý, trị boolean trả về của onLongClick() là true, cho biết handler này đã *tiêu thụ* (consume) sự kiện, không truyền tiếp cho onClickListener xử lý.

- Activity cũng là Event Listener. Ta cài đặt giao diện listener cho activity, phương thức xử lý sự kiện trở thành phương thức cần phải định nghĩa trong activity. Listener có thể truy cập các thành viên private của activity.

```

TextView mTextView;
Button mButton;
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTextView = (TextView) findViewById(R.id.text_id);
    mButton = (Button) findViewById(R.id.button_id);
    mButton.setOnClickListener(this);
}

@Override public void onClick(View v) {
    if (v.getId() == R.id.button_id) {
        mTextView.setText("Button clicked!");
    }
}

```

- Chỉ định trực tiếp Event Handler trong tập tin layout XML, phương thức xử lý sự kiện phải được cài đặt như phương thức nghiệp vụ public của activity.

```

TextView mTextView;
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTextView = (TextView) findViewById(R.id.text_id);
}

public void sayHello(View v) {
    mTextView.setText("Button clicked!");
}

```

Trong tập tin layout XML:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView android:id="@+id/text_id"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:gravity="center"
        android:layout_gravity="center_horizontal"/>
    <Button android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/button_label"
        android:onClick="sayHello"
        android:layout_gravity="center_horizontal"/>
</LinearLayout>

```

- Thiết lập listener cho lớp UI (thường là fragment). Fragment chứa một *interface callbacks*, giữ vai trò như một listener trung gian. Activity cài đặt interface callbacks này và sẽ thiết lập listener cho fragment khi nó liên kết với fragment đó, trong phương thức callback onAttach().

```

public class MainFragment extends Fragment {
    private Button mButton;
}

```

```

private MainFragmentListener listener;

public interface MainFragmentListener {
    void action();
}

@Override public View onCreateView(LayoutInflater inflater,
                                   ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_main, container, false);
    mButton = (Button) view.findViewById(R.id.button_id);
    mButton.setOnClickListener(new View.OnClickListener() {
        @Override public void onClick(View v) {
            listener.action();
        }
    });
    return view;
}

@Override public void onAttach(Activity activity) {
    super.onAttach(activity);
    try {
        listener = (MainFragmentListener) activity;
    } catch (ClassCastException e) {
        throw new ClassCastException(activity.toString() + " must implement MainFragmentListener");
    }
}

@Override public void onDetach() {
    super.onDetach();
    listener = null;
}
}

```

Activity quản lý fragment:

```

public class MainActivity extends ActionBarActivity
    implements MainFragment.MainFragmentListener {
    @Override public void action() {
        Toast.makeText(this, "You tap me!", Toast.LENGTH_SHORT).show();
    }

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            getSupportFragmentManager()
                .beginTransaction()
                .add(R.id.container, new MainFragment())
                .commit();
        }
    }
}

```

2. Gesture

a) Các gesture đặc biệt

Một chuỗi chạm (touch) liên tục sẽ được xác định như một gesture (cử chỉ). Gesture bắt đầu khi người dùng chạm lần đầu vào màn hình, hệ thống sẽ theo dõi tiếp vị trí ngón tay người dùng, và kết thúc bằng cách bắt sự kiện cuối cùng khi ngón tay người dùng rời khỏi màn hình. Trong suốt chuỗi tương tác này, MotionEvent được chuyển sẽ cung cấp chi tiết về mọi tương tác. Khi người dùng chạm một hay nhiều ngón tay lên màn hình activity, phương thức callback onTouchEvent() của Activity sẽ được kích hoạt.

```

public class MainActivity extends ActionBarActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override public boolean onTouchEvent(MotionEvent event) {
        handleTouch(event);
        return true;
    }

    private void handleTouch(MotionEvent event) {

```

```

TextView mOneStatus = (TextView) findViewById(R.id.text_id_1);
TextView mTwoStatus = (TextView) findViewById(R.id.text_id_2);
int pointerCount = event.getPointerCount();
for (int i = 0; i < pointerCount; ++i) {
    int x = (int) event.getX(i);
    int y = (int) event.getY(i);
    int id = event.getPointerId(i);
    int action = event.getActionMasked();
    int actionIndex = event.getActionIndex();
    String actionString;
    switch (action) {
        case MotionEvent.ACTION_DOWN: actionString = "DOWN"; break;
        case MotionEvent.ACTION_UP: actionString = "UP"; break;
        case MotionEvent.ACTION_POINTER_DOWN: actionString = "PNTR DOWN"; break;
        case MotionEvent.ACTION_POINTER_UP: actionString = "PNTR UP"; break;
        case MotionEvent.ACTION_MOVE: actionString = "MOVE"; break;
        default: actionString = "";
    }
    String status = String.format("[%d] (%d, %d) %s ", actionIndex, x, y, actionString);
    if (id == 0) mOneStatus.setText(status);
    else mTwoStatus.setText(status);
}
}
}

```

Ta cũng có thể đăng ký `OnTouchListener` với view để chặn sự kiện `MotionEvent` và xử lý sự kiện bằng phương thức `onTouch()`. Ví dụ với layout của activity:

```

public class MainActivity extends ActionBarActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        RelativeLayout layout = (RelativeLayout) findViewById(R.id.layout_id);
        layout.setOnTouchListener(new RelativeLayout.OnTouchListener() {
            @Override public boolean onTouch(View view, MotionEvent event) {
                handleTouch(event);
                return true;
            }
        });
    }
    // ...
}

```

hoặc với fragment:

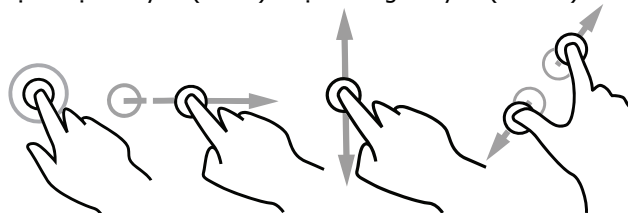
```

public static class MainFragment extends Fragment {
    private static final String DEBUG_TAG = "TOUCH";
    @Override public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                       Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container, false);
        rootView.setOnTouchListener(new View.OnTouchListener() {
            @Override public boolean onTouch(View view, MotionEvent event) {
                handleTouch(event);
                return true;
            }
        });
        return rootView;
    }
    // ...
}

```

Các phương thức `onTouchEvent()` và `onTouch()` có:

- Tham số `MotionEvent` chứa thông tin về sự kiện: vị trí chạm, loại hành động chạm.
- Trả về trị `boolean` cho biết sự kiện được chuyển (`true`) hoặc không chuyển (`false`) đến listener khác đăng ký cùng view.



Từ trái sang phải: double tap, fling, scroll và pinch

Đối với các gesture phức tạp như double tap (gõ nhẹ hai lần), fling (vuốt ngang hoặc dọc), scroll (cuộn ngang hoặc dọc), pinch (véo, pinch open: banh ra, pinch close: gom lại), Android cung cấp lớp `android.view.GestureDetector` để nhận sự kiện.

Bạn cần tạo đối tượng GestureDetector và gán listener cho nó. Các listener được định nghĩa như lớp nội của GestureDetector:

- GestureDetector.OnGestureListener

Bao gồm các phương thức onDown(), onShowPress(), onSingleTapUp(), onScroll(), onLongPress(), onFling().

- GestureDetector.OnDoubleTapListener

Bao gồm các phương thức onSingleTapConfirmed(), onDoubleTap(), onDoubleTapEvent().

Khi tạo GestureDetector, tham số thứ nhất là context, tham số thứ hai là listener. Sau đó, cài đặt cho onTouchEvent() của activity để nhận và chuyển tiếp sự kiện MotionEvent đến phương thức onTouchEvent() của GestureDetector.

```
public class MainActivity extends ActionBarActivity implements
    GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {
    private TextView mTextView;
    private GestureDetector detector;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.text_id);
        detector = new GestureDetector(this, this); // (context, OnGestureListener)
        detector.setOnDoubleTapListener(this);
    }

    @Override public boolean onTouchEvent(MotionEvent event) {
        this.detector.onTouchEvent(event);
        return super.onTouchEvent(event);
    }

    @Override public boolean onSingleTapConfirmed(MotionEvent event) {
        mTextView.setText("OnDoubleTapListener::onSingleTapConfirmed");
        return true;
    }

    @Override public boolean onDoubleTap(MotionEvent event) {
        mTextView.setText("OnDoubleTapListener::onDoubleTap");
        return true;
    }

    @Override public boolean onDoubleTapEvent(MotionEvent event) {
        mTextView.setText("OnDoubleTapListener::onDoubleTapEvent");
        return true;
    }

    @Override public boolean onDown(MotionEvent event) {
        mTextView.setText("OnDoubleTapListener::onDown");
        return true;
    }

    @Override public void onShowPress(MotionEvent event) {
        mTextView.setText("OnGestureListener::onShowPress");
    }

    @Override public boolean onSingleTapUp(MotionEvent event) {
        mTextView.setText("OnGestureListener::onSingleTapUp");
        return true;
    }

    @Override public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
        mTextView.setText("OnGestureListener::onScroll");
        return true;
    }

    @Override public void onLongPress(MotionEvent event) {
        mTextView.setText("OnGestureListener::onLongPress");
    }

    @Override public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        mTextView.setText("OnGestureListener::onFling");
        return true;
    }
}
```

Bạn cũng có thể đơn giản dùng listener GestureDetector.SimpleOnGestureListener:

```
TextView mTextView = (TextView) findViewById(R.id.text_id);
```



```

GestureDetector detector = new GestureDetector(this, new Gesture());

// lớp nội Gesture
class Gesture extends GestureDetector.SimpleOnGestureListener {
    @Override public boolean onSingleTapUp(MotionEvent event) {
        mTextView.setText("SimpleOnGestureListener::onSingleTapUp");
        return true;
    }

    @Override public void onLongPress(MotionEvent event) {
        mTextView.setText("SimpleOnGestureListener::onLongPress");
    }

    @Override public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
        mTextView.setText("SimpleOnGestureListener::onScroll");
        return true;
    }

    @Override public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        mTextView.setText("SimpleOnGestureListener::onFling");
        return true;
    }
}

```

GestureDetector có nhiều lớp con để xử lý chi tiết các kiểu gesture khác nhau, như ScaleGestureDetector cho pinch. Bạn phải cài đặt phương thức onScale() của listener để nhận tỷ lệ co giãn do pinch tạo ra:

```

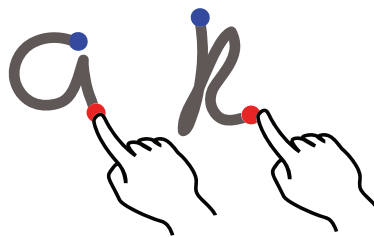
ImageView mImageView;
Matrix matrix = new Matrix();
float scale = 1f;
ScaleGestureDetector sgDetector = new ScaleGestureDetector(this, new ScaleListener());

private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override public boolean onScale(ScaleGestureDetector detector) {
        // tính hệ số co giãn trong đoạn [0.1, 5.0]
        scale *= detector.getScaleFactor();
        scale = Math.max(0.1f, Math.min(scale, 5.0f));
        // co giãn ảnh trong mImageView
        matrix.setScale(scale, scale);
        mImageView.setImageMatrix(matrix);
        return true;
    }
}

public boolean onTouchEvent(MotionEvent event) {
    sgDetector.onTouchEvent(event);
    return true;
}

```

b) Gesture tùy biến



Gesture tùy biến, chữ **a** và chữ **k**

Bạn có thể tạo các gesture tùy biến trong AVD (Android 1.6+) bằng tiện ích "Gestures Builder" trong AVD, bạn cũng có thể tạo một số gesture cùng tên để tăng độ chính xác khi so trùng. Gesture có thể có một nét (stroke) hoặc nhiều nét. Các gesture được tạo sẽ lưu trong tập tin /mnt/sdcard/gestures (phải có SD card) gọi là thư viện gesture. Tài nguyên này có thể sử dụng sau đó trong ứng dụng của bạn:

- Chuyển tập tin gestures vào /res/raw/ để activity của bạn có thể tải vào bằng `GestureLibraries.fromRawResource()`.
- Để nhận dạng gesture, dùng `GestureOverlayView` như một lớp phủ lên các view con chứa trong nó để người dùng vẽ gesture của họ lên. Tạo `GestureOverlayView` bằng code hoặc đặt nó vào layout XML của bạn:

```

<android.gesture.GestureOverlayView
    android:id="@+id/gestures"
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="1"
    android:gestureStrokeType="multiple"

```

```

        android:eventsInterceptionEnabled="true"
        android:orientation="vertical" >
        <ListView android:id="@android:id/list"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </android.gesture.GestureOverlayView >

```

android:gestureStrokeType="multiple" cho phép nhận gesture nhiều nét.

android:eventsInterceptionEnabled="true" cho phép ngăn sự kiện tác động tiếp xuống các view con của lớp phủ.

android:orientation="vertical" cho thấy một nét dọc đơn giản không được xem là gesture vì nhầm lẫn với thao tác cuộn dọc của list nằm bên dưới, vì vậy gesture bắt đầu bằng nét dọc phải có thêm ít nhất một nét ngang.

- Một listener, OnGesturePerformedListener, phải được tạo và đăng ký với GestureOverlayView. Nếu một gesture được phát hiện, phương thức onGesturePerformed() của listener sẽ được gọi. Công việc bạn muốn ứng dụng thực hiện khi nhận được gesture được viết tại đây, tùy theo prediction.name nhận được.

Mặc định, các gesture người dùng thực hiện có màu vàng nhạt (khi chưa so trùng) hoặc vàng tươi (khi bắt đầu so trùng). Có thể tắt bằng setColor(Color.TRANSPARENT) hoặc setUncertainGestureColor(Color.TRANSPARENT) trên GestureOverlayView.

```

public class MainActivity extends ActionBarActivity implements OnGesturePerformedListener {
    private GestureLibrary gestureLibrary;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GestureOverlayView gestureOverlayView = (GestureOverlayView) findViewById(R.id.gesture);
        // hoặc bằng code:
        // View inflate = getLayoutInflater().inflate(R.layout.activity_main, null);
        // GestureOverlayView gestureOverlayView = new GestureOverlayView(this);
        // gestureOverlayView.addView(inflate);
        gestureOverlayView.addOnGesturePerformedListener(this);
        gestureLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
        if (!gestureLibrary.load()) {
            finish();
        }
        setContentView(gestureOverlayView);
    }

    @Override public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture) {
        ArrayList<Prediction> predictions = gestureLibrary.recognize(gesture);
        for (Prediction prediction : predictions) {
            if (prediction.score > 2.0) {
                Toast.makeText(this, prediction.name, Toast.LENGTH_SHORT).show();
            }
        }
    }
}

```

Style và Theme

1. Style

Android cho phép định kiểu tĩnh bằng cách áp dụng tag HTML trong tài nguyên chuỗi:

```
<string name="styledText"><i>Static</i> style in a <b>TextView</b>.</string>
```

hoặc định kiểu động bằng code, xem chuỗi được định kiểu như một đối tượng Spannable:

```

EditText et = (EditText) this.findViewById(R.id.et);
et.setText("Styling the content of an EditText dynamically");
Spannable spn = (Spannable) et.getText();
spn.setSpan(new BackgroundColorSpan(Color.RED), 0, 7, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
spn.setSpan(new StyleSpan(android.graphics.Typeface.BOLD_ITALIC),
    0, 7, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);

```

Ngoài cách định kiểu trực tiếp như trên, Android còn cung cấp cơ chế định kiểu chung để dùng lại kiểu cho mọi view.

Style là một tài nguyên XML, tách biệt với tập tin layout XML, chứa các thuộc tính định kiểu cho view: width, height, padding, font color, font size, ... Các thuộc tính định kiểu của style khá giống với CSS (Cascading Style Sheets) trong web.

Tập tin định kiểu /res/values/style.xml, định kiểu cho một @style/CommandButton sẽ dùng trong tập tin layout XML.

```

<resources>
    <style name="AppTheme" parent="android:Theme.Light" />
    <style name="CommandButton">
        <item name="android:layout_width">0dp</item>
        <item name="android:layout_height">match_parent</item>
        <item name="android:textSize">20dp</item>
        <item name="android:layout_margin">3dp</item>
        <item name="android:background">@drawable/button_shape_shadowed</item>
    </style>

```

```
</resources>
```

Chú ý thuộc tính `parent` của `<style>` cho thấy khả năng thừa kế theme có sẵn. Style con có thể thêm thuộc tính mới hoặc quy định lại thuộc tính của style cha để sử dụng.

Nếu không thừa kế các style của Android, bạn có thể đơn giản khai báo kế thừa style cha đã định nghĩa trước đó như sau:

```
<resources>
    <style name="CommandButton.Danger" >
        <item name="android:textColor" color="#FF0000"/>
    </style>
</resources>
```

Thuộc tính `@drawable/button_shape_shadowed` tham chiếu một *layer-list drawable* và *inset drawable*, định nghĩa trong tập tin `/res/drawable/button_shape_shadowed.xml`:

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item>
        <shape android:shape="rectangle" >
            <corners android:radius="5dp" />
            <gradient android:angle="90"
                android:centerColor="#303339"
                android:centerY="0.05"
                android:endColor="#000000"
                android:startColor="#00000000" />
        </shape>
    </item>
    <item>
        <inset android:drawable="@drawable/button_shape"
            android:insetBottom="5dp" />
    </item>
</layer-list>
```

`@drawable/button_shape` tham chiếu *state list drawable*, định nghĩa trong tập tin `/res/drawable/button_shape.xml`. Hai element `<item>` mô tả hình dạng nút trong hai trạng thái khác nhau, bình thường và khi nhấn vào.

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_shape_normal" android:state_pressed="false"/>
    <item android:drawable="@drawable/button_shape_pressed" android:state_pressed="true"/>
</selector>
```

Các `@drawable` lại tham chiếu các *XML drawable*, định nghĩa trong tập tin `/res/drawable/button_shape_normal.xml`:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >
    <corners android:radius="3dp" />
    <gradient android:angle="90"
        android:endColor="#cccccc"
        android:startColor="#acacac" />
</shape>
```

và trong tập tin `/res/drawable/button_shape_pressed.xml`:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >
    <corners android:radius="3dp" />
    <gradient android:angle="270"
        android:endColor="#cccccc"
        android:startColor="#acacac" />
</shape>
```

Sau khi định kiểu `CommandButton`, bạn có thể áp dụng style cho view trong tập tin layout XML:

```
<Button
    android:id="@+id/enterButton"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    style="@style/CommandButton"
    android:text="Enter" />
```

Chú ý rằng, từng phần của layout có thể định nghĩa riêng, ví dụ trong `/res/layout/button_row.xml`, tạo một hàng có ba nút `CommandButton`.

```
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
    <Button style="@style/CommandButton" />
    <Button style="@style/CommandButton" />
    <Button style="@style/CommandButton" />
</TableRow>
```

rồi "include" nó vào tập tin layout XML, tạo ma trận 3 x 3 các nút được định kiểu:

```
<include android:layout_weight="1" layout="@layout/button_row" />
<include android:layout_weight="1" layout="@layout/button_row" />
<include android:layout_weight="1" layout="@layout/button_row" />
```

2. Theme

Bạn cũng có thể dùng các style được Android cung cấp, lấy từ *theme* của ứng dụng. Theme là một bộ các style, áp dụng cho toàn bộ ứng dụng, không chỉ cho các view cụ thể. Theme định kiểu cho text, button, list, window, dialog, panel, widget, action bar, ... Đôi khi, khái niệm theme và style đôi khi bị dùng lẫn lộn, vì có thể xem theme là một tài nguyên style mà các thuộc tính của nó chỉ đến các tài nguyên style khác.

Android cung cấp nhiều built-in theme để bạn sử dụng:

Theme	Có nền tối và chữ sáng, mặc định chạy được trên mọi phiên bản Android.
Theme.Light	Tùy chỉnh từ Theme, có nền sáng và chữ tối.
Theme.Holo	Theme tương thích với các phiên bản Android kể từ Honeycomb về sau, có nền tối và chữ sáng.
Theme.Holo.Light	Tùy chỉnh từ Theme.Holo, có nền sáng và chữ tối.

Để chỉ định theme tĩnh cho một activity hoặc một ứng dụng, bạn thêm thuộc tính `android:theme` đến các tag tương ứng trong tập tin manifest, ví dụ:

```
<activity android:theme="@style/MyActivityTheme"></activity>
<application android:theme="@style/MyApplicationTheme"></application>
<application android:theme="@android:style/Theme.NoTitleBar"></application>
```

hoặc chỉ định theme động bằng cách code:

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTheme(R.style.MyActivityTheme);
    setContentView(R.layout.activity_main);
}
```

Bạn cũng có thể áp dụng style, lấy từ một theme chỉ định, cho view bằng cách dùng một tham chiếu đến thuộc tính của theme đó, có cú pháp như sau:

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/title" />
```

Giao diện người dùng

Mặc dù mọi ứng dụng Android chạy như một ứng dụng toàn màn hình, Android vẫn cung cấp một số tính năng cửa sổ như action bar, toast, dialog và notification.

Menu

Menu tăng cường chức năng cho ứng dụng bằng cách cung cấp thêm các tác vụ lựa chọn. Android có hai kiểu menu:

- Options menu được kích hoạt bằng cách nhấn nút Menu cứng trên thiết bị. Các thiết bị Android sau này không còn nút Menu cứng này. Trong các phiên bản trước, options menu đặt dưới đáy màn hình. Từ Android 3+, options menu đặt trên Action Bar.
- Context menu là menu tắt, xuất hiện khi nhấn và giữ (tap-and-hold) trên widget có liên kết với menu.

Option menu và context menu có thể chứa: text, icon, radio button, checkbox, submenu và phím tắt.

1. Options menu

Options menu xuất hiện khi nhấn nút Menu cứng của thiết bị Android, liên kết chỉ với activity hiện hành. Options menu có tối đa 6 mục, nếu nhiều hơn sẽ có một mục là More.

Options menu được thiết lập trong tập tin XML và gán đến activity bằng cách viết lại phương thức onCreateOptionsMenu().

Đầu tiên, tạo tập tin menu_main.xml định nghĩa options menu trong /res/menu:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/icon1"
        android:title="One"/>
    <item android:id="@+id/icon2"
        android:title="Two"/>
</menu>
```

Các mục menu có thể kèm theo icon, nhưng từ API Level 11, các mục menu chỉ có text.

Trong lớp Activity, viết lại phương thức onCreateOptionsMenu() để "inflate" menu và viết lại phương thức onOptionsItemSelected() để xử lý sự kiện chọn một mục trong menu.

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
    // "inflate" menu, điều này sẽ thêm các mục đến action bar nếu có.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.icon1:
            Toast.makeText(this, "Icon 1 Beep Bop!", Toast.LENGTH_LONG).show();
            break;
        case R.id.icon2:
            Toast.makeText(this, "Icon 2 Beep Bop!", Toast.LENGTH_LONG).show();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

Khi sử dụng options menu mà không đủ chỗ cho submenu, thường dùng spinner để thay thế. Từ Android 4+, options menu chuyển vào action bar.

2. Context menu

Để hiển thị context menu, tap rồi giữ vài giây trên view có đăng ký với context menu. Context menu sẽ biến mất sau khi được người dùng sử dụng hoặc tap vị trí khác.

Đầu tiên, tạo một tập tin XML /res/menu/context_menu.xml để định nghĩa context menu.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_delete"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="Delete Item" />
    <item android:id="@+id/menu_edit"
        android:icon="@android:drawable/ic_menu_edit"
        android:title="Edit Item" />
</menu>
```

Sau đó viết lại các phương thức onCreateContextMenu() và onContextItemSelected() của Activity để "inflate" menu và xử lý lựa chọn của người dùng.

```
@Override public void onCreateContextMenu(ContextMenu menu, View view,
                                          ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, view, menuInfo);
    getMenuInflater().inflate(R.menu.context_menu, menu);
    menu.setHeaderTitle("Choose an Option");
}

@Override public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_delete:
```



```

        // thực hiện hành động xóa
        break;
    case R.id.menu_edit:
        // thực hiện hành động biên soạn
        break;
    }
    return super.onContextItemSelected(item);
}

```

Để các phương thức callback trên được gọi, bạn phải đăng ký view trên đó context menu sẽ được kích hoạt. Ví dụ sau, đăng ký context menu cho ListView.

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ListView list = new ListView(this);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items);
    list.setAdapter(adapter);
    registerForContextMenu(list);
    setContentView(list);
}

```

Action bar

Android cung cấp action bar để giúp người dùng dễ dàng định danh ứng dụng đang chạy, vị trí bên trong ứng dụng, các hành động quan trọng và các tùy chọn điều hướng. Một action bar chứa:

- Tiêu đề và icon của ứng dụng đang chạy.
- Icon và nhãn của các hành động quan trọng (search, save).
- Ba chấm dọc phía phải action bar chính là options menu được đưa vào từ Android 4+. Hiện tại gọi là overflow menu, như một menu thả xuống, liệt kê các hành động không đủ chỗ hiển thị trên action bar.
- Hỗ trợ điều hướng bên trong ứng dụng thông qua nút Back, các tab và menu thả xuống.

API của action bar bắt đầu từ API Level 11, nhưng cũng được cung cấp thông qua thư viện hỗ trợ (Android Support Library), phần sau sẽ không dùng thư viện này. Từ API Level 11, ActionBar có trong tất cả activity bằng cách dùng theme mặc định Theme.Holo. Ứng dụng có thể truy cập thực thể ActionBar bất cứ lúc nào bằng cách gọi phương thức `getActionBar()` của lớp Activity.

Nếu không cần đến action bar, ứng dụng đơn giản loại nó bằng cách gọi phương thức `hide()` của thực thể ActionBar:

```

@Override protected void onCreate(Bundle savedInstanceState) {
    ActionBar actionBar = getActionBar();
    actionBar.hide();
}

```

1. Thêm hành động vào ActionBar

Action bar cung cấp một vị trí nổi bật để hiển thị các hành động quan trọng của ứng dụng, liên quan với ngữ cảnh hiện hành. Các hành động này hiển thị trên action bar với icon hoặc cả icon lẫn text.

- Trước hết, định nghĩa các hành động như tài nguyên menu trong `/res/menu/`, ví dụ `main.xml`. Mỗi hành động định nghĩa trong element `<item>` với id, icon (`@drawable`) và title (`@string`).

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/action_search"
        android:title="@string/action_search"
        android:icon="@drawable/ic_action_search"
        android:showAsAction="ifRoom|collapseActionView" />
    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:icon="@drawable/ic_action_settings"
        android:showAsAction="never" />
</menu>

```

Ngoài ra, thuộc tính `android:showAsAction` định nghĩa cách hành động sẽ hiển thị trên action bar:

- | | |
|---------------------------------|---|
| <code>ifRoom</code> | Chỉ đặt hành động này trong action bar nếu đủ chỗ; nếu không, đặt nó vào menu tràn. |
| <code>never</code> | Không bao giờ đặt hành động này trong action bar, luôn đặt nó trong menu tràn. |
| <code>withText</code> | Bao gồm tiêu đề văn bản cho mục hành động. |
| <code>always</code> | Luôn đặt hành động này trong action bar, khuyến nghị dùng <code>ifRoom</code> thay thế. |
| <code>collapseActionView</code> | Thu gọn action view thành icon. |

- Viết lại phương thức `onCreateOptionsMenu()` của lớp Activity, trong đó "inflate" tài nguyên menu XML như một thực thể Menu. Android gọi phương thức này khi action bar hiển thị.

```

@Override public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

```

- Viết lại phương thức `onOptionsItemSelected()` của lớp Activity, trong đó xử lý sự kiện tap lên thực thể MenuItem của hành động được chọn. Ứng dụng trích lấy ID của mục được tap bằng phương thức `getItemId()` để định danh hành động được chọn. Nếu sự kiện tap được xử lý, trả về `true` sẽ được trả về.

```

@Override public boolean onOptionsItemSelected(MenuItem item) {

```

```

boolean consumed = true;
switch (item.getItemId()) {
    case R.id.action_search:
        break;
    case R.id.action_settings:
        break;
    default:
        consumed = super.onOptionsItemSelected(item);
}
return consumed;
}

```

Bạn cũng có thể dùng thuộc tính `android:onClick` của `<item>` trong tài nguyên menu XML để chỉ định phương thức callback, xử lý sự kiện tap lên một hành động cụ thể:

```

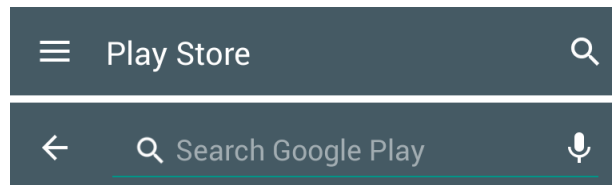
<item android:id="@+id/action_save"
    android:title="@string/action_save"
    android:icon="@drawable/ic_action_save"
    android:showAsAction="ifRoom"
    android:onClick="save"/>

```

Phương thức void `save(MenuItem item)` phải được cài đặt trong lớp Activity và sẽ được triệu gọi để xử lý sự kiện.

2. Action view

Để cung cấp truy cập nhanh hơn đến một số hành động, Android cung cấp action view. Khi tap lên icon của hành động đó trên action bar, một widget sẽ xuất hiện trên action bar thay thế cho icon của hành động, cho phép truy cập trực tiếp đến hành động. `SearchView` là một ví dụ điển hình của action view, `SearchView` cho phép người dùng bắt đầu hành động tìm kiếm ngay lập tức trên action bar.



- Action view được khai báo thông qua thuộc tính `android:actionViewClass` của `<item>` trong tài nguyên menu XML.

```

<item android:id="@+id/action_search"
    android:title="@string/action_search"
    android:icon="@drawable/ic_action_search"
    android:showAsAction="ifRoom"
    android:actionViewClass="android.widget.SearchView"/>

```

Action view có thể là một View hoặc một layout. Nếu action view là layout, thuộc tính `android:actionLayout` sẽ được dùng thay thế cho `android:actionViewClass`.

- Sau khi "inflate" tài nguyên menu XML, ứng dụng có thể truy cập đến thực thể action view bên trong phương thức `onOptionsItemSelected()`:

```

@Override public boolean onOptionsItemSelected(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    MenuItem menuItem = menu.findItem(R.id.action_search);
    SearchView searchView = (SearchView) menuItem.getActionView();
    return true;
}

```

Ví dụ dùng fragment, activity chính sẽ thêm fragment sau:

```

public class ActionBarFragment extends ListFragment implements
    TextView.OnEditorActionListener, SearchView.OnQueryTextListener, SearchView.OnCloseListener {
    private static final String STATE_QUERY = "q";
    private static final String STATE_MODEL = "m";
    private static final String[] items = {
        "apple", "apricot", "avocado", "banana", "blackberry", "blueberry", "cantaloupe", "cherry",
        "coconut", "durian", "fig", "grape", "agrapefruit", "guava", "honeydew", "jackfruit", "kiwi",
        "kumquat", "lemon", "lime", "longan", "lychee", "mandarin", "mango", "mangosteen" };
    private ArrayList<String> fruits = null;
    private ArrayAdapter<String> adapter = null;
    private CharSequence initialQuery = null;
    private SearchView searchView = null;

    @Override public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (savedInstanceState == null) {
            initAdapter(null);
        } else {
            initAdapter(savedInstanceState.getStringArrayList(STATE_MODEL));
            initialQuery = savedInstanceState.getCharSequence(STATE_QUERY);
        }
    }
}

```

```

        setHasOptionsMenu(true);
    }

    @Override public void onSaveInstanceState(Bundle state) {
        super.onSaveInstanceState(state);
        if (!searchView.isIconified()) {
            state.putCharSequence(STATE_QUERY, searchView.getQuery());
        }
        state.putStringArrayList(STATE_MODEL, fruits);
    }

    @Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        inflater.inflate(R.menu.main, menu);
        configureSearchView(menu);
        super.onCreateOptionsMenu(menu, inflater);
    }

    // TextView.OnEditorActionListener
    @Override public boolean onEditorAction(TextView view, int actionId, KeyEvent event) {
        if (event == null || event.getAction() == KeyEvent.ACTION_UP) {
            adapter.add(view.getText().toString());
            view.setText("");
            InputMethodManager imm =
                (InputMethodManager) getActivity().getSystemService(Context.INPUT_METHOD_SERVICE);
            imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
        }
        return true;
    }

    // SearchView.OnQueryTextListener
    @Override public boolean onQueryTextChange(String newText) {
        if (TextUtils.isEmpty(newText)) {
            adapter.getFilter().filter("");
        } else {
            adapter.getFilter().filter(newText.toString());
        }
        return true;
    }

    @Override public boolean onQueryTextSubmit(String query) {
        return false;
    }

    // SearchView.OnCloseListener
    @Override public boolean onClose() {
        adapter.getFilter().filter("");
        return true;
    }

    @Override public void onListItemClick(ListView l, View v, int position, long id) {
        Toast.makeText(getActivity(), adapter.getItem(position), Toast.LENGTH_LONG).show();
    }

    private void configureSearchView(Menu menu) {
        MenuItem search = menu.findItem(R.id.action_search);
        searchView = (SearchView) search.getActionView();
        searchView.setOnQueryTextListener(this);
        searchView.setOnCloseListener(this);
        searchView.setSubmitButtonEnabled(false);
        searchView.setIconifiedByDefault(true);
        if (initialQuery != null) {
            searchView.setIconified(false);
            search.expandActionView();
            searchView.setQuery(initialQuery, true);
        }
    }

    private void initAdapter(ArrayList<String> startingPoint) {
        if (startingPoint == null) {
            fruits = new ArrayList<String>();
            fruits.addAll(Arrays.asList(items));
        } else {
            fruits = startingPoint;
        }
    }

```

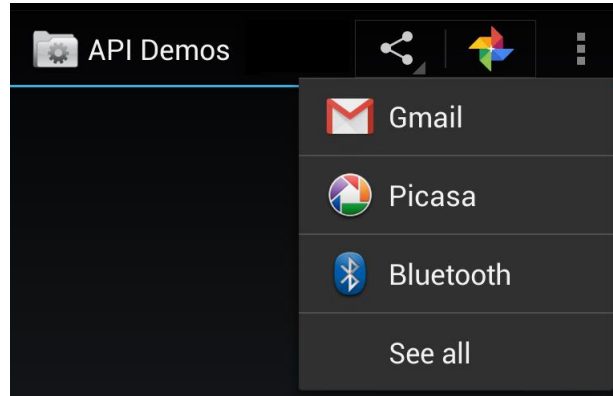
```

    }
    adapter = new ArrayAdapter<String>(getActivity(), android.R.layout.simple_list_item_1, fruits);
    setListAdapter(adapter);
}
}

```

3. Action provider

Mặc dù action view cho phép thay thế icon của hành động với widget, nhưng nó không xử lý tự động hành động mà widget phải thực hiện. Android cung cấp action provider, ngoài việc thay thế icon của hành động với layout tùy biến, nó cung cấp xử lý cho tất cả hành động. ShareActionProvider là một ví dụ điển hình của action provider. Menu thả xuống của nó hiển thị các cách có thể để chia sẻ nội dung.



- Action provider cũng được khai báo thông qua thuộc tính `android:actionProviderClass` của `<item>` trong tài nguyên menu XML.

```

<item android:id="@+id/action_share"
      android:title="@string/action_share"
      android:showAsAction="ifRoom"
      android:actionProviderClass="android.widget.ShareActionProvider"/>

```

Khi action provider cần thực hiện một tác vụ nào đó, nó có thể yêu cầu một số thông tin bổ sung từ ứng dụng. Ứng dụng khởi động action provider từ bên trong phương thức `onCreateOptionsMenu()` của lớp `Activity`:

```

@Override public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_toast, menu);
    MenuItem menuItem = menu.findItem(R.id.action_share);
    ShareActionProvider shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("image/*");
    shareActionProvider.setShareIntent(intent);
    return true;
}

```

Toast

1. Toast

Với ứng dụng chạy bề mặt, Toast cung cấp một cách tiện dụng để cảnh báo cho người dùng, cảnh báo này chỉ xuất hiện tạm thời vài giây, nhấp nháy trên màn hình rồi biến mất, không nhận focus. Tham số thứ nhất là context, tham số thứ hai là thông điệp, tham số thứ ba là thời gian hiển thị thông điệp, định nghĩa như hằng của lớp Toast: `LENGTH_SHORT` hoặc `LENGTH_LONG`.

```

Toast toast = Toast.makeText(this, "Hello Word!", Toast.LENGTH_LONG);
toast.show();
// thường viết tắt như sau, chú ý đừng quên gọi show()
Toast.makeText(this, "Hello Word!", Toast.LENGTH_LONG).show();

```

Mặc định, thông báo của Toast chỉ hiển thị tại vị trí dưới-giữa của màn hình. Tuy nhiên, bạn có thể thay đổi vị trí hiển thị của Toast bằng phương thức sau:

```
void setGravity(int gravity, int xOffset, int yOffset)
```

Tham số gravity, thiết lập vị trí tương đối theo chiều ngang và chiều dọc. Các trị offset tính bằng pixel, dùng kết hợp với gravity. Ví dụ: cạnh dưới Toast cách biên dưới 10 pixel, cạnh phải cách biên phải 20 pixel.

```
toast.setGravity(Gravity.BOTTOM|Gravity.RIGHT, 10, 20);
```

2. Tùy biến Toast

Bạn cũng có thể tạo một Toast tùy biến cho ứng dụng của bạn:

- Định nghĩa hình dạng, màu nền của Toast trong drawable/`toast_background.xml`

```

<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle" >
    <stroke android:width="2dp" android:color="#507DE0" />
    <solid android:color="#1F3157" />
    <corners android:radius="8dp" />

```

```
<padding android:left="10dp" android:top="8dp" android:right="10dp" android:bottom="8dp" />
</shape>
```

- Định nghĩa bố trí của Toast trong layout/`toast_layout.xml`, layout này có tham chiếu tập tin XML trên. Bạn có thể dùng `ImageView` thêm icon vào Toast.

```
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:id="@+id/toast_layout_root"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/txtToast"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="@drawable/toast_background"
        android:textColor="#FFFFFF" />
</LinearLayout>
```

- Dùng `setView()` để tạo Toast tùy biến:

```
private void showCustomToast(String message) {
    Toast toast = new Toast(getApplicationContext());
    View layout = getLayoutInflater().inflate(R.layout.toast_layout,
        (ViewGroup) findViewById(R.id.toast_layout_root));

    TextView txtToast = (TextView) layout.findViewById(R.id.txtToast);
    txtToast.setText(message);
    toast.setGravity(Gravity.NO_GRAVITY, 0, 0);
    toast.setDuration	Toast.LENGTH_SHORT);
    toast.setView(layout);
    toast.show();
}

// hiển thị Toast tùy biến
showCustomToast("Hello World!");
```

Dialog

1. AlertDialog

Trong ứng dụng, đôi khi bạn muốn người dùng xác nhận sự chấp nhận của họ, hoặc yêu cầu thực hiện một số lựa chọn, cung cấp một vài thông tin; nhưng vẫn giữ lại activity đang hoạt động và không muốn thay đổi màn hình. Khi đó, bạn có thể dùng `AlertDialog`, một loại hộp thoại *modal* (hộp thoại chờ tương tác, ngăn tương tác với giao diện phía dưới nó).

Phương thức `onCreateDialog()` của activity trả về một `Dialog`, trong phương thức này `AlertDialog` (lớp con của `Dialog`) được tạo. Với `AlertDialog`, bạn có thể cung cấp cho người dùng ba tùy chọn, tương ứng với ba nút có thể được dùng trong bất kỳ kịch bản nào:

- Positive reaction: thực hiện công việc được yêu cầu của activity.
- Neutral reaction: không thực hiện công việc, đóng activity. Thường không cần thiết lập tùy chọn này.
- Negative reaction: không thực hiện công việc được yêu cầu, quay trở lại activity.

Ngoài ba nút, hộp thoại cho phép chọn nhiều mục (multi choice) bằng checkbox, hoặc chọn chỉ một mục (single choice) bằng radio button. Nút, checkbox và radio button được lắng nghe nhờ các đối tượng listener của `DialogInterface`.

Trước tiên, bạn cần tạo một đối tượng `AlertDialog.Builder`, là lớp nội của `AlertDialog`:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

Sau đó, bạn dùng các phương thức của đối tượng builder để tùy biến cho hộp thoại, thiết lập nút (positive, neutral và negative), thiết lập cách chọn mục:

```
setIcon(Drawable icon)
```

Thiết lập icon của hộp thoại.

```
setCancelable(boolean cancelable)
```

Thiết lập các thuộc tính cho hộp thoại có thể thoát hoặc không.

```
setMessage(CharSequence message)
```

Thiết lập thông điệp hiển thị bên trong hộp thoại.

```
setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems,
    DialogInterface.OnMultiChoiceClickListener listener)
```

Thiết lập danh sách các mục kèm checkbox để chọn nhiều mục. Các mục chọn được sẽ được báo bởi listener.

```
setSingleChoiceItems(CharSequence[] items, int checkedItem, DialogInterface.OnClickListener listener)
```

Thiết lập danh sách các mục kèm radio button để chọn một mục. Mục chọn được sẽ được báo bởi listener.

```
setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)
```

Thiết lập callback sẽ được gọi nếu hộp thoại thoát.

```
setTitle(CharSequence title)
```

Thiết lập tiêu đề của hộp thoại.

Sau khi tạo và cấu hình đối tượng builder, bạn tạo một hộp thoại bằng cách gọi phương thức `create()` của builder. Nếu hộp thoại không được gọi từ `onCreateDialog()`, hiển thị nó bằng cách gọi phương thức `show()`:

```
alertDialogBuilder.create().show();
```

Ví dụ minh họa:


```

public class DialogActivity extends Activity {
    CharSequence[] items= { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked= new boolean[items.length];

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
    }

    public void onClick(View v) { // xử lý nút nhấn, gọi từ tập tin layout XML
        showDialog(0);
    }

    @Override protected Dialog onCreateDialog(int id) {
        switch (id) {
            case 0:
                return new AlertDialog.Builder(this)
                    .setIcon(R.mipmap.ic_launcher)
                    .setTitle("Favourite company...")
                    .setPositiveButton("OK",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int whichButton) {
                                Toast.makeText(getBaseContext(), "OK clicked!", Toast.LENGTH_SHORT).show();
                            }
                        }
                    )
                    .setNegativeButton("Cancel",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int whichButton) {
                                Toast.makeText(getBaseContext(), "Cancel clicked!", Toast.LENGTH_SHORT).show();
                            }
                        }
                    )
                    .setMultiChoiceItems(items, itemsChecked,
                        new DialogInterface.OnMultiChoiceClickListener() {
                            public void onClick(DialogInterface dialog, int which, boolean isChecked) {
                                Toast.makeText(getBaseContext(),
                                    items[which] + (isChecked ? " checked!":" unchecked!"),
                                    Toast.LENGTH_SHORT).show();
                            }
                        }
                    )
                    .create();
            // switch
            return null;
        }
    }
}

```

Trong ví dụ trên, khi gọi phương thức `showDialog(int)`, phương thức callback `onCreateDialog()` sẽ được gọi. Từ API Level 13, phương thức `showDialog(int)` đã lạc hậu, thay thế bằng phương thức `show()` của `DialogFragment`:

```

public void onClick(View v) {
    DialogFragment fragment = ChoiceDialogFragment.newInstance();
    fragment.show(getFragmentManager(), "dialog");
}

public static class ChoiceDialogFragment extends DialogFragment {
    public static ChoiceDialogFragment newInstance() {
        return new ChoiceDialogFragment();
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // tạo AlertDialog như trên
    }
}

```

Ngoài việc hỗ trợ ba nút, `AlertDialog` cũng chứa các đơn thể nhập phức tạp. Bạn có thể đưa vào `AlertDialog`:

- Một list:

```

final String[] items = { "Obama", "Putin", "Holland", "Castro" };
AlertDialog alertDialog = new AlertDialog.Builder(this)
    .setTitle("Dialog Title")
    .setItems(items, new DialogInterface.OnClickListener() {
        @Override public void onClick(DialogInterface dialogInterface, int what) {

```

```
String item = items[what];
}
}).create();
```

- Một multi-choice list:

```
final String[] items = { "Obama", "Putin", "Holland", "Castro" };
final boolean[] checked = { true, false, false, false };
AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this)
    .setTitle("Dialog Title")
    .setMultiChoiceItems(items, checked, new DialogInterface.OnMultiChoiceClickListener() {
        @Override public void onClick(DialogInterface dialogInterface, int what, boolean isChecked) {
        }
    })
    .create();
```

- Một single-choice list:

```
final String[] items = { "Obama", "Putin", "Holland", "Castro" };
final boolean[] checked = { true, false, false, false };
AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this)
    .setTitle("Dialog Title")
    .setSingleChoiceItems(items, 1, new DialogInterface.OnClickListener() {
        @Override public void onClick(DialogInterface dialogInterface, int what) {
        }
    })
    .create();
```

- Một layout tùy biến: ví dụ rating bar "inflate" tập tin layout XML custom_dialog.xml

```
LayoutInflater inflater = getLayoutInflater();
View customDialog = inflater.inflate(R.layout.custom_dialog, null);
final RatingBar ratingBar = (RatingBar) customDialog.findViewById(R.id.ratingBar);
final AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this)
    .setTitle("Dialog Title")
    .setView(customDialog)
    .setPositiveButton("OK", new DialogInterface.OnClickListener() {
        @Override public void onClick(DialogInterface dialogInterface, int i) {
            int numberOfStars = ratingBar.getNumStars();
        }
    })
    .create();
```

2. Date/Time

Android hỗ trợ các widget (DatePicker, TimePicker) và các hộp thoại (DatePickerDialog, TimePickerDialog) giúp người dùng nhập ngày/tháng/năm và thời điểm.

- DatePicker và DatePickerDialog cho phép chọn ngày/tháng/năm. Trị tháng bắt đầu từ 0 (cho tháng 1) đến 11 (cho tháng 12). Mỗi widget cung cấp một đối tượng callback (OnDateChangeListener hoặc OnDateSetListener) sẽ báo một ngày mới được chọn bởi người dùng.

- TimePicker và TimePickerDialog cho phép chọn giờ (0 – 23) và phút (0 – 59), chỉ định chế độ 12-hour (AM/PM) hoặc chế độ 24-hour. Mỗi widget cung cấp một đối tượng callback (OnTimeChangeListener hoặc OnTimeSetListener) sẽ báo một thời điểm mới được chọn bởi người dùng.

```
EditText selectDate;
EditText selectTime;
// các phương thức callback
DatePickerDialog.OnDateSetListener dateListener = new DatePickerDialog.OnDateSetListener() {
    public void onDateSet(DatePicker view, int year, int month, int day) {
        selectDate.setText(day + "/" + (month + 1) + "/" + year);
    }
};

TimePickerDialog.OnTimeSetListener timeListener = new TimePickerDialog.OnTimeSetListener() {
    public void onTimeSet(TimePicker view, int hour, int minute) {
        selectTime.setText(hour + ":" + minute);
    }
};
// xử lý các nút trong onCreate của activity
final Calendar c = Calendar.getInstance();
Button btnDate = (Button) findViewById(R.id.btnDatePicker);
btnDate.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        new DatePickerDialog(MainActivity.this, dateListener,
            c.get(Calendar.YEAR), c.get(Calendar.MONTH), c.get(Calendar.DAY_OF_MONTH)).show();
    }
});
Button btnTime = (Button) findViewById(R.id.btnTimePicker);
btnTime.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
```

```

        new TimePickerDialog(MainActivity.this, timeListener,
            c.get(Calendar.HOUR_OF_DAY), c.get(Calendar.MINUTE), true).show();
    }
});

```

Tương tự, nhưng dùng fragment:

```

EditText selectDate;
EditText selectTime;
// Các fragment dùng như lớp nội của activity để dễ thay đổi các EditText của activity
class DatePickerFragment extends DialogFragment
    implements DatePickerDialog.OnDateSetListener {
    @Override public Dialog onCreateDialog(Bundle savedInstanceState) {
        // dùng ngày hiện tại làm ngày mặc định của picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
        // tạo một thực thể mới của DatePickerDialog rồi trả về
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    @Override public void onDateSet(DatePicker view, int year, int month, int day) {
        // cập nhật UI với ngày được chọn
        selectDate.setText(day + "/" + (month + 1) + "/" + year);
    }
}

class TimePickerFragment extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener {
    @Override public Dialog onCreateDialog(Bundle savedInstanceState) {
        // dùng thời điểm hiện tại làm thời điểm mặc định của picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR);
        int minute = c.get(Calendar.MINUTE);
        // tạo một thực thể mới của TimePickerDialog rồi trả về
        return TimePickerDialog(getActivity(), this, hour, minute, DateFormat.is24HourFormat(getActivity()));
    }

    @Override public void onTimeSet(TimePicker view, int hour, int minute) {
        // cập nhật UI với thời điểm được chọn
        selectTime.setText(hour + ":" + minute);
    }
}

public void showDatePickerDialog(View v) {
    new DatePickerFragment().show(this.getFragmentManager(), "datePicker");
}

public void showTimePickerDialog(View v) {
    new TimePickerFragment().show(this.getFragmentManager(), "timePicker");
}

```

Android cũng cung cấp các widget DigitalClock và AnalogClock, có hình ảnh đồng hồ số và đồng hồ kim, các widget này tự động cập nhật thời gian.

3. ProgressDialog

Bạn nên hiển thị hộp thoại ProgressDialog khi thực hiện một tác vụ chạy nền chiếm nhiều thời gian, nội dung hộp thoại đơn giản yêu cầu người dùng chờ đợi hoặc cung cấp thêm thông tin về quá trình thực hiện tác vụ chạy nền.

Trong ví dụ minh họa sau, tạo hai nút gọi các phương thức xử lý nút nhấn từ tập tin layout XML: launchRingDialog() hiển thị ProgressDialog đơn giản và launchBarDialog() hiển thị ProgressDialog với thông tin phức tạp. Phương thức thứ hai cập nhật UI thread bằng cách dùng post() (xem phần Tác vụ chạy nền).

```

public class MainActivity extends Activity {
    ProgressDialog bar;
    Handler updateBarHandler = new Handler();

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void launchRingDialog(View view) {
        final ProgressDialog ring = ProgressDialog

```

```

        .show(MainActivity.this, "Please wait...", "Downloading Image...", true);
ring.setCancelable(true);
new Thread(new Runnable() {
    @Override public void run() {
        try {
            // thực hiện tác vụ chiếm thời gian
            Thread.sleep(10000);
        } catch (Exception e) { }
        ring.dismiss();
    }
}).start();
}

public void launchBarDialog(View view) {
    bar = new ProgressDialog(MainActivity.this);
    bar.setTitle("Downloading Image ...");
    bar.setMessage("Download in progress ...");
    bar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    bar.setProgress(0);
    bar.setMax(20);
    bar.show();
    new Thread(new Runnable() {
        @Override public void run() {
            try {
                while (barProgressDialog.getProgress() <= barProgressDialog.getMax()) {
                    // thực hiện tác vụ chiếm thời gian
                    Thread.sleep(2000);
                    updateBarHandler.post(new Runnable() {
                        @Override public void run() {
                            bar.incrementProgressBy(2);
                        }
                    });
                    if (bar.getProgress() == barProgressDialog.getMax()) {
                        bar.dismiss();
                    }
                } // while
            } catch (Exception e) { }
        }
    }).start();
}
}

```

Notification

Nếu cần cảnh báo bền vững hơn, bạn dùng notification. Notification là một thông điệp được hiển thị như một icon bé trên thanh cảnh báo (notification bar) hay thanh trạng thái (status bar), trên phải với Gingerbread và dưới phải với Honeycomb. Để nhìn thấy notification, chọn icon bé thể hiện notification và kéo trượt xuống (drawer) để thấy nội dung chi tiết như hình dưới. Nội dung của notification có thể trình bày dưới dạng thường (normal view) hoặc dạng lớn (big view) có nhiều dòng hơn. Notification có thể tùy chọn kèm theo đèn LED nhấp nháy, âm thanh hoặc thiết bị rung báo.



1. Tạo và gửi notification

Theo các bước sau:

- Tạo một đối tượng `Notification.Builder` và thiết lập các thuộc tính cho notification bằng cách gọi các phương thức của đối tượng `Notification.Builder`, tùy theo yêu cầu của bạn, tuy nhiên tối thiểu nên có:

- + Một icon bé, thiết lập bởi `setSmallIcon()`.
- + Một tiêu đề (title), thiết lập bởi `setContentTitle()`.
- + Một thông điệp (message) văn bản chi tiết, thiết lập bởi `setContentText()`.

Bạn cũng có thể thêm vào notification đến ba nút. Khi notification thu gọn, các nút này không hiển thị và ứng dụng cung cấp một hành động mặc định cho trường hợp này.

Sau đó, gọi phương thức `build()` của `Notification.Builder` để trả về thực thể `Notification`.

```

Notification notification = new Notification.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("You got mail")
    .setContentText("karlmarx@gmail.com")
    .addAction(R.drawable.ic_action_reply, "Reply", replyPendingIntent)
    .addAction(R.drawable.ic_action_discard, "Discard", discardPendingIntent)

```

```
.setDefaults(Notification.DEFAULT_VIBRATE | Notification.DEFAULT_SOUND | Notification.FLAG_SHOW_LIGHTS)
.setLights(Color.RED, 1, 1)
.build();
```

- Phát hành notification

Gọi `NotificationManager.notify()` để gửi notification của bạn. Phương thức này cần một định danh cho notification.

```
NotificationManager manager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
manager.notify(1, notification);
```

- Gán hành động chờ sẵn

Đây là phần tùy chọn, cần khi bạn muốn gán một hành động chờ sẵn cho notification. Một hành động chờ sẵn cho phép người dùng nhảy trực tiếp từ notification ngược trở lại activity nào đó trong ứng dụng của bạn.

Hành động chờ sẵn được định nghĩa bởi đối tượng `PendingIntent`, sẽ khởi động Activity nào đó trong ứng dụng của bạn. Để liên kết `PendingIntent` với một thao tác, gọi phương thức `setContentIntent()` của `Notification.Builder`.

```
Intent intent = new Intent(this, MessageActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
Notification notification = new Notification.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("You got mail")
    .setContentText("karlmarx@gmail.com")
    .setContentIntent(pendingIntent)
    .build();
```

Hoạt động của notification làm ảnh hưởng đến back stack. Ví dụ, một ứng dụng email thông thường sẽ có một activity `Inbox` để liệt kê các mail, và một activity `Message` để hiển thị nội dung của từng mail. Trong trường hợp notification báo có mail mới, người dùng nhảy trực tiếp đến activity `Message` mà không phải thông qua activity `Inbox`. Khi đó ứng dụng tạo một back stack mới, bằng cách dùng `TaskStackBuilder`.

```
Intent intent = new Intent(this, ToastActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(MessageActivity.class);
stackBuilder.addNextIntent(intent);
PendingIntent pendingIntent = stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
```

Phương thức `addParentStack()` thử xây dựng tự động back stack dựa trên activity cha của activity chỉ định. Thuộc tính `android:parentActivityName` của `<activity>` cho biết activity cha.

```
<activity android:name=".InboxActivity">
</activity>
<activity android:name=".MessageActivity"
    android:parentActivityName=".InboxActivity">
</activity>
```

Khi notification đã hiển thị, ứng dụng vẫn có thể cập nhật nó. Bằng cách dùng cùng định danh, ứng dụng có thể gửi một notification mới thay thế cho cái trước đó. Ví dụ sau dùng `Notification.InboxStyle`, một loại notification lớn và nhiều dòng, để gom các notification thành một notification.

```
Notification.InboxStyle inboxStyle = new Notification.InboxStyle();
inboxStyle.addLine("Fw: Information you have requested");
inboxStyle.setSummaryText("+2 more");
notification = new Notification.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("You have 3 New Messages")
    .setContentText("karlmarx@gmail.com")
    .setContentIntent(pendingIntent)
    .setStyle(inboxStyle)
    .build();
manager.notify(1, notification);
```

Định danh của notification cũng dùng để hủy nó:

```
manager.cancel(1);
```

2. Lớp Notification.Builder

`Notification.Builder` cho phép dễ dàng điều khiển các cờ, cũng như giúp xây dựng các layout cảnh báo điển hình. Các phương thức thường dùng:

`Notification build()`

Kết hợp tất cả các tùy chọn đã được thiết lập và trả về một đối tượng `Notification` mới.

`Notification.Builder setAutoCancel(boolean autoCancel)`

Thiết lập cờ này để notification tự thoát khi người dùng tap lên panel.

`Notification.Builder setContent(RemoteViews views)`

Cung cấp một `RemoteViews` tùy chỉnh thay vì dùng cái chuẩn.

`Notification.Builder setContentInfo(CharSequence info)`

Thiết lập văn bản lớn tại bên phải của notification.

`Notification.Builder setContentIntent(PendingIntent intent)`

Hỗ trợ một `PendingIntent` để gửi khi tap lên notification.

`Notification.Builder setContentText(CharSequence text)`

Thiết lập văn bản (dòng thứ hai) của notification, trong một notification chuẩn.

`Notification.Builder setTitle(CharSequence title)`
Thiết lập văn bản (dòng thứ nhất) của notification, trong một notification chuẩn.

`Notification.Builder setDefaults(int defaults)`
Thiết lập các tùy chọn notification mặc định sẽ được dùng.

`Notification.Builder setLargeIcon(Bitmap icon)`
Thiết lập icon lớn trình bày trong notification.

`Notification.Builder setNumber(int number)`
Thiết lập số lớn tại bên phải của notification.

`Notification.Builder setOngoing(boolean ongoing)`
Thiết lập đây là một thông báo đang diễn ra.

`Notification.Builder setSmallIcon(int icon)`
Thiết lập icon nhỏ trình bày trong notification.

`Notification.Builder setStyle(Notification.Style style)`
Thêm một kiểu notification phong phú để áp dụng tại thời gian build.

`Notification.Builder setTicker(CharSequence tickerText)`
Thiết lập văn bản sẽ hiển thị trong thanh trạng thái khi notification xuất hiện lần đầu.

`Notification.Builder setVibrate(long[] pattern)`
Thiết lập một mẫu rung báo để dùng.

`Notification.Builder setWhen(long when)`
Thiết lập thời gian để sự kiện xảy ra. Notification trong panel được sắp xếp theo thời điểm này.

Transition

Các version Android trước 4.4 dùng hiệu ứng hình ảnh động thay cho transition.

Ví dụ, tạo chuyển tiếp giữa hai activity. Đầu tiên, tạo 4 tập tin mô tả hiệu ứng hình ảnh động: trong Android Studio tạo một Android resource file, chọn Resource type là Animation, Root element là set và Directory name là anim.

activity_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate android:fromXDelta="100%" android:toXDelta="0" android:duration="400" />
  <alpha android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="400" />
</set>
```

activity_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate android:fromXDelta="0" android:toXDelta="-100%" android:duration="200" />
  <alpha android:fromAlpha="1.0" android:toAlpha="0.0" android:duration="200" />
</set>
```

activity_in_back.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate android:fromXDelta="-100" android:toXDelta="0" android:duration="200" />
  <alpha android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="200" />
</set>
```

activity_out_back.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate android:fromXDelta="0" android:toXDelta="100%" android:duration="200" />
  <alpha android:fromAlpha="1.0" android:toAlpha="0.0" android:duration="200" />
</set>
```

Trong MainActivity, dùng phương thức `Activity.overridePendingTransition()` gọi ngay sau `startActivity()`:

```
public class MainActivity extends ActionBarActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.button_id).setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                startActivity(new Intent(MainActivity.this, TwoActivity.class));
                overridePendingTransition(R.anim.activity_in, R.anim.activity_out);
            }
        });
    }
}
```

Trong TwoActivity, trong phương thức `onBackPressed()` tạo hiệu ứng hình ảnh động ngược lại khi nhấn nút Back.

```
@Override public void onBackPressed() {
    super.onBackPressed();
    overridePendingTransition(R.anim.activity_in_back, R.anim.activity_out_back);
}
```

Transition framework được giới thiệu từ Android 4.4 và tiếp tục phát triển trong Android 5. Transition (chuyển tiếp) dùng để thêm hiệu ứng chuyển tiếp cho các view trong thời gian chạy, như fade, trượt, thay đổi kích thước, di chuyển.

Transition cho phép thay đổi layout và sự xuất hiện các view trong giao diện bằng nhiều cách khác nhau, với sự tham gia của nhiều đối tượng khác nhau.

- Scene

Một scene (cảnh) thể hiện toàn bộ layout của một màn hình hoặc một layout con (thể hiện bởi ViewGroup). Để thực hiện hiệu ứng chuyển tiếp, scene thường được định nghĩa hai trạng thái: trước và sau. Sự thay đổi giữa hai scene có thể kèm hiệu ứng hình ảnh động tạo nên trải nghiệm chuyển tiếp trơn mượt cho người dùng.

- Transition

Hiệu ứng hình ảnh động được xử lý thông qua các lớp thừa kế Transition, như Fade, ChangeBounds. Mỗi transition cho phép thiết lập các thuộc tính dễ dàng nhờ giao diện fluent của nó. Nhiều hiệu ứng chuyển tiếp có thể gom trong một TransitionSet (lớp con của Transition) và được cấu hình để thực hiện song song hoặc tuần tự.

- TransitionManager

Lớp TransitionManager có nhiều phương thức static để tạo ra một chuyển tiếp đơn giản:

+ beginDelayedTransition() có tham số thứ nhất là tham chiếu đến scene gốc, tham số thứ hai tùy chọn là Transition hoặc TransitionSet mô tả chuyển tiếp.

+ go() có tham số thứ nhất là tham chiếu đến scene đích, tham số thứ hai là Transition hoặc TransitionSet mô tả chuyển tiếp.

- Interpolator (bộ nội suy) nội suy các hình ảnh chuyển tiếp theo cách được định nghĩa trước để thực hiện tùy biến chuyển tiếp hình ảnh động. Một số interpolator được cung cấp: AccelerateInterpolator, AccelerateDecelerateInterpolator, AnticipateInterpolator, AnticipateOvershootInterpolator, BounceInterpolator, CycleInterpolator, DecelerateInterpolator, LinearInterpolator và OvershootInterpolator.

Scene, Transition, TransitionSet và các interpolator được tạo bằng code hoặc khai báo trong tập tin layout XML.

Ví dụ sau minh họa vai trò của Scene, TransitionSet, TransitionManager và Interpolator.

Scene thứ nhất có layout scene_root.xml. Scene thứ hai có layout scene_dest.xml, thường có cùng các view như trong scene thứ nhất nhưng có vị trí khác.

```
public class MainActivity extends ActionBarActivity {
    Scene mSceneDest;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.scene_root);
        ViewGroup mSceneRoot = (ViewGroup) findViewById(R.id.scene_root);
        mSceneDest = Scene.getSceneForLayout(mSceneRoot, R.layout.scene_dest, this);
        findViewById(R.id.button_id).setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                TransitionSet set = new TransitionSet()
                    .addTransition(new Fade())
                    .addTransition(new ChangeBounds())
                    .setOrdering(TransitionSet.ORDERING_TOGETHER)
                    .setDuration(2000)
                    .setInterpolator(new AccelerateInterpolator());
                TransitionManager.go(mSceneDest, set);
            }
        });
    }
}
```

Bạn có thể định nghĩa transition tương tự tập transition trên, trong tập tin transition.xml: trong Android Studio tạo một Android resource file, chọn Resource type là Transition, Root element là transitionSet và Directory name là transition.

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android"
    android:transitionOrdering="together"
    android:duration="2000"
    android:interpolator="@android:interpolator/accelerate_decelerate">
    <fade android:fadingMode="fade_in">
        <targets><target android:targetId="@id/textView" /></targets>
    </fade>
    <changeBounds>
        <targets><target android:targetId="@id/transition_square" /></targets>
    </changeBounds>
</transitionSet>
```

dùng transition/transition.xml trong code để thực hiện một chuyển tiếp:

```
public class MainActivity extends ActionBarActivity {
    Scene mSceneDest;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.scene_root);
        ViewGroup mSceneRoot = (ViewGroup) findViewById(R.id.scene_root);
        mSceneDest = Scene.getSceneForLayout(mSceneRoot, R.layout.scene_dest, this);
        findViewById(R.id.button_id).setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                Transition transition = TransitionInflater.from(getBaseContext())
                    .inflateTransition(R.transition.transition);
                TransitionManager.go(mSceneDest, transition);
            }
        });
    }
}
```

```

    });
}
}

```

Localization

Thay vì thiết lập cứng trị các chuỗi và media trong ứng dụng, ta thường nạp chúng từ thư mục /res của project. Đây là tiền đề cho khả năng bản địa hóa ứng dụng Android.

Để có một ứng dụng Android đa ngôn ngữ, bạn cần cung cấp cho Android các tập tin tài nguyên bản địa tương ứng. Khi Android tìm trị một chuỗi để dùng trong UI, ví dụ chuỗi @string/title, nếu thiết đặt bản địa cho thiết bị là en-US, Android sẽ tìm trong các tập tin theo thứ tự sau:

```

res/values-en-rUS/strings.xml
res/values-en/strings.xml
res/values/strings.xml

```

Khi tìm được chuỗi trong tập tin tài nguyên, nó sử dụng trị đó và dùng tìm kiếm. Chú ý, mã bản địa (locale code) có thể bao gồm mã vùng (region code) hoặc không. Ví dụ: fr (không region code), fr-rFR (French/France) và fr-rCA (French/Canada). Như vậy, ứng dụng lựa chọn tập tin tài nguyên để nạp trong thời gian chạy dựa trên thiết đặt bản địa của thiết bị. Nếu tài nguyên bản địa chỉ định không có sẵn, nó sẽ dùng tài nguyên mặc định.

Tạo ứng dụng Android đa ngôn ngữ theo các bước sau:

- Với từng ngôn ngữ áp dụng cho ứng dụng, ví dụ Pháp và Nhật, chuẩn bị một tập tin tài nguyên bản địa strings.xml, chứa các chuỗi định nghĩa bằng ngôn ngữ đó và đặt vào thư mục tương ứng, values-fr và values-jp.
- Tài nguyên drawable cho từng ngôn ngữ, ví dụ hình nền, có tên giống nhau nhưng đặt vào thư mục tương ứng với ngôn ngữ, drawable-fr và drawable-jp.
- Để kiểm tra, bạn thiết đặt bản địa cho thiết bị bằng cách dùng Custom Locale trên thiết bị.

Text-to-Speech và Speech-to-Text

1. Text-to-Speech

Android hỗ trợ chuyển văn bản thành giọng nói từ phiên bản 1.6, cho phép đến vài ngôn ngữ khác nhau. Android cung cấp lớp TextToSpeech cho mục đích này. Bạn cần tạo một thực thể thuộc lớp này và chỉ định OnInitListener cho nó.

Trong listener, bạn chỉ định thuộc tính cho đối tượng TextToSpeech, như ngôn ngữ, cao độ, ... Sau đó, bạn đơn giản gọi phương thức speak() của đối tượng TextToSpeech để đọc văn bản.

```

public class MainActivity extends Activity {
    private TextToSpeech ttObj;
    private EditText txtText;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtText = (EditText)findViewById(R.id.txtText);
        ttObj = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
            @Override public void onInit(int status) {
                if (status != TextToSpeech.ERROR) {
                    ttObj.setLanguage(Locale.UK);
                }
            }
        });
    }

    @Override public void onPause(){
        if (ttObj != null ) {
            ttObj.stop();
            ttObj.shutdown();
        }
        super.onPause();
    }

    public void speakText(View view) {
        String toSpeak = txtText.getText().toString();
        Toast.makeText(getApplicationContext(), toSpeak, Toast.LENGTH_SHORT).show();
        ttObj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
    }
}

```

Một số phương thức của đối tượng TextToSpeech:

addSpeech(String text, String filename)

Tạo một ánh xạ giữa một chuỗi text và một tập tin âm thanh.

getLanguage()

Trả về một thực thể Locale mô tả ngôn ngữ.

isSpeaking()

Kiểm tra xem engine TextToSpeech có đang bận đọc hay không.

setPitch(float pitch)

Thiết lập cao độ giọng nói cho engine TextToSpeech.

setSpeechRate(float speechRate)

Thiết lập tốc độ giọng nói cho engine TextToSpeech.

shutdown()

Giải phóng tài nguyên được dùng bởi engine TextToSpeech.

stop()

Dừng giọng nói.

2. Speech-to-Text

Tính năng chuyển giọng nói thành văn bản được cung cấp trong gói android.speech. Activity gọi sẽ gửi intent bằng phương thức startActivityForResult() để kích hoạt intent đặc biệt android.speech.RecognizerIntent, nó sẽ hiển thị một activity để nhận dạng speech đầu vào. Activity này chuyển speech thành text và gửi ngược trở lại activity gọi. Khi gửi RecognizerIntent, bạn phải cung cấp extras RecognizerIntent.EXTRA_LANGUAGE_MODEL với trị là mã bản địa, ví dụ en-US.

Kết quả chuyển giọng nói thành văn bản trả về như một ArrayList với key RecognizerIntent.EXTRA_RESULTS. Tại activity xử lý kết quả, trong phương thức onActivityResult(), nếu resultCode là RESULT_OK, xử lý ArrayList nhận được.

```
public class MainActivity extends Activity {
    protected static final int RESULT_SPEECH = 1;
    private ImageButton btnSpeak;
    private TextView txtText;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtText = (TextView) findViewById(R.id.txtText);
        btnSpeak = (ImageButton) findViewById(R.id.btnSpeak);
        btnSpeak.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
                intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, "en-US");
                try {
                    startActivityForResult(intent, RESULT_SPEECH);
                    txtText.setText("");
                } catch (ActivityNotFoundException a) {
                    Toast.makeText(getApplicationContext(), "Not supported", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

    @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode) {
            case RESULT_SPEECH:
                if (resultCode == RESULT_OK && null != data) {
                    ArrayList<String> text = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                    txtText.setText(text.get(0));
                }
                break;
        }
    }
}
```

Lưu trữ dữ liệu

Các ứng dụng di động đa số là hướng dữ liệu (data driven), vì vậy lưu trữ dữ liệu bền vững là một phần thiết yếu khi xây dựng ứng dụng. Android cung cấp một số tùy chọn để lưu trữ dữ liệu bền vững cho ứng dụng của bạn. Bạn lựa chọn giải pháp tùy theo trường hợp sử dụng và đặc tả yêu cầu: lưu trữ dữ liệu riêng tư hoặc công cộng, dung lượng lưu trữ.

Các tùy chọn lưu trữ dữ liệu như sau:

Shared Preference	Lưu trữ dữ liệu dưới dạng các cặp key-value.
Internal Storage	Lưu trữ dữ liệu cho từng ứng dụng trên vùng lưu trữ riêng của thiết bị.
External Storage	Lưu trữ dữ liệu công cộng trên thiết bị lưu trữ ngoài dùng chung (thường là card SD).
SQLite Databases	Lưu trữ dữ liệu có cấu trúc trong cơ sở dữ liệu riêng.
Network Connection	Lưu trữ dữ liệu trên mạng với máy chủ chỉ định.
Content Provider	Kho dữ liệu dùng chung toàn cục, chia sẻ bởi các ứng dụng.

Shared Preferences

Android cung cấp một framework để lưu trữ, chia sẻ và quản lý dữ liệu như các cặp key/value đơn giản, nhẹ (lightweight) thông qua `android.content.SharedPreferences`. Các cặp key/value này thường được ứng dụng dùng đến, lưu trữ thông tin người dùng, tùy chọn cấu hình, trạng thái ứng dụng. Preference chỉ lưu trữ trị thuộc các kiểu dữ liệu sau: Boolean, Float, Integer, Long, String, Set của nhiều String (từ API Level 11).

Preference cũng tương tự Bundle, nhưng Bundle không được lưu trữ bền vững, trong lúc preference được lưu trữ trong tập tin XML và được quản lý thông qua API hoặc bằng PreferenceActivity.

Để sử dụng preference cho ứng dụng, theo các bước sau:

- Lấy thực thể SharedPreferences.
- Tạo một SharedPreferences.Editor để hiệu chỉnh nội dung của preference.
- Thay đổi preference bằng Editor.
- Hoàn tất (commit) các thay đổi.

1. Các loại preference

Android cung cấp hai kiểu preference, mặc dù cơ bản chúng giống nhau:

- Preference riêng của activity

Một activity có thể có preference riêng. Dù vẫn thể hiện bởi `android.content.SharedPreferences`, các preference của riêng một activity không chia sẻ được với các activity khác trong ứng dụng. Trong context của một activity, preference của activity, thường được đặt tên theo lớp activity, được lấy bằng phương thức `Activity.getPreferences()`.

```
SharedPreferences mSettingsActivity = getPreferences(MODE_PRIVATE);
```

- Preference dùng chung bởi nhiều activity (generic shared preferences)

Preference dùng chung, còn gọi là preference cấp ứng dụng, được lấy theo cách tương tự, chỉ khác ở chỗ phải cung cấp tên cho tập tin preference và dùng phương thức `Context.getSharedPreferences()`.

```
SharedPreferences mSettings = getSharedPreferences("application", MODE_PRIVATE);
```

Tham số đầu là khóa, tham số thứ hai là các chế độ, cũng thường dùng khi đọc/ghi tập tin:

Chế độ	Mô tả
MODE_APPEND	Nối một preference mới vào preference đang tồn tại.
MODE_ENABLE_WRITE_AHEAD_LOGGING	Cờ mở cơ sở dữ liệu. Nếu thiết lập, nó cho phép ghi log trước theo mặc định.
MODE_MULTI_PROCESS	Kiểm tra sự thay đổi của preference dù preference dùng chung được nạp.
MODE_PRIVATE	Preference chỉ có thể truy xuất bằng cách dùng ứng dụng gọi.
MODE_WORLD_READABLE	Cho phép ứng dụng khác đọc preference.
MODE_WORLD_WRITEABLE	Cho phép ứng dụng khác ghi vào preference.

Nếu dùng tên tập tin preference mặc định, preference lấy được gọi là preference của context, lấy bằng phương thức `PreferenceManager.getDefaultSharedPreferences()` với tham số là đối tượng Context.

```
SharedPreferences mSettings = PreferenceManager.getDefaultSharedPreferences(this);
```

2. Làm việc với preference

a) Lấy và đọc các preference

Cặp key/value của preference được đọc trực tiếp bằng cách dùng các getters của thực thể SharedPreferences mà bạn đang làm việc. Tham số cuối của getters là trị mặc định sẽ trả về nếu không tìm thấy key được yêu cầu.

```
String name = settingsPreferences.getString("name", null);
boolean registered = settingsPreferences.getBoolean("registered", false);
```

Bảng sau trình bày một số phương thức thường gọi của giao diện SharedPreferences:

Phương thức	Mục đích
contains()	Kiểm tra theo tên một preference có tồn tại hay không.
edit()	Lấy một SharedPreferences.Editor để thay đổi preference.
getAll()	Lấy một map chứa tất cả các cặp key/value.
getBoolean()	Lấy theo tên, một preference kiểu Boolean.
getFloat()	Lấy theo tên, một preference kiểu Float.
getInt()	Lấy theo tên, một preference kiểu Integer.
getLong()	Lấy theo tên, một preference kiểu Long.
getString()	Lấy theo tên, một preference kiểu String.
getStringSet()	Lấy theo tên, một preference kiểu Set các String (thêm vào từ API Level 11).

b) Thêm, cập nhật và xóa preference

Để thay đổi preference, bạn phải gọi các phương thức thông qua đối tượng `SharedPreferences.Editor`. Lấy đối tượng `Editor` bằng phương thức `edit()` của thực thể `preference`, sau đó gọi các phương thức của `Editor` để thay đổi preference theo ý muốn rồi hoàn tất (`commit()`) các thay đổi để lưu xuống tập tin.

```
SharedPreferences settings = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putString("name", "Karl Marx");
editor.putBoolean("registered", true);
editor.commit();
```

Bảng sau trình bày một số phương thức hữu dụng của giao diện `SharedPreferences.Editor`:

Phương thức	Mục đích
<code>clear()</code>	Loại bỏ tất cả preference, trước các thao tác put.
<code>remove()</code>	Loại bỏ một preference theo tên, trước các thao tác put.
<code>putBoolean()</code>	Thiết lập theo tên, một preference kiểu Boolean.
<code>putFloat()</code>	Thiết lập theo tên, một preference kiểu Float.
<code>putInt()</code>	Thiết lập theo tên, một preference kiểu Integer.
<code>putLong()</code>	Thiết lập theo tên, một preference kiểu Long.
<code>putString()</code>	Thiết lập theo tên, một preference kiểu String.
<code>putStringSet()</code>	Thiết lập theo tên, một preference kiểu Set các String (thêm vào từ API Level 11).
<code>commit()</code>	Hoàn tất các thay đổi của phiên hiệu chỉnh này.
<code>apply()</code>	Giống <code>commit()</code> , hoàn tất thay đổi trong bộ nhớ lập tức, hoàn tất thay đổi trên đĩa bất đồng bộ bên trong vòng đời của ứng dụng. Áp dụng từ API Level 9, nhanh hơn <code>commit()</code> .

Ứng dụng của bạn có thể lắng nghe những thay đổi của preference dùng chung, bằng cách cài đặt một listener và đăng ký nó với đối tượng `SharedPreferences` chỉ định bằng cách dùng các phương thức:

`registerOnSharedPreferenceChangeListener()` và `unregisterOnSharedPreferenceChangeListener()`.

```
SharedPreferences.OnSharedPreferenceChangeListener listener =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
        @Override
        public void onSharedPreferenceChanged(SharedPreferences preference, String key) {
            if (!preference.contains(key)) {
                // loại bỏ preference
            } else {
                // lấy trị mới cho key
            }
        }
    };
settings.registerOnSharedPreferenceChangeListener(listener);
```

c) Tập tin preference XML

Dữ liệu của preference được lưu trong tập tin XML, bạn có thể tham chiếu các tập tin preference này bằng cách dùng tab `File Explorer` của ADM tại thư mục: `/data/data/<package name>/shared_prefs/<preferences filename>.xml`. `<preferences filename>` là tên lớp `Activity` nếu dùng preference riêng và là tên chỉ định nếu dùng preference dùng chung. Nếu dùng preference dùng chung mặc định, tên tập tin preference có dạng: `<package name>_preferences.xml`.

3. Dùng PreferenceScreen

Android cung cấp lớp `android.preference.PreferenceFragment`, sẽ tự động sinh một màn hình UI, giúp người dùng dễ dàng thao tác lên tập tin preference thông qua đơn thể UI. Điều này thuận tiện nếu bạn muốn tạo một tập các thiết đặt (settings) người dùng một cách đơn giản, nhất quán, và cho phép người dùng chỉnh sửa chúng.

Thực hiện theo các bước sau:

- Định nghĩa màn hình preference trong một tập tin nguồn preference, ví dụ `/res/xml/personal_settings.xml`

Các control dùng nhập thông tin trong màn hình preference có thể là:

`<CheckBoxPreference>` cung cấp checkbox cho một preference boolean.

`<EditTextPreference>` cung cấp ô nhập văn bản cho một preference.

`<ListPreference>` cung cấp danh sách các tùy chọn cho một preference.

`<MultiSelectListPreference>` cung cấp danh sách các tùy chọn kèm theo checkbox cho một preference.

Mỗi control được khai báo với một số thuộc tính:

Thuộc tính	Mô tả
<code>android:key</code>	Chỉ định key của preference tương ứng với control.
<code>android:title</code>	Chỉ định tên của preference, hiển thị như tiêu đề của control.
<code>android:summary</code>	Cung cấp chi tiết về preference, hiển thị như thông tin hướng dẫn kèm theo tiêu đề.
<code>android:defaultValue</code>	Chỉ định trị mặc định của preference, hiển thị trong control.
<code>android:entries</code>	Tham chiếu mảng chuỗi (@array) chứa tên các mục cho <code>ListPreference</code> .
<code>android:entryValues</code>	Tham chiếu mảng chuỗi (@array) chứa các trị cho tên mục tương ứng trong <code>ListPreference</code> .

Ví dụ, tập tin nguồn preference `personal_settings.xml` có hai trường nhận thông tin `Username` và `Email`.

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="Username and Email" >
```

```

<EditTextPreference
    android:key="username"
    android:title="Username"
    android:summary="This is your username"
    android:defaultValue="username01"
    android:dialogTitle="Enter your username:" />
<EditTextPreference
    android:key="email"
    android:title="Email"
    android:summary="Enter your email address"
    android:defaultValue="your@email.com" />
</PreferenceCategory>
</PreferenceScreen>

```

Các chuỗi, mảng chuỗi, trong tập tin XML trên có thể tham chiếu từ /res/values/strings.xml

Cài đặt lớp PreferenceFragment (API Level 11) và kết nối nó với tập tin nguồn preference. Lớp này sẽ nạp tập tin nguồn XML vào màn hình cấu hình chuẩn, bằng phương thức addPreferencesFromResource(). Sau đó, dùng fragment này.

```

public class UserPreferenceFragment extends PreferenceFragment {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PreferenceManager manager = getPreferenceManager();
        manager.setSharedPreferencesName("user_prefs");
        addPreferencesFromResource(R.xml.personal_settings);
    }
}

```

4. Tổ chức preference với header

Khái niệm header của preference được thêm vào từ API Level 11. Mỗi header thể hiện một PreferenceScreen, danh sách các header hiển thị tập trung trong một màn hình để dễ quản lý. Thực hiện theo các bước sau:

- Tạo lớp PrefsActivity là màn hình hiển thị các header, thừa kế PreferenceActivity. Trong lớp này, tạo các lớp nội thừa kế PreferenceFragment. Mỗi PreferenceFragment thể hiện một PreferenceScreen, có màn hình cấu hình định nghĩa trong một tập tin nguồn preference XML, nạp vào bằng phương thức addPreferencesFromResource().

Từ activity chính, bạn hiển thị PrefsActivity bằng cách gửi intent.

- Tạo tập tin nguồn header, /res/xml/preference_headers.xml, trong đó định nghĩa danh sách header bằng cách dùng element <preference-headers> và <header> trong tập tin XML, ví dụ:

```

<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
    <header android:fragment="com.twe.examples.PrefsActivity$UserNameFragment"
        android:title="Personal Settings"
        android:summary="Configure your personal settings" />
    <header android:fragment="com.twe.examples.PrefsActivity$GameSettingsFragment"
        android:title="Game Settings"
        android:summary="Configure your game settings" />
</preference-headers>

```

- Để hiển thị các header trong màn hình của PrefsActivity, trong onBuildHeaders() của PreferenceActivity nạp tập tin nguồn header bằng phương thức loadHeadersFromResource().

```

public class MainActivity extends PreferenceActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override public void onBuildHeaders(List<Header> target) {
        loadHeadersFromResource(R.xml.preference_headers, target);
    }

    public static class UserNameFragment extends PreferenceFragment {
        @Override public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            getPreferenceManager().setSharedPreferencesName("user_prefs");
            addPreferencesFromResource(R.xml.personal_settings);
        }
    }

    public static class GameSettingsFragment extends PreferenceFragment {
        @Override public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            getPreferenceManager().setSharedPreferencesName("user_prefs");
            addPreferencesFromResource(R.xml.game_settings);
        }
    }
}

```

File

Android hỗ trợ tất cả Java API trong gói `java.io` dùng cho các tác vụ truy cập tập tin truyền thống: tạo, đọc, cập nhật và xóa (CRUD) tập tin. Ngoài ra, Android cung cấp thêm một số hỗ trợ giúp truy cập thuận tiện tập tin tại một số vị trí chỉ định trên thiết bị:

- Lưu trữ trong (internal storage): vùng lưu trữ trong (non-removable) của thiết bị, mặc định là dùng riêng (private). Thường lưu trữ các dữ liệu có cấu trúc phức tạp dùng riêng cho ứng dụng. Khi gỡ cài đặt ứng dụng, Android sẽ loại bỏ dữ liệu này theo ứng dụng.
 - Lưu trữ ngoài (external storage): vùng lưu trữ bên ngoài trên thẻ, được ghép (mount) vào hệ thống tập tin của thiết bị. Thường là card SD (Secure Digital) trong thiết bị. Cần quyền `WRITE_EXTERNAL_STORAGE` từ API Level 4+. Thường lưu trữ các tập tin có dữ liệu dùng chung, công cộng, ví dụ tập tin log hoặc tập tin có kích thước lớn.
- Các tập tin lưu trữ ngoài có thể được đọc/ghi khi người dùng kết nối thiết bị với máy tính thông qua USB (cho phép USB mass storage). Chú ý rằng khi đó các tập tin này xem như không có sẵn cho ứng dụng đang chạy trên thiết bị.
- Assets: vùng lưu trữ được bảo vệ chỉ đọc, trong gói `.apk`. Thường dùng như nguồn dữ liệu cục bộ.

Như đã nói ở trên, ta chủ yếu dùng các lớp `FileInputStream` và `FileOutputStream` của gói `java.io` cho việc truy cập dữ liệu tập tin. Thực tế, cũng giống như trong Java, bạn có thể tạo thực thể `File` với đường dẫn tuyệt đối và sử dụng stream để đọc và ghi dữ liệu. Bạn cũng có thể lồng các stream bằng `PrintWriter`, `BufferedReader`, `Scanner` để có các phương thức truy cập thuận tiện hơn.

1. Internal storage

a) Tập tin tài nguyên

Các tập tin lưu như tập tin tài nguyên (raw resource), có thể được lưu trong thư mục `/res/raw` và truy cập dễ dàng.

Ví dụ, kéo thả tập tin `data.txt` vào `/res/raw/data.txt`, rồi lấy stream nhập dùng truy cập tập tin đó trong chương trình:

```
String s = "";
StringBuffer buffer = new StringBuffer();
InputStream in = this.getResources().openRawResource(R.raw.data);
BufferedReader reader = new BufferedReader(new InputStreamReader(in));
if (in != null) {
    while ((s = reader.readLine()) != null) {
        buffer.append(s + "\n");
    }
}
in.close();
Toast.makeText(getApplicationContext(), buffer.toString(), Toast.LENGTH_LONG).show();
```

b) Lưu trữ trong

Các tập tin lưu trữ trong theo ứng dụng, tại `/data/data/<package_name>/files/` trên máy ảo. Bạn chú ý, trên thiết bị thật chưa root, sẽ không thấy tập tin này. Bạn có thể dùng `File Explorer` của ADM để truy cập chúng.

Các tập tin này có thể được tạo, truy cập, chỉnh sửa bằng cách dùng các phương thức `Context.openFileInput()` và `Context.openFileOutput()`, chúng trả về các stream `InputStream/OutputStream` cần thiết. Các phương thức này chỉ yêu cầu tham số là *tên của tập tin*, thay vì toàn bộ đường dẫn, và sẽ tham chiếu đến tập tin liên quan trong thư mục được bảo vệ dành riêng cho ứng dụng, không quan tâm đường dẫn chính xác trên thiết bị cụ thể.

```
public class InternalActivity extends Activity {
    static final String FILENAME = "data.txt";
    static final int READ_BLOCK_SIZE = 128;
    EditText mEditText;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        mEditText = (EditText) findViewById(R.id.text_id);
    }

    public void onClickSave(View view) {
        // Tạo một tập tin và ghi vào tập tin
        String s = mEditText.getText().toString();
        try {
            OutputStreamWriter out = new OutputStreamWriter(openFileOutput(FILENAME, MODE_APPEND));
            out.write(s);
            out.flush();
            out.close();
            mEditText.setText("");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

public void onClickLoad(View view) {
    // Đọc tập tin vừa tạo và hiển thị lên màn hình
    try {
        InputStreamReader in = new InputStreamReader(openFileInput(FILENAME));
        char[] buffer = new char[READ_BLOCK_SIZE];
        String s = "";
        int charRead;
        while ((charRead = in.read(buffer)) > 0) {
            s += String.copyValueOf(buffer, 0, charRead);
            buffer = new char[READ_BLOCK_SIZE];
        }
        mEditText.setText(s);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    // Xóa tập tin vừa đọc (nếu cần)
    deleteFile(FILENAME);
}
}

```

Do các phương thức `openFileInput()`, `openFileOutput()`, `close()` và `deleteFile()` thuộc về đối tượng `Context`, không liên kết với `Activity`, kiểu truy cập tập tin này có thể xuất hiện tại vị trí bất kỳ nào trong ứng dụng mà bạn cần, như từ `BroadcastReceiver` hoặc đơn thể khác.

Android cũng hỗ trợ các chế độ `MODE_WORLD_READABLE` và `MODE_WORLD_WRITABLE`, nhưng đã lạc hậu. Nếu muốn chia sẻ dữ liệu cho ứng dụng khác trên thiết bị, bạn nên dùng cách an toàn hơn, như `content provider`.

2. External storage

Khi ghi lên thiết bị lưu trữ ngoài, thường là card SD, bạn cần chú ý:

- Tập tin được lưu trữ tại `/mnt/sdcard/<dir_name>/<file_name>`, dùng `File Explorer` của ADM để lấy tập tin này hoặc dùng `File Manager` trên thiết bị ảo để xem.
- Thêm khai báo cấp quyền cả đọc lẫn ghi lên thiết bị lưu trữ ngoài trong tập tin manifest của ứng dụng:

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

- Cần thận kiểm tra xem thiết bị lưu trữ ngoài có gắn vào hay không, bằng `Environment.getExternalStorageState()`. Lấy root của thiết bị lưu trữ ngoài bằng `Environment.getExternalStorageDirectory()`. Nếu thiết bị lưu trữ không được ghép (mount) vào hệ thống, trả về khác với `Environment.MEDIA_MOUNTED`, ta không tiến hành lưu trữ vì không ghi lên thiết bị được.

```

public class ExternalActivity extends Activity {
    private static final String FILENAME = "data.txt";
    private static final String DNAME = "myfiles";
    static final int READ_BLOCK_SIZE = 128;
    EditText mEditText;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        mEditText = (EditText) findViewById(R.id.text_id);
    }

    public void onClickSave(View view) {
        // Kiểm tra thiết bị lưu trữ ngoài có tồn tại
        if (!Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())) {
            Toast.makeText(this, "Cannot use storage.", Toast.LENGTH_SHORT).show();
            finish();
            return;
        }
        String s = mEditText.getText().toString();
        // Tạo một thư mục mới trên external storage
        try {
            File rootPath = Environment.getExternalStorageDirectory();
            File directory = new File(rootPath.getAbsolutePath() + "/" + DNAME);
            directory.mkdirs();
            // Tạo và ghi vào tập tin
            File file = new File(directory, FILENAME);
            OutputStreamWriter out = new OutputStreamWriter(new FileOutputStream(file));
            out.write(s);
            out.flush();
            out.close();
            mEditText.setText("");
        } catch (FileNotFoundException e) {

```

```

        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void onClickLoad(View view) {
    // Kiểm tra thiết bị lưu trữ ngoài có đang dùng
    if (!Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())) {
        Toast.makeText(this, "Cannot use storage.", Toast.LENGTH_SHORT).show();
        finish();
        return;
    }
    // Đọc tập tin vừa tạo và hiển thị lên màn hình
    try {
        File rootPath = Environment.getExternalStorageDirectory();
        File directory = new File(rootPath.getAbsolutePath() + "/" + DNAME);
        File file = new File(directory, FILENAME);
        InputStreamReader in = new InputStreamReader(new FileInputStream(file));
        char[] buffer = new char[READ_BLOCK_SIZE];
        String s = "";
        int charRead;
        while ((charRead = in.read(buffer)) > 0) {
            s += String.copyValueOf(buffer, 0, charRead);
            buffer = new char[READ_BLOCK_SIZE];
        }
        mEditText.setText(s);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    // Xóa tập tin vừa đọc (nếu cần)
    dataFile.delete();
}
}

```

3. External System Directory

Nếu không dùng Genymotion mà dùng AVD do Android cung cấp, để lưu giữ các tập tin ghi lên nó, bạn cần phải tạo card SD như sau:

- Vào thư mục <Anroid sdk>/tools, dùng công cụ mksdcard tạo ảnh của card SD:

mksdcard <dung lượng> <tên tập tin ảnh>

- Trong bước Verify Configuration một AVD, click nút Show Advanced Settings, trong phần Memory and Storage > SD card, chọn External file và tìm đến tập tin ảnh vừa tạo.

Card SD thường được tổ chức khác nhau tùy theo nhà sản xuất thiết bị Android mà card gắn vào. Ví dụ, khi lưu hình ảnh từ camera, Nexus lưu tại /sdcard/DCIM/Camera, HTC lưu tại /sdcard/DCIM/100MEDIA. Vì vậy bạn cần dùng các hằng do Environment cung cấp để xác định vị trí thư mục lưu trữ chuẩn trên card SD. Trong ví dụ trên, thay vì dùng đường dẫn đến thư mục chứa hình ảnh, nên dùng DIRECTORY_PICTURES. Điều này cũng tránh việc dùng root để lưu trữ.

Environment.getExternalStoragePublicDirectory(String type)

Trả về thư mục chung, nơi các ứng dụng lưu trữ tập tin. Nội dung của thư mục này được người dùng và các ứng dụng khác thấy rõ. Một số type có trị hợp lệ là DIRECTORY_PICTURES, DIRECTORY_MUSIC, DIRECTORY_MOVIES, và DIRECTORY_RINGTONES.

Các phương thức của Context giúp truy cập vị trí lưu trữ chuẩn này khi ghi tập tin:

Context.getExternalFilesDir(String type)

Trả về một thư mục trên thiết bị lưu trữ ngoài cho các tập tin của ứng dụng chỉ định. Tập tin lưu ở đây không xem là công cộng, không hiển thị trong MediaStore. Tuy nhiên, người dùng và các ứng dụng khác vẫn có thể xem và chỉnh sửa. Tập tin cũng có thể bị gỡ bỏ khi gỡ cài đặt ứng dụng. Việc lưu trữ này thích hợp khi ứng dụng cần đặt chỗ cho tập tin lớn, không tiện lưu trữ trong. Một số type có trị hợp lệ là DIRECTORY_PICTURES, DIRECTORY_MUSIC, DIRECTORY_MOVIES, và DIRECTORY_RINGTONES.

Context.getExternalCacheDir()

Cache là vùng lưu trữ tạm thời, dùng cung cấp truy cập nhanh đến dữ liệu lấy được từ một cơ chế khác. Ví dụ, tập tin dữ liệu lớn được tải về từ mạng có thể đặt vào thư mục cache của ứng dụng. Phương thức trên dùng thư mục cache của ứng dụng, nằm trên thiết bị lưu trữ ngoài.

Khi gỡ cài đặt ứng dụng, thư mục cache của ứng dụng cũng bị xóa. Ngoài ra, Android sẽ xóa thư mục cache nếu thiếu không gian lưu trữ, khi đó ứng dụng lấy dữ liệu theo cách như trước khi dùng cache.

4. JSON và XML

Sử dụng tập tin đơn giản để lưu trữ dữ liệu sẽ gặp khó khăn khi phải làm việc với dữ liệu có cấu trúc phức tạp như các bản ghi (record). Thông thường ứng dụng cần chuyển đổi qua lại giữa dữ liệu lưu trữ và các POJO hoặc collection các POJO. Để dễ chuyển đổi, dữ liệu phức tạp thường được lưu trữ dưới định dạng XML hoặc JSON.

a) Phân tích dữ liệu JSON (JavaScript Object Notation)

Năm 1999, Douglas Crockford giới thiệu JSON như một giải pháp thay thế XML trong lưu chuyển dữ liệu giữa client và server. JSON là một định dạng lưu chuyển dữ liệu nhẹ. Nó dễ dàng được tạo ra, đọc, viết và parse. Đặc biệt, JSON là một ánh xạ trực quan của XML nhưng đơn giản hơn nhiều. Cú pháp tóm tắt của JSON được mô tả trong hình dưới.

JSON được xây dựng trên hai cấu trúc:

- Một collection các cặp tên/trị. Trong nhiều ngôn ngữ, điều này hiện thực như một đối tượng.
- Một danh sách có thứ tự. Trong nhiều ngôn ngữ, điều này hiện thực như một mảng.

Chúng ta hãy so sánh kết quả lưu trữ cùng một nội dung giữa tập tin XML với tập tin JSON.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<books>
  <book isbn="978-1-484200-20-9">
    <title>Android Apps for Absolute Beginners</title>
    <price>39.99</price>
  </book>
  <book isbn="978-1-4302-6532-0">
    <title>Introducing Spring Framework</title>
    <price>44.99</price>
  </book>
  <book isbn="978-1-4302-5962-6">
    <title>Pro SQL Server Internals</title>
    <price>59.99</price>
  </book>
</books>
```

```
{
  "books": {
    "book": [
      {
        "isbn": "978-1-484200-20-9",
        "title": "Android Apps for Absolute Beginners",
        "price": "39.99"
      },
      {
        "isbn": "978-1-4302-6532-0",
        "title": "Introducing Spring Framework",
        "price": "44.99"
      },
      {
        "isbn": "978-1-4302-5962-6",
        "title": "Pro SQL Server Internals",
        "price": "59.99"
      }
    ]
  }
}
```

Cú pháp JSON

string	" "
object	" chars "
{ }	chars
{ members }	char
members	char chars
pair	char
pair , members	any-Unicode-character-
pair	except-"-or-\-or-
string : value	control-character
array	\ " \f
[]	\\ \n
[elements]	\/ \r
elements	\b \t
value	\u four-hex-digits
value , elements	number
value	int
string	int frac
number	int exp
object	int frac exp
array	int
true	digit
false	digit1-9 digits
null	- digit
	- digit1-9 digits
	frac
	. digits
	exp
	e digits
	digits
	digit
	digit digits
	e
	e E
	e+ E+
	e- E-

Android cung cấp bốn lớp để thao tác với dữ liệu JSON: JSONArray, JSONObject, JSONStringer và JSNTTokenizer.

Đầu tiên, cần xác định được cấu trúc của đối tượng JSON mà bạn quan tâm. Đối tượng JSON có thể phức hợp, ví dụ là một mảng các đối tượng JSON con.

- Xem cấu trúc đối tượng JSON tại: <http://json.parser.online.fr/> hoặc <http://chris.photobooks.com/json/>

- Xem và hiệu chỉnh cấu trúc JSON tại: <http://www.jsoneditoronline.org/>

[-] Object, 1 property

books	[-] Object, 1 property			
	book	[-] Array data structure, 3 items		
		[key]	@isbn	price
		0	978-1-484200-20-9	39.99
		1	978-1-4302-6532-0	44.99
		2	978-1-4302-5962-6	59.99
				title
				Android Apps for Absolute Beginners
				Introducing Spring Framework
				Pro SQL Server Internals

Sau đây là các thành phần trong tập tin JSON:

Thành phần	Mô tả
Array ([])	[] thể hiện một mảng JSON.
Object ({ })	{ } thể hiện một đối tượng JSON.
Key	Khóa kiểu string, các cặp Key/Value tạo thành đối tượng JSON.
Value	Trị có kiểu string, char, int, exp, number.

Để phân tích dữ liệu JSON, bạn tạo một đối tượng thuộc lớp `JSONObject`, truyền cho nó một chuỗi chứa dữ liệu JSON, lấy từ một stream nhập nào đó. Tùy theo cấu trúc của đối tượng JSON phức hợp này, tách lấy đối tượng JSON con rồi dùng các getters của nó để lấy dữ liệu đưa vào POJO tương ứng.

```
String in = new BufferedReader(new FileReader("books.json")).readLine();
List<Book> books = new ArrayList<Book>(); // books là ArrayList chứa các POJO Book
try {
    JSONObject json = new JSONObject(in);
    JSONArray jArray = json.getJSONArray("books");
    for (int i = 0; i < jArray.length(); i++) {
        JSONObject json_data = jArray.getJSONObject(i);
        Book book = new Book();
        book.setIsbn(json_data.getString("isbn"));
        book.setTitle(json_data.getString("title"));
        book.setPrice(Float.parseFloat(json_data.getString("price")));
        books.add(book);
    }
} catch (JSONException e) {
    e.printStackTrace();
}
return books;
```

Trong thực tế, bạn có thể dùng thư viện Gson để dễ dàng chuyển đổi qua lại giữa đối tượng JSON và POJO tương ứng. Thư viện Gson tải về từ: <https://code.google.com/p/google-gson/>.

Đầu tiên, tạo lớp POJO, nếu tên trường của lớp khác với key tương ứng của đối tượng JSON, dùng `@SerializedName()` để báo tên key tương ứng. Sau đó, thay vì phải parse "cấp thấp" dữ liệu JSON, bạn gọi phương thức `fromJson()` của đối tượng Gson. Phương thức này có hai tham số: stream nhập chứa dữ liệu JSON và kiểu lớp POJO tương ứng.

```
import com.google.gson.Gson;
import com.google.gson.annotations.SerializedName;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.List;

public class BookList {
    @SerializedName("book")
    private List<Book> bookList;

    static class Book {
        String isbn;
        String title;
        double price;

        @Override public String toString() {
            return String.format("[%s] %s (%.2f)", isbn, title, price);
        }
    }

    public static void main(String[] args) {
        try {
            Gson gson = new Gson();
            BookList books = gson.fromJson(new FileReader("books.json"), BookList.class);
            final List<Book> list = books.bookList;
            for (BookList.Book book : list) {
                System.out.println(book);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

b) Đọc/ghi dữ liệu JSON

Android cung cấp API để đọc/ghi dữ liệu trong định dạng JSON. Ứng dụng có thể lưu trữ bền vững dữ liệu với định dạng JSON dễ dàng trên lưu trữ trong và thiết bị lưu trữ ngoài.

Lớp `android.util.JsonWriter` dùng ghi một stream JSON ra một stream xuất:

```
try {
    JsonWriter writer = new JsonWriter(new OutputStreamWriter(openFileOutput("data.json", MODE_PRIVATE)));
    try {
        writer.setIndent("  ");
        writer.beginObject();
        writer.name("name");
        writer.value("Karl Marx");
```

```

        writer.name("email");
        writer.value("karlmarx.cinar@socialism.com");
        writer.endObject();
    } finally {
        writer.close();
    }
} catch (IOException e) {
    e.printStackTrace();
}

```

Dữ liệu lưu trong tập tin data.json trên có thể được đọc dễ dàng bằng cách dùng lớp android.util.JsonReader.

```

try {
    JsonReader reader = new JsonReader(new InputStreamReader(openFileInput("data.json")));
    try {
        reader.beginObject();
        while (reader.hasNext()) {
            String name = reader.nextName();
            if ("name".equals(name)) {
                String firstName = reader.nextString();
            } else if ("email".equals(name)) {
                String email = reader.nextString();
            } else {
                reader.skipValue();
            }
        }
        reader.endObject();
    } finally {
        reader.close();
    }
} catch (IOException e) {
    e.printStackTrace();
}

```

c) Phân tích dữ liệu XML

Android cung cấp ba kiểu parser dùng cho dữ liệu XML là DOM, SAX và XmlPullParser. Android đề nghị chọn XmlPullParser do dễ dùng và hiệu quả. Khi sử dụng parser kiểu này, ứng dụng sẽ chủ động gọi parser để "kéo" về sự kiện kế tiếp trong một vòng lặp xử lý.

Đầu tiên bạn cũng cần xác định cấu trúc dữ liệu XML lưu trữ của đối tượng mà bạn cần quan tâm.

Để phân tích dữ liệu XML, bạn cần tạo một đối tượng XmlPullParserFactory, từ đối tượng này sinh ra đối tượng XmlPullParser bằng phương thức newPullParser():

```

private XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
private XmlPullParser parser = factory.newPullParser();

```

Tiếp theo, cung cấp stream lấy từ chuỗi URL, chứa dữ liệu XML cho XmlPullParser.

```

parser.setInput(new URL(url).openStream(), null);

```

Sau đó, chạy vòng lặp xử lý sự kiện, chương trình hỏi parser bằng cách gọi phương thức next(), mỗi lần gọi phương thức này sẽ phát ra một sự kiện được liệt kê như hằng số của lớp XmlPullParser.

Tùy thuộc vào kiểu sự kiện nhận được, ta lấy thêm thông tin bằng cách gọi phương thức tương ứng, phù hợp với kiểu sự kiện đó. Ví dụ, nếu parser phát ra sự kiện START_ELEMENT, ta gọi getName() để lấy tên của element đang duyệt.

XmlPullParser có các phương thức riêng dùng phân tích các thành phần của XML như: tag, text, các attribute, v.v...

```

List<Book> books = new ArrayList<Book>();
try {
    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
    XmlPullParser parser = factory.newPullParser();
    parser.setInput(new URL(url).openStream(), null);
    int eventType = parser.getEventType();
    Book book;
    while (eventType != XmlPullParser.END_DOCUMENT) {
        String tagName = parser.getName();
        switch (eventType) {
            case XmlPullParser.START_TAG:
                if (tagName.equals("book")) {
                    book = new Book();
                    book.setIsbn(parser.getAttributeValue(null, "isbn"));
                }
                break;
            case XmlPullParser.START_TAG:
                books.add(book);
                break;
            case XmlPullParser.TEXT:
                if (tagName.equals("title")) {

```

```

        book.setTitle(parser.getText());
    } else if (tagName.equals("price")) {
        book.setPrice(Float.parseFloat(parser.getText("price")));
    }
}
eventType = parser.next();
}
} catch (XmlPullParserException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
return books;

```

SQLite Relational Database

1. SQLite

SQLite là một hệ cơ sở dữ liệu nhúng dựa trên chuẩn SQL-92, được phát hành phiên bản Alpha vào năm 2000, với đặc điểm giống như một phần mềm khả chuyển (portable): không cần cài đặt, không cần cấu hình, không cần server, không cần khởi động, không username/password, không phân quyền sử dụng.

Cơ sở dữ liệu SQLite có thể sao chép dễ dàng từ hệ thống này sang hệ thống khác mà không cần chuyển đổi, nâng cấp. SQLite gọn, nhẹ, đơn giản, tất cả được đóng gói trong một tập tin duy nhất chưa đến 550KB, rất khác so với các hệ cơ sở dữ liệu khác như SQL Server hay Oracle. Tuy vậy, SQLite vẫn đáp ứng hầu hết mọi yêu cầu mà một hệ cơ sở dữ liệu quan hệ cần có: transaction, view, query, data types, constraint, v.v...

a) Ưu điểm của SQLite

- Tin cậy, transaction được thực hiện trọn vẹn, không gây lỗi khi xảy ra sự cố phần cứng.
- Tốc độ nhanh hơn nhiều so với các hệ cơ sở dữ liệu khác.
- Hỗ trợ các ngôn ngữ lập trình phổ biến (36 ngôn ngữ lập trình).
- Mã nguồn mở và được chú thích, hướng dẫn sử dụng rõ ràng.
- Hỗ trợ tập Unicode UTF-8.
- Hỗ trợ dòng lệnh.
- Không cần chỉ định kiểu dữ liệu.
- Phù hợp với nhiều hệ điều hành UNIX (Linux, Mac OS-X, Android, iOS) và Windows (Win32, WinCE, WinR).

SQLite giữ vị trí thứ 8 về mức độ sử dụng phổ biến của các hệ cơ sở dữ liệu trên thế giới. SQLite được tác giả Dwayne Richard Hipp viết bằng ngôn ngữ C dưới dạng thư viện.

b) Hạn chế của SQLite

- Chưa hỗ trợ right outer join, full outer join, drop column, alter column, add constraint.
- Chưa hỗ trợ đầy đủ trigger.
- View trong SQLite là chỉ đọc.

c) Kiểu dữ liệu trong SQLite

SQLite là cơ sở dữ liệu không cần chỉ định kiểu, điều này có nghĩa là bạn có thể lưu trữ dữ liệu mà không cần quan tâm đến kiểu của nó, chức năng này có từ SQLite Version 2. SQLite mong muốn tạo ra một hệ cơ sở dữ liệu thực sự tiện lợi và đơn giản, tuy nhiên nếu bạn muốn sử dụng các kiểu dữ liệu cố định cho từng cột và ràng buộc về quan hệ giữa các bảng, bạn vẫn có thể khai báo kiểu dữ liệu với nhiều cách khác nhau.

SQLite có bộ thư viện riêng hỗ trợ datatype map tương ứng với từng ngôn ngữ lập trình. Tuy nhiên, hiện tại SQLite vẫn chưa hỗ trợ các kiểu dữ liệu có kích thước lớn ví dụ như nvarchar, thay vào đó bạn sử dụng ntext (Microsoft, Oracle khuyên tránh sử dụng kiểu dữ liệu text hay ntext). Vì thế khi quyết định lựa chọn SQLite làm hệ cơ sở dữ liệu, bạn nên nghĩ đến việc dữ liệu bạn lưu trữ không thực sự quá lớn. Ngoài ra, để đảm bảo tính ổn định, độ tin cậy, các kiểu dữ liệu có kích thước lớn như image tốt nhất không nên lưu vào SQLite.

2. Cài đặt và sử dụng

Lựa chọn phiên bản phù hợp và tải về từ <http://www.sqlite.org/download.html>.

Tiến hành giải nén và nhận được tập tin sqlite3.exe (SQLite version 3), để tiện cho việc sử dụng SQLite bằng dòng lệnh, nên đặt tập tin sqlite3.exe trong thư mục nằm ở root của Windows, ví dụ C:\sqlite3.

Vào console của Windows, di chuyển đến thư mục chứa tập tin SQLite, tạo cơ sở dữ liệu MyDB.db:

```
sqlite3 MyDB.db
```

Nên đặt tên database và phần mở rộng phù hợp với sản phẩm, ngôn ngữ lập trình đang sử dụng.

Phần mở rộng chưa có định nghĩa chính thức nên có thể không cần hoặc đặt tùy ý, nhưng nên tránh .sdb. Các phần mở rộng hay dùng: .db, .sqlite (Firefox, Apple sử dụng .sqlite).

Cũng có thể mở cơ sở dữ liệu có sẵn trên thiết bị, chú ý đường dẫn đến cơ sở dữ liệu trên thiết bị Android:

```
sqlite3 /data/data/com.android.providers.contacts/databases/contacts.db
```

Một số lệnh SQLite:

Lệnh	Mô tả
.help	Xem trợ giúp.
.databases	Hiển thị database hiện hành và các database đang attach.
.open <i>db_name</i>	Mở database <i>db_name</i> . Sau khi mở, có thể nhập và thực thi trực tiếp các phát biểu SQL.

.tables	Xem danh sách các bảng.
.schema <i>table_name</i>	Xem cấu trúc của bảng <i>table_name</i> .
.exit	Thoát khỏi SQLite.

Ví dụ sau trình bày một phiên làm việc với cơ sở dữ liệu SQLite tự tạo lưu trên AVD đang chạy. Màu xanh là các lệnh nhập từ dấu nhắc (màu nâu) lần lượt của Windows, Android và SQLite.

```
D:\android_sdk\platform-tools>adb shell
root@android:/ # cd /data/data
cd /data/data
root@android:/data/data # mkdir com.twe.dbexample
mkdir com.twe.dbexample
root@android:/data/data # cd com.twe.dbexample
cd com.twe.dbexample
root@android:/data/data/com.twe.dbexample # mkdir databases
mkdir databases
root@android:/data/data/com.twe.dbexample # cd databases      (1)
cd databases
root@android:/data/data/com.twe.dbexample/databases # sqlite3 ./mydatabase.db      (2)
sqlite3 ./mydatabase.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
(3)
sqlite> create table contacts (_id integer primary key autoincrement, name text, address text, phone text);
create table contacts (_id integer primary key autoincrement, name text, address text, phone text);
sqlite> .tables
.tables
contacts
sqlite> insert into contacts (name, address, phone) values ("Barack Obama", "1600 Pennsylvania Ave NW, Washington", "(202) 456-1111");      (4)
insert into contacts (name, address, phone) values ("Barack Obama", "1600 Pennsylvania Ave NW, Washington", "(202) 456-1111");
sqlite> insert into contacts (name, address, phone) values ("Vladimir Putin", "103073, Moscow, Kremlin", "(495) 695-3776");
insert into contacts (name, address, phone) values ("Vladimir Putin", "103073, Moscow, Kremlin", "(495) 695-3776");
sqlite> select * from contacts;      (5)
select * from contacts;
1|Barack Obama|1600 Pennsylvania Ave NW, Washington|(202) 456-1111
2|Vladimir Putin|103073, Moscow, Kremlin|(495) 695-3776
sqlite> select * from contacts where name like '%bama%';
select * from contacts where name like '%bama%';
1|Barack Obama|1600 Pennsylvania Ave NW, Washington|(202) 456-1111
sqlite> .exit
.exit
root@android:/data/data/com.twe.dbexample/databases # exit
exit
```

(1) Tập tin SQLite lưu tại /data/data/<package name>/databases/<database filename>.db

(2) Tạo tập tin cơ sở dữ liệu và vào dấu nhắc tương tác SQLite.

(3) Tạo bảng và xem bảng.

(4) Chèn hai dòng mới vào cơ sở dữ liệu.

(5) Truy vấn cơ sở dữ liệu.

3. Quản lý SQLite

Có khá nhiều công cụ GUI dùng quản lý SQLite: SQLite Database Browser (<http://sourceforge.net/projects/sqlitebrowser/>), SQLite Administrator, SQLiteStudio, SQLite Manager (add-on của Firefox), Dbeaver, Navicat, .v.v... Chúng tôi đề nghị:

a) SQLite Manager

SQLite Manager là add-on của FireFox, cài đặt từ <https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>.

SQLite Manager cho phép bạn quản lý cơ sở dữ liệu bằng các chức năng và công cụ có sẵn. Nếu bạn có những yêu cầu đặc biệt đối với cơ sở dữ liệu mà SQLite Manager chưa hỗ trợ, bạn có thể sử dụng vùng "Create statement" để thực thi các câu lệnh của giống như sử dụng dòng lệnh.

b) Navicat

Navicat hỗ trợ rất nhiều hệ cơ sở dữ liệu và SQLite là một trong số đó. Thao tác với Navicat tương tự như sử dụng SQL Server Management Studio của Microsoft.

Tải về và cài đặt từ <http://www.navicat.com/products/navicat-for-sqlite>. Navicat là phần mềm thương mại có thu phí.

Sau khi cài đặt hoàn tất, chạy Navicat và chọn Connect để kết nối đến cơ sở dữ liệu chỉ định hoặc tạo cơ sở dữ liệu mới.

4. Truy cập cơ sở dữ liệu với SQLite

Gói android.database.sqlite chứa các lớp giúp truy cập cơ sở dữ liệu SQLite.

a) Tạo cơ sở dữ liệu

Để tạo cơ sở dữ liệu, bạn cần gọi phương thức `openCreateDatabase()` với tham số là tên cơ sở dữ liệu và chế độ. Phương thức này trả về thực thể `SQLiteDatabase`, chính là cơ sở dữ liệu bạn cần tạo:

```
SQLiteDatabase database = openOrCreateDatabase("database_name", MODE_PRIVATE, null);
```

Còn nhiều phương thức tạo cơ sở dữ liệu:

```
openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler handler)
```

Phương thức này chỉ mở cơ sở dữ liệu đang có tại đường dẫn `path`, với cờ chế độ tương ứng. Cờ chế độ thường dùng là các hằng thuộc lớp `SQLiteDatabase`: `OPEN_READWRITE`, `OPEN_READONLY`, `CREATE_IF_NECESSARY`.

```
openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)
```

Tương tự phương thức trên, mở cơ sở dữ liệu đang có tại đường dẫn `path`, nhưng không cung cấp xử lý lỗi.

```
openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)
```

Mở cơ sở dữ liệu và tạo mới nếu nó chưa tồn tại.

```
openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)
```

Tương tự phương thức trên, nhưng nó dùng đối tượng `File` như một đường dẫn thay vì dùng `String`.

Tập tin cơ sở dữ liệu SQLite được lưu tại: `/data/data/<package_name>/databases/<database-name>` trên máy ảo. Trên thiết bị thật chưa root, sẽ không thấy tập tin này.

Bạn nên cài đặt thêm plugin Questoid SQLite Manager để xem các bảng dữ liệu được lưu trữ.

Tải về `com.questoid.sqlitebrowser_xxx.jar`, đưa vào thư mục `eclipse/dropins/`. Vào File Explorer của ADM sẽ thấy icon gọi plugin này.

b) Thực hiện truy vấn SQL

execSQL()

Với các truy vấn thay đổi cấu trúc cơ sở dữ liệu (action query) như `create table`, `insert`, `update`, `delete`, ... cách thực hiện truy vấn đơn giản nhất là xây dựng phát biểu truy vấn SQL rồi truyền nó như tham số đến phương thức `execSQL()` của đối tượng thuộc lớp `SQLiteDatabase`:

```
execSQL(String sql, Object[] bindArgs)
```

Thực hiện phát biểu SQL với tham số cung cấp bởi mảng đối tượng truyền đến.

Ví dụ: tạo bảng `Accounts` và chèn một record vào bảng.

```
database.execSQL("CREATE TABLE IF NOT EXISTS Accounts (Username VARCHAR, Password VARCHAR);");
database.execSQL("INSERT INTO Accounts VALUES ('root', 'toor');");
```

rawQuery() và query()

Với các truy vấn truy cập dữ liệu (retrieval query), sẽ trả về một tập kết quả (resultset) với con trỏ chỉ đến, chính là đối tượng thuộc lớp `Cursor`. Có hai cách thực hiện truy vấn:

- Nếu truy vấn liên quan nhiều bảng, dùng phương thức `rawQuery()` của lớp `SQLiteDatabase`.

```
Cursor resultSet = database.rawQuery("SELECT * FROM Accounts", null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

- Nếu truy vấn đơn giản *chỉ trong một bảng*, dùng phương thức `query()` của lớp `SQLiteDatabase`. Phương thức này cung cấp các tham số đầy đủ để tạo câu truy vấn chi tiết cho một bảng.

Tùy phương thức nạp chồng được gọi, các tham số có thể là:

Tham số	Ý nghĩa
<code>distinct</code>	Trị boolean, true nếu muốn các hàng trả về là duy nhất.
<code>table</code>	Tên bảng.
<code>columns</code>	Danh sách các cột trả về, null sẽ trả về tất cả các cột.
<code>selection</code>	Lọc các hàng trả về, giống phát biểu SQL WHERE, null sẽ trả về tất cả các hàng.
<code>selectionArgs</code>	Mảng chuỗi chứa các trị tương ứng các tham số "giữ chỗ" ? trong phần selection.
<code>groupBy</code>	Lọc các hàng trả về theo nhóm, giống phát biểu SQL GROUP BY, null sẽ không nhóm các hàng.
<code>having</code>	Khi các hàng được nhóm, dùng để lọc nhóm sẽ chứa trong Cursor, giống phát biểu SQL HAVING.
<code>orderBy</code>	Cách sắp xếp các hàng trả về, giống phát biểu SQL ORDER BY, null sẽ sắp xếp mặc định (không sắp xếp).
<code>limit</code>	Giới hạn số hàng trả về, dạng chuỗi, giống phát biểu SQL LIMIT, null sẽ không giới hạn số hàng trả về.

Ví dụ:

```
String[] columns = { "dno", "Avg(salary) as AVG" };
String[] conditionArgs = { "F", "123456789" };
Cursor cursor = database.query(
    "EmployeeTable",           // table
    columns,                   // columns
    "gender = ? AND superSsn = ?", // selection
    conditionArgs,             // selectionArgs
    "dno",                     // groupBy
    "Count(*) > 2",             // having
    "AVG Desc",                 // orderBy
    "10,3",                     // limit 10 offset 3
    );
```

Cursor

`rawQuery()` và `query()` trả về đối tượng lớp `android.database.Cursor`, bạn có thể dùng các phương thức của lớp này để di chuyển con trỏ và truy cập dữ liệu trong tập kết quả. Vài điểm cần lưu ý khi sử dụng đối tượng này:

- Cursor là một tập hợp các dòng kết quả.
- Bạn cần gọi phương thức `moveToFirst()` trước khi đọc dữ liệu từ Cursor.
- Bạn cần biết tên và kiểu dữ liệu của các cột.
- Con trỏ của Cursor có thể di chuyển tùy ý, bạn có thể di chuyển xuôi, ngược, nhảy đến vị trí bất kỳ trong số các dòng của Cursor. Tuy nhiên, chỉ truy cập được một dòng trong một thời điểm và chỉ đọc.

Lớp Cursor cung cấp nhiều phương thức cho phép truy cập dữ liệu hiệu quả, bao gồm: các tác vụ nhận biết vị trí, các tác vụ định vị, các tác vụ xác định cấu trúc và các getter để lấy dữ liệu. Một số phương thức:

Phương thức	Mục đích
<code>getColumnCount()</code>	Trả về số cột của Cursor.
<code>getColumnIndex(String columnName)</code>	Chỉ định tên cột, trả về chỉ số của cột đó.
<code>getColumnName(int columnIndex)</code>	Chỉ định chỉ số, tính từ 1, trả về tên của cột đó.
<code>getColumnNames()</code>	Trả về mảng tất cả tên cột của Cursor.
<code>getCount()</code>	Trả về số dòng trong Cursor.
<code>getPosition()</code>	Trả về vị trí hiện tại của con trỏ trong Cursor.
<code>moveToFirst()</code>	Di chuyển đến dòng đầu tiên trong Cursor.
<code>moveToLast()</code>	Di chuyển đến dòng cuối cùng trong Cursor.
<code>moveToNext()</code>	Di chuyển con trỏ đến dòng kế tiếp.
<code>move()</code>	Di chuyển theo offset chỉ định tính từ vị trí hiện tại.
<code>isAfterLast()</code>	Kiểm tra xem con trỏ có vượt qua dòng cuối không.
<code>get<type>()</code>	Các getter trả về trị phù hợp <type> chỉ định chứa tại cột chỉ định (bằng chỉ số) tại dòng hiện hành.

```
public List<String> getAllEmails() {
    SQLiteDatabase database = getReadableDatabase();
    Cursor cursor = database.query("address_book", new String[]{"email"},
                                   null, null, null, null, "ORDER BY email");
    LinkedList<String> emails = new LinkedList<String>();
    if (cursor.moveToFirst()) {
        do {
            emails.add(cursor.getString(0));
        } while (cursor.moveToNext());
    }
    return emails;
}
```

Các phương thức bổ sung

Mặc dù, thực thi truy vấn SQL được xem như là cách tiếp cận chuẩn, nhưng `SQLiteDatabase` cũng cung cấp một số phương thức bổ sung để thực hiện các tác vụ insert, update và delete lên cơ sở dữ liệu.

`public long insert(String table, String nullColumnHack, ContentValues values)`

Chèn vào bảng table. Tham số `nullColumnHack` đặt bằng null để ngăn values rỗng, do SQL không cho phép chèn dòng rỗng (trừ ROWID); nếu muốn chèn dòng rỗng (trừ ROWID), truyền tên một cột cho tham số này. Tham số values, kiểu ContentValues, thực chất là một map chứa các cặp khóa/trị, làm dữ liệu đầu vào (tên cột/trị chứa) cho câu truy vấn. Trả về ROWID của dòng mới chèn vào.

`public int update(String table, ContentValues values, String whereClause, String[] whereArgs)`

Cập nhật một số dòng trong bảng table, tùy phát biểu whereClause. Trả về số dòng bị ảnh hưởng.

`public int delete(String table, String whereClause, String[] whereArgs)`

Xóa một số dòng trong bảng table, tùy phát biểu whereClause. Trả về số dòng bị ảnh hưởng.

c) Lớp SQLiteOpenHelper

Thông thường, để dễ dàng quản lý các tác vụ lên cơ sở dữ liệu, ta tạo một lớp tiếp hợp (adapter) đóng gói (wrapper) các thao tác truy cập cơ sở dữ liệu phức tạp: tạo, mở, đóng, truy vấn cơ sở dữ liệu SQLite.

Lớp tiếp hợp sẽ tham chiếu một lớp công cụ, thừa kế từ lớp `android.database.sqlite.SQLiteOpenHelper`, để hỗ trợ việc tạo và quản lý phiên bản cơ sở dữ liệu. Bạn cần viết lại hai phương thức của lớp này:

`onCreate()` dùng tạo một cơ sở dữ liệu mới nếu cơ sở dữ liệu yêu cầu chưa tồn tại.

`onUpgrade()` được gọi khi phiên bản cơ sở dữ liệu thấp hơn phiên bản cung cấp cho constructor, nó cho phép ứng dụng thực hiện các bước cần thiết để nâng cấp schema của cơ sở dữ liệu.

Trong ví dụ, ta tạo một cơ sở dữ liệu tên MyDB, chứa một bảng tên contacts, có ba cột: `_id`, `name` và `email`.

Các tác vụ truy cập cơ sở dữ liệu được thực hiện bởi lớp `DBAdapter`, với sự hỗ trợ của thực thể `SQLiteDatabase` tạo từ lớp `DatabaseHelper` được cài đặt như lớp nội của `DBAdapter`.

```
public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_NAME = "name";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";
    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "contacts";
    static final int DATABASE_VERSION = 1;
```

```

static final String DATABASE_CREATE = "CREATE TABLE contacts " +
    "(_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT null, email TEXT NOT NULL);";
final Context context;
DatabaseHelper databaseHelper;
SQLiteDatabase database;

public DBAdapter(Context context) {
    this.context = context;
    databaseHelper = new DatabaseHelper(context);
}

// lớp nội hỗ trợ, thừa kế SQLiteOpenHelper
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(DATABASE_CREATE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to " + newVersion +
            ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS contacts");
        onCreate(db);
    }
} // end of DatabaseHelper

// mở cơ sở dữ liệu cho phép ghi
public DBAdapter open() throws SQLException {
    database = databaseHelper.getWritableDatabase();
    return this;
}

// đóng cơ sở dữ liệu
public void close() {
    databaseHelper.close();
}

// chèn một record đến cơ sở dữ liệu
public long insertContact(String name, String email) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NAME, name);
    initialValues.put(KEY_EMAIL, email);
    return database.insert(DATABASE_TABLE, null, initialValues);
}

// xóa một record chỉ định
public boolean deleteContact(long rowId) {
    return database.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}

// xóa toàn bộ cơ sở dữ liệu
public void deleteDatabase() {
    context.deleteDatabase(DATABASE_NAME);
}

// lấy tất cả các record
public Cursor getAllContacts() {
    return database.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME, KEY_EMAIL},
        null, null, null, null, null);
}

// lấy một record chỉ định
public Cursor getContact(long rowId) throws SQLException {
    Cursor cursor = database.query(true, DATABASE_TABLE,
        new String[] {KEY_ROWID, KEY_NAME, KEY_EMAIL},

```

```

        KEY_ROWID + "=" + rowId, null, null, null, null, null);

    if (cursor != null) {
        cursor.moveToFirst();
    }
    return cursor;
}

// cập nhật một record
public boolean updateContact(long rowId, String name, String email) {
    ContentValues args = new ContentValues();
    args.put(KEY_NAME, name);
    args.put(KEY_EMAIL, email);
    return database.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
}

```

Activity sử dụng lớp tiếp hợp trên:

```

public class DatabasesActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DBAdapter database = new DBAdapter(this);
        database.open();
        long id = database.insertContact("Karl Marx", "karlmarx@google.ru");
        id = database.insertContact("Barack Obama", "obama@whitehouse.us");
        database.close();
    }
}

```

d) Transaction

Thường gặp tình huống, nhóm tác vụ truy cập cơ sở dữ liệu phải được thực hiện như một đơn vị công việc (unit-of-work), một tác vụ trong nhóm thất bại sẽ dẫn đến kết thúc bất thường làm mất tính nhất quán của cơ sở dữ liệu. Đơn vị công việc như vậy gọi là transaction (giao tác), các tác vụ thuộc một transaction hoặc được thực hiện thành công tất cả hoặc xem như không có tác vụ nào được thực hiện.

Giả sử database là một thể hiện của lớp SQLiteDatabase, đoạn code sau minh họa cách thực hiện một transaction:

```

database.beginTransaction();
try {
    // thực hiện nhóm tác vụ truy cập cơ sở dữ liệu thuộc một transaction
    database.setTransactionSuccessful(); // hoàn tất (commit) các thay đổi đến cơ sở dữ liệu
} catch (SQLException e) {
    // xử lý exception
} finally {
    database.endTransaction();
}
}

```

Transaction được phân định giữa các phương thức: beginTransaction() và endTransaction(). Nếu nhóm tác vụ truy cập cơ sở dữ liệu thuộc transaction đều được thực hiện thành công, bạn gọi setTransactionSuccessful() để hoàn tất (commit) các thay đổi. Nếu phương thức này không được gọi, quá trình rollback không tưởng mình sẽ được gọi để phục hồi trạng thái ban đầu của cơ sở dữ liệu.

e) Cơ sở dữ liệu tạo trước

Trong ứng dụng thực tế, bạn thường *tạo trước cơ sở dữ liệu* trong thời gian thiết kế và sử dụng nó trong thời gian chạy. Bạn có thể dùng một trong các công cụ quản lý SQLite giới thiệu ở trên để tạo cơ sở dữ liệu, tạo bảng, chèn các record dữ liệu một cách trực quan. Kết quả bạn có tập tin cơ sở dữ liệu SQLite tạo trước.

- Với thiết bị thật, bạn kéo thả tập tin cơ sở dữ liệu SQLite vào thư mục /assets của project. Trong ứng dụng, viết code sao chép tập tin SQLite này đến thư mục lưu trữ chuẩn, có dạng /data/data/<package_name>/databases/.

```

try {
    String destPath = "/data/data/" + getPackageName() + "/databases";
    File file = new File(destPath);
    if (!file.exists()) {
        file.mkdirs();
        file.createNewFile();
        // sao chép mydb từ thư mục assets vào thư mục databases
        copyDB(getBaseContext().getAssets().open("mydb"), new FileOutputStream(destPath + "/MyDB"));
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}

```

```
// phương thức hỗ trợ
protected void copyDB(InputStream inputStream, OutputStream outputStream) throws IOException {
    byte[] buffer = new byte[1024];
    int length;
    while ((length = inputStream.read(buffer)) > 0) {
        outputStream.write(buffer, 0, length);
    }
    inputStream.close();
    outputStream.close();
}
```

- Với thiết bị ảo, sao chép tập tin cơ sở dữ liệu SQLite tạo sẵn vào thư mục /data/data/<package_name>/databases/, sử dụng File Explorer của ADM.

5. Tổ chức dữ liệu cho ứng dụng

Các hướng dẫn sau đây giúp bạn tổ chức dữ liệu cho ứng dụng của bạn một cách hợp lý và dễ truy xuất, bằng cách dùng cơ sở dữ liệu SQLite.

a) Tổ chức tập các POJO

Các đối tượng mà ứng dụng quan tâm thường là tập các POJO có quan hệ với nhau. Ví dụ, một Category có nhiều Item:

```
public class Item {
    private String name;
    private String image;
    private String description;
    // constructors và getters/setters
}

public class Category {
    private String image;
    private String name;
    private ArrayList<Item> items = new ArrayList<Item>();
    // constructors và getters/setters
}
```

b) Trung tâm cung cấp dữ liệu

Bạn nên xây dựng một đối tượng đơn nhất, chuyên cung cấp (hoặc lưu trữ) dữ liệu cho ứng dụng, có các đặc điểm sau:

- Đối tượng duy nhất, theo design pattern Singleton. Như vậy, đối tượng này chỉ nạp dữ liệu đúng một lần. Mặt khác, chỉ tồn tại một đối tượng duy nhất khi ứng dụng còn trong bộ nhớ, dùng cho các activity, fragment.
- Dữ liệu được nạp từ nguồn dữ liệu vào tập các POJO có quan hệ với nhau, dùng lớp DBAdapter do ta viết.
- Tùy theo nhu cầu dữ liệu của ứng dụng, viết một số getter truy cập dữ liệu được nạp và trả về mảng chuỗi (dùng cho UI), mảng các POJO hoặc POJO cụ thể (dùng cho phần Model).

```
public class DataCenter {
    private static DataCenter singleton;
    private ArrayList<Category> list = new ArrayList<Category>();

    // Singleton design pattern
    private DataCenter(Context appContext) {
        load(appContext);
    }

    public static DataCenter get(Context context) {
        if (singleton == null) {
            singleton = new DataCenter(context.getApplicationContext());
        }
        return singleton;
    }

    // nạp dữ liệu từ nguồn dữ liệu
    private void load(Context context) {
        DBAdapter database = new DBAdapter(context);
        database.open();
        list = database.getAll();
        database.close();
    }

    // các getter cung cấp dữ liệu cho UI và cho Model
    // UI: list tên các Category, tap một mục trả về list tên các Item
    // phương thức lấy list tên các Item
    public List<String> getItemNames(int parentPosition) {
        List<String> itemNames = new ArrayList<String>();
    }
}
```



```

    List<Item> items = list.get(parentPosition).getItems();
    for (Item item : items) {
        itemNames.add(item.getName());
    }
    return itemNames;
}

// Model: dữ liệu được truy cập khi chọn một mục trong list các tên Item
// phương thức lấy Item khi chọn một mục trong list các tên Item
public Item getItem(int parentPosition, int childPosition) {
    return list.get(parentPosition).getItems().get(childPosition);
}
}

```

c) Nguồn dữ liệu

Nguồn dữ liệu cho trung tâm dữ liệu trên thường là tập tin cơ sở dữ liệu SQLite tạo trước, truy cập thông qua lớp DBAdapter.

```

public class DBAdapter {
    static final String DATABASE_NAME = "mydb.db"; // tên tập tin cơ sở dữ liệu
    static final String CATEGORY_TABLE = "categories"; // tên bảng
    static final String ITEM_TABLE = "items"; // tên bảng (một categories có nhiều items)
    static final int DATABASE_VERSION = 1;
    final Context context;
    DatabaseHelper databaseHelper;
    SQLiteDatabase database;

    public DBAdapter(Context context) {
        this.context = context;
        databaseHelper = new DatabaseHelper(context);
    }

    public DBAdapter open() throws SQLException {
        database = dbHelper.getReadableDatabase();
        return this;
    }

    public void close() {
        databaseHelper.close();
    }

    public ArrayList<Category> getAll () {
        ArrayList<Category> results = new ArrayList<Category>();
        Cursor cCursor = database.query(CATEGORY_TABLE,
            new String[] {"id", "name", "image"}, // các columns trong bảng categories
            null, null, null, null, null);
        if (cCursor != null) {
            cCursor.moveToFirst();
            while (!cCursor.isAfterLast()) {
                long id = cCursor.getLong(cCursor.getColumnIndex("id"));
                Category category = new Category();
                category.setName(cCursor.getString(cCursor.getColumnIndex("name")));
                category.setImage(cCursor.getString(cCursor.getColumnIndex("image")));
                Cursor iCursor = database.query(ITEM_TABLE,
                    new String[] {"id", "name", "image", "description"},
                    "cateid=" + id, null, null, null, null);
                if (iCursor != null) {
                    iCursor.moveToFirst();
                    while (!iCursor.isAfterLast()) {
                        Item item = new Item();
                        item.setName(iCursor.getString(iCursor.getColumnIndex("name")));
                        item.setImage(iCursor.getString(iCursor.getColumnIndex("image")));
                        item.setDescription(iCursor.getString(iCursor.getColumnIndex("description")));
                        category.addItem(item);
                        iCursor.moveToNext();
                    }
                }
                results.add(category);
                cCursor.moveToNext();
            }
        }
        return results;
    }
}

```

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override public void onCreate(SQLiteDatabase db) { }
    @Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { }
}
}
```

Lưu trữ bền vững một cấu trúc các đối tượng phức tạp yêu cầu ánh xạ giữa cấu trúc đối tượng đó với cơ sở dữ liệu (ORM – Object Relationship Mapping). Android không cung cấp trực tiếp điều này, tuy nhiên bạn có thể dùng một số thư viện hỗ trợ ORM trên nền tảng Android như Ormlite (http://ormlite.com/sqlite_java_android_orm.shtml).

Android Backup Service

Để hỗ trợ nâng cấp thiết bị, Android cho phép sao lưu (backup) lên cloud danh sách các ứng dụng tải về và cấu hình hệ thống, để phục hồi (restore) trở lại khi người dùng nâng cấp lên thiết bị mới. Tuy nhiên theo mặc định, dữ liệu của ứng dụng không được sao lưu kèm theo tiến trình này. Người phát triển ứng dụng phải đăng ký với Android Backup Service và cung cấp một cài đặt tùy biến cho `android.app.backup.BackupAgent` trong ứng dụng của họ để tương tác với Android Backup Service.

1. Android Backup Service

Các ứng dụng có hỗ trợ sao lưu dữ liệu phải đăng ký với Android Backup Service thông qua trang đăng ký tại:

<https://developer.android.com/google/backup/signup.html>

Trước hết bạn phải chấp nhận các điều khoản của dịch vụ và cung cấp tên gói ứng dụng trong phần Application package name. Sau khi đăng ký với Android Backup Service, bạn sẽ nhận được một khóa API key duy nhất. Khóa này được dùng để tích hợp vào tập tin manifest của ứng dụng, như sau:

```
<application>
    <meta-data android:name="com.google.android.backup.api_key"
        android:value="AEd2zEfV3XzHSHqKxoIJAYVZ6ZWnz4W_AmA" />
</application>
```

2. Backup Agent

Android Backup Service không tự động sao lưu và phục hồi thư mục data của ứng dụng. Nó cần ứng dụng cung cấp một cài đặt cho Backup Agent, để xử lý sao lưu và phục hồi tùy ứng dụng (application-specific). Cách đơn giản nhất là thừa kế lớp `android.app.backup.BackupAgentHelper`. Khi đó, bạn viết lại phương thức `onCreate()` của lớp hỗ trợ và chỉ định danh sách các thứ cần sao lưu bằng cách dùng các lớp hỗ trợ sau:

`FileBackupHelper`

Quản lý sao lưu và phục hồi một danh sách các tập tin chỉ định trong thư mục data của ứng dụng.

`SharedPreferencesBackupHelper`

Quản lý sao lưu và phục hồi các tập tin shared preferences.

Nếu phải xử lý các kiểu dữ liệu khác, thừa kế lớp `BackupAgent` và cung cấp cài đặt các tác vụ sao lưu và phục hồi cho riêng kiểu dữ liệu đó.

```
public class CloudBackupAgent extends BackupAgentHelper {
    @Override public void onCreate() {
        super.onCreate();
        addHelper("preferences", new SharedPreferencesBackupHelper(this,
            getPackageName() + "_preferences", "user"));
        addHelper("files", new FileBackupHelper(this, "records.db"));
    }
}
```

Code trên dùng `SharedPreferencesBackupHelper` để quản lý sao lưu và phục hồi hai shared preference: shared preference mặc định và một shared preference tên "user". Code trên cũng dùng `FileBackupHelper` để quản lý sao lưu và phục hồi tập tin `records.db`.

Cài đặt cho Backup Agent phải được khai báo trong tập tin manifest của ứng dụng cho Android Backup Service thấy được. Điều này thực hiện nhờ thuộc tính `android:backupAgent` của element `<application>`:

```
<application android:backupAgent=".CloudBackupAgent" ></application>
```

Để chắc rằng tất cả dữ liệu của ứng dụng được sao lưu đúng với mọi thay đổi, ứng dụng Android Backup Service thực hiện sao lưu, bằng cách gọi phương thức `BackupManager.dataChanged`. Yêu cầu sao lưu sẽ được lập lịch và thực hiện bằng cách gọi các phương thức được cài đặt cho Backup Agent của ứng dụng.

Cài đặt của Backup Agent có thể được kiểm tra thông qua công cụ dòng lệnh `bmgr`:

- Dùng ADB shell, sinh một yêu cầu sao lưu `adb shell bmgr backup`
- Ép Android Backup Service thực hiện sao lưu `adb shell bmgr run`

Storage Access Framework

Những năm gần đây việc lưu trữ dữ liệu của người dùng bằng dịch vụ lưu trữ từ xa, dựa trên Cloud (điện toán đám mây) được áp dụng rộng rãi. Điều này được thúc đẩy bởi:

- Các thiết bị di động cung cấp kết nối Internet tốc độ cao, liên tục, cho phép chuyển dữ liệu nhanh chóng.
- Các thiết bị di động vẫn còn hạn chế về dung lượng lưu trữ so với thiết bị để bàn cố định.

Để đáp ứng nhu cầu thực tế này, Android giới thiệu Storage Access Framework từ Android 4.4, giúp giảm bớt đáng kể việc tích hợp lưu trữ dữ liệu dựa trên Cloud vào ứng dụng Android.

Storage Access Framework cung cấp một cách thức giúp người dùng thông qua giao diện ứng dụng Android dễ dàng duyệt, chọn, xóa và tạo các tập tin lưu trữ bởi các dịch vụ lưu trữ.

Các dịch vụ lưu trữ này còn được gọi là các nhà cung cấp tài liệu (document providers). Các nhà cung cấp tài liệu này sử dụng dịch vụ lưu trữ dựa trên Cloud, nổi bật là Box, Google Drive. Ngoài ra, các nhà cung cấp tài liệu cũng cung cấp truy cập đến tài liệu lưu trữ nội bộ trên thiết bị như Downloads, Internal storage.

Android 4.4 giới thiệu một tập các intent mới được thiết kế để tích hợp các tính năng của Storage Access Framework vào ứng dụng Android. Khi phát, các intent này hiển thị giao diện (picker) giúp người dùng lựa chọn nhà cung cấp tài liệu và trả về kết quả lựa chọn trong phương thức onActivityResult() của activity phát intent.

Các intent của Storage Access Framework bao gồm:

- ACTION_OPEN_DOCUMENT, cho phép truy cập giao diện dùng lựa chọn các tập tin của nhà cung cấp tài liệu được cấu hình trên thiết bị mà bạn lựa chọn. Các tập tin được chọn sẽ được truyền cho ứng dụng dưới dạng một đối tượng Uri.

- ACTION_CREATE_DOCUMENT, cho phép người dùng lựa chọn một nhà cung cấp tài liệu, một vị trí tại nơi lưu trữ của nhà cung cấp và một tên cho tập tin lưu trữ mới. Tập tin sẽ được tạo bởi Storage Access Framework tại nơi lưu trữ và Uri của nó sẽ được trả về ứng dụng để xử lý tiếp nếu cần.

Có được Uri của tập tin, bạn có thể thao tác dễ dàng tập tin. Chú ý, intent được thiết lập type là kiểu MIME của tập tin.

Khi ứng dụng đạt được quyền truy cập vào tập tin thông qua Storage Access Framework, truy cập vẫn hiệu lực cho đến khi thiết bị đang truy cập khởi động lại. Quyền truy cập bền vững cần thiết có thể lấy từ Uri của tập tin:

```
final int takeFlags = intent.getFlags()
    & (Intent.FLAG_GRANT_READ_URI_PERMISSION
    | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
getContentResolver().takePersistableUriPermission(fileUri, takeFlags);
```

Khi ứng dụng lấy được quyền truy cập tập tin và giả sử Uri đã được ứng dụng lưu, người dùng có thể tiếp tục truy cập tập tin sau khi thiết bị khởi động lại mà không cần chọn lại tập tin. Sau đó, nếu ứng dụng không cần truy cập nữa, giải phóng bằng:

```
final int releaseFlags = intent.getFlags()
    & (Intent.FLAG_GRANT_READ_URI_PERMISSION
    | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
getContentResolver().releasePersistableUriPermission(fileUri, releaseFlags);
```

Để thực hiện ví dụ, bạn cần một tài khoản Google Drive trên thiết bị và thiết lập minimum SDK là API Level 19.

Giao diện gồm ba nút: New, Open, Save và EditText để nhập nội dung cho tập tin.

```
public class StorageActivity extends ActionBarActivity {
    private static final int CREATE_REQUEST_CODE = 100;
    private static final int OPEN_REQUEST_CODE = 101;
    private static final int SAVE_REQUEST_CODE = 102;
    private static EditText mEditText;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_storage);
        mEditText = (EditText) findViewById(R.id.text_id);
    }

    public void newFile(View view) { // handler cho nút New
        Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT)
            .addCategory(Intent.CATEGORY_OPENABLE)
            .setType("text/plain")
            .putExtra(Intent.EXTRA_TITLE, "my_file.txt");
        startActivityForResult(intent, CREATE_REQUEST_CODE);
    }

    public void saveFile(View view) { // handler cho nút Save
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT)
            .addCategory(Intent.CATEGORY_OPENABLE)
            .setType("text/plain");
        startActivityForResult(intent, SAVE_REQUEST_CODE);
    }

    public void openFile(View view) { // handler cho nút Open
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT)
            .addCategory(Intent.CATEGORY_OPENABLE)
            .setType("text/plain");
        startActivityForResult(intent, OPEN_REQUEST_CODE);
    }

    @Override public void onActivityResult(int requestCode, int resultCode, Intent resultData) {
        Uri uri;
        if (resultCode == Activity.RESULT_OK) {
            switch (requestCode) {
```

```

        case CREATE_REQUEST_CODE:
            if (resultData != null) mEditText.setText("");
            break;
        case SAVE_REQUEST_CODE:
            if (resultData != null) {
                uri = resultData.getData();
                writeFileContent(uri);
            }
            break;
        case OPEN_REQUEST_CODE:
            if (resultData != null) {
                uri = resultData.getData();
                try {
                    String content = readFileContent(uri);
                    mEditText.setText(content);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

// các phương thức hỗ trợ thao tác trên tập tin
private void writeFileContent(Uri uri) {
    try {
        ParcelFileDescriptor pfd = this.getContentResolver().openFileDescriptor(uri, "w");
        FileOutputStream out = new FileOutputStream(pfd.getFileDescriptor());
        String textContent = mEditText.getText().toString();
        out.write(textContent.getBytes());
        out.close();
        pfd.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private String readFileContent(Uri uri) throws IOException {
    InputStream in = getContentResolver().openInputStream(uri);
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
    StringBuilder sb = new StringBuilder();
    String s;
    while ((s = reader.readLine()) != null) sb.append(s).append('\n');
    in.close();
    return sb.toString();
}
}

```

Messaging

SMS

Bạn có thể gửi các thông điệp SMS từ trong ứng dụng của bạn bằng cách lập trình, hoặc bằng cách dùng ứng dụng Messaging có sẵn. Với yêu cầu đầu ta dùng SmsManager và với yêu cầu thứ hai ta dùng Intent.

Khi thử nghiệm, chạy hai AVD, chạy chương trình trên một AVD rồi gửi SMS cho AVD còn lại với số phone là port đang chạy AVD (xem trên title bar của AVD).

1. SmsManager

SmsManager quản lý các tác vụ SMS như gửi SMS đến một thiết bị mobile cho trước. Tạo SmsManager bằng cách gọi phương thức static SmsManager.getDefault() như sau:

```
SmsManager manager = SmsManager.getDefault();
```

Khi đã có SmsManager, dùng phương thức sendDataMessage() của nó để gửi SMS đến số mobile chỉ định:

```
manager.sendTextMessage("0908123456", null, "SMS text", null, null);
```

Có thể viết riêng thành một phương thức hỗ trợ như sau:

```
protected void sendSMS(String phoneNo, String message) {
    try {
        SmsManager manager = SmsManager.getDefault();
        manager.sendTextMessage(phoneNo, null, message, null, null);
    } catch (Exception e) {
        Toast.makeText(getApplicationContext(), "SMS failed.", Toast.LENGTH_LONG).show();
        e.printStackTrace();
    }
}
```

Trong tập tin manifest, cấp thêm quyền cho phép gửi SMS:

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

SmsManager còn nhiều phương thức quan trọng khác:

ArrayList<String> divideMessage(String text)

Phương thức này chia văn bản thông điệp thành vài mảnh, không lớn hơn kích thước tối đa của một thông điệp SMS.

static SmsManager getDefault()

Phương thức này được dùng để lấy thực thể mặc định của SmsManager.

void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)

Phương thức này được dùng để gửi dữ liệu SMS đến một port ứng dụng.

void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)

Gửi một SMS dạng văn bản gồm nhiều phần.

void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Gửi SMS dạng văn bản. scAddress là địa chỉ trung tâm dịch vụ, dùng null cho SMSC mặc định. sentIntent là PendingIntent triệu gọi khi thông điệp đã gửi. deliveryIntent là PendingIntent triệu gọi khi thông điệp đã chuyển.

2. Trạng thái gửi SMS

Có hai giai đoạn khi gửi SMS:

- SENT: thiết bị đã gửi SMS đến SMSC (Short Message Service Center), SMSC xác nhận đã nhận SMS.

- DELIVERED: SMSC đã chuyển SMS đến thiết bị nhận (hoặc đến SMSC khác) và gửi SMS-DELIVER cho thiết bị gửi.

Để xác định trạng thái gửi SMS, ta dùng PendingIntent.

Như đã thảo luận, PendingIntent đóng gói hành động sẽ được thực hiện trong tương lai. Ứng dụng của bạn truyền một ý muốn chờ được xử lý cho ứng dụng khác (NotificationManager, AlarmManager, AppWidgetManager, ...) và cho phép ứng dụng đó thực hiện hành động mong muốn với quyền tương đương ứng dụng của bạn.

Tùy theo kết quả gửi SMS của tiến trình gửi, PendingIntent được truyền như tham số của các phương thức gửi SMS. Ví dụ sau có hai PendingIntent, tương ứng với hai giai đoạn chuyển SMS, và bạn phải đăng ký hai BroadcastReceiver để chặn bắt chúng.

```
public class SMSActivity extends Activity {
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    PendingIntent sentPI, deliveredPI;
    BroadcastReceiver smsSentReceiver, smsDeliveredReceiver;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // tạo hai PendingIntent
        sentPI = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
        deliveredPI = PendingIntent.getBroadcast(this, 0, new Intent(DELIVERED), 0);
    }
}
```



```

@Override public void onResume() {
    super.onResume();
    // cài đặt cho BroadcastReceiver chặn PendingIntent khi SMS đã gửi (SENT)
    smsSentReceiver = new BroadcastReceiver() {
        @Override public void onReceive(Context arg0, Intent arg1) {
            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS sent", Toast.LENGTH_SHORT).show(); break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                    Toast.makeText(getBaseContext(), "Generic failure", Toast.LENGTH_SHORT).show(); break;
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                    Toast.makeText(getBaseContext(), "No service", Toast.LENGTH_SHORT).show(); break;
                case SmsManager.RESULT_ERROR_NULL_PDU:
                    Toast.makeText(getBaseContext(), "Null PDU", Toast.LENGTH_SHORT).show(); break;
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                    Toast.makeText(getBaseContext(), "Radio off", Toast.LENGTH_SHORT).show(); break;
            }
        }
    };
    // cài đặt BroadcastReceiver chặn PendingIntent khi SMS đã chuyển (DELIVERED)
    smsDeliveredReceiver = new BroadcastReceiver() {
        @Override public void onReceive(Context arg0, Intent arg1) {
            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS delivered", Toast.LENGTH_SHORT).show(); break;
                case Activity.RESULT_CANCELED:
                    Toast.makeText(getBaseContext(), "SMS not delivered", Toast.LENGTH_SHORT).show(); break;
            }
        }
    };
    // đăng ký hai BroadcastReceiver trên
    registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
    registerReceiver(smsSentReceiver, new IntentFilter(SENT));
}

@Override public void onPause() {
    super.onPause();
    unregisterReceiver(smsSentReceiver);
    unregisterReceiver(smsDeliveredReceiver);
}

public void onClick(View v) {
    sendSMS("5556", "Hello my friends!");
}

private void sendSMS(String phoneNumber, String message) {
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
}
}

```

3. Gửi SMS bằng Messaging

Bạn có thể dùng ứng dụng built-in Messaging của Android để gửi SMS bằng cách gửi Intent đến nó. Phần sau giải thích các phần của đối tượng Intent cần để gửi SMS.

- Action: bạn dùng action android.content.Intent.ACTION_VIEW để khởi động ứng dụng Messaging built-in cài đặt trên thiết bị Android. Sau đây là cú pháp đơn giản tạo intent với action ACTION_VIEW:

```
Intent smsIntent = new Intent(android.content.Intent.ACTION_VIEW);
```

- Data/Type: để gửi một SMS bạn dùng phương thức setData() chỉ định smsto: URI và phương thức setType() chỉ định kiểu MIME là vnd.android-dir/mms-sms.

```

smsIntent.setData(Uri.parse("smsto:"));
smsIntent.setType("vnd.android-dir/mms-sms");

```

- Extras: danh sách số phone và thông điệp SMS được thiết lập bằng extras như sau:

```

smsIntent.putExtra("address", new String("0903759412;0908995899"));
smsIntent.putExtra("sms_body", "Test SMS to Alice");

```

Bạn có thể chỉ định một hay nhiều số phone trong một chuỗi tách biệt bởi dấu (;).

Có thể viết thành một phương thức hỗ trợ như sau:

```

protected void sendSMS(String phoneNo, String message) {
    Intent smsIntent = new Intent(Intent.ACTION_VIEW);
    smsIntent.setData(Uri.parse("smsto:"));
}

```

```

smsIntent.setType("vnd.android-dir/mms-sms");
smsIntent.putExtra("address", phoneNo);
smsIntent.putExtra("sms_body", message);
try {
    startActivity(smsIntent);
    finish();
} catch (android.content.ActivityNotFoundException ex) {
    Toast.makeText(MainActivity.this, "SMS failed, please try again.", Toast.LENGTH_SHORT).show();
}
}

```

Không cần thêm quyền cho phép gửi SMS.

4. Đáp ứng khi nhận SMS

Khi nhận được SMS bạn có thể cần xử lý SMS đó, chẳng hạn bạn muốn chương trình thực hiện một hành động khi SMS chỉ định được nhận. Ví dụ thực tế, bạn cần xác định vị trí thiết bị bằng cách gửi SMS đến nó, thiết bị sẽ tự động gửi SMS chứa vị trí ngược trở lại cho bạn.

Trong tập tin manifest, cấp thêm quyền cho phép nhận và đọc SMS (và cho phép gửi nếu cần).

Khi thông điệp SMS được gửi đến, ứng dụng của bạn và ứng dụng built-in Messaging của Android đều nhận SMS (đụng độ intent). Để ngăn ứng dụng built-in Messaging nhận SMS, intent filter của receiver trong ứng dụng của bạn cần có độ ưu tiên cao hơn, thiết lập bằng thuộc tính android:priority.

```

<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />

<application>
    <receiver android:name=".SMSReceiver">
        <intent-filter android:priority="100">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
    </receiver>
</application>

```

Tạo BroadcastReceiver chặn các SMS gửi đến. Ý tưởng là khi SMS vào, phương thức onReceive() sẽ được kích hoạt, thực hiện công việc chỉ định.

Thông điệp SMS đóng gói thành một Bundle được chứa trong một đối tượng Intent. Bundle đó là một mảng các Object có định dạng PDU (Protocol Data Unit). Nếu thông điệp SMS chứa ít hơn 160 ký tự, mảng chỉ có một phần tử. Nếu nhiều hơn, thông điệp sẽ chia thành nhiều thông điệp nhỏ hơn và lưu thành nhiều phần tử của mảng.

Phương thức createFromPdu() dùng lấy thông điệp, sau đó dùng getOriginatingAddress() và getMessageBody() để lấy các thông tin chi tiết về SMS nhận được.

```

public class SMSReceiver extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        // lấy thông điệp SMS chuyển đến
        Bundle bundle = intent.getExtras();
        SmsMessage[] messages = null;
        String s = "SMS from ";
        if (bundle != null) {
            // lấy thông điệp SMS nhận được
            Object[] pdus = (Object[]) bundle.get("pdus");
            messages = new SmsMessage[pdus.length];
            for (int i = 0; i < messages.length; ++i) {
                messages[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                if (i == 0) {
                    // lấy address/phone number của người gửi
                    s += messages[i].getOriginatingAddress() + ": ";
                }
                // lấy nội dung thông điệp
                s += messages[i].getMessageBody().toString();
            }
            // gửi intent để đẩy activity lên bề mặt, chuẩn bị nhận intent quảng bá
            // trong tập tin manifest thiết lập thuộc tính android:launchMode cho activity là singleTask
            Intent mainActivityIntent = new Intent(context, SMSActivity.class);
            mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(mainActivityIntent);

            // gửi intent quảng bá để chuyển SMS nhận được cho activity
            // activity phải đăng ký broadcast receiver với intent filter chặn các intent SMS_RECEIVED_ACTION
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("SMS_RECEIVED_ACTION");
            broadcastIntent.putExtra("sms", s);
            context.sendBroadcast(broadcastIntent);
        }
    }
}

```

```
// chặn không cho thông điệp SMS đến các receiver có độ ưu tiên thấp hơn (nếu cần)
this.abortBroadcast();
}
}
```

Telephony

1. Thực hiện cuộc gọi

Bạn có thể thực hiện cuộc gọi từ trong ứng dụng. Ví dụ, người dùng đang sử dụng ứng dụng của bạn và cần thực hiện một cuộc gọi; trong trường hợp này, ứng dụng của bạn có thể quay số điện thoại một cách trực tiếp.

Để quay số điện thoại trực tiếp từ trong ứng dụng, ta dùng intent có các thành phần như sau:

Action android.content.Intent.ACTION_DIAL. Nếu cần gọi ngay, không chờ người dùng nhấn nút gọi, dùng android.content.Intent.ACTION_CALL thay thế (cần quyền android.permission.CALL_PHONE).

Data chỉ định số điện thoại với "tel:+"

```
public class MainActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String phoneNumber= "0903759412";
        Intent intent = new Intent(android.content.Intent.ACTION_DIAL, Uri.parse("tel:+" + phoneNumber));
        startActivity(intent);
    }
}
```

2. Quản lý trạng thái gọi

Chức năng gọi điện của một thiết bị Android luôn ở một trong 3 trạng thái: idle (không thực hiện cuộc gọi nào), ringing (có cuộc gọi đến), off-hook (người dùng trả lời cuộc gọi). Để kiểm soát hay thay đổi trạng thái gọi của điện thoại, dùng lớp con của PhoneStateListener. Để xác định trạng thái của điện thoại thay đổi, ta viết lại phương thức onCallStateChanged(). Chú ý rằng ta có thể lấy được số điện thoại của người gọi đến và có thể thực hiện một hành động như gửi trả tin nhắn.

```
public class PhoneReceiver extends PhoneStateListener {
    Context context;
    public PhoneReceiver(Context context) {
        this.context = context;
    }

    @Override public void onCallStateChanged(int state, String incomingNumber) {
        super.onCallStateChanged(state, incomingNumber);
        Toast.makeText(context, "onCallStateChanged state=" + state + ", incomingNumber=" + incomingNumber, Toast.LENGTH_SHORT).show();

        switch (state) {
            case TelephonyManager.CALL_STATE_IDLE:
                Toast.makeText(context, "idle", Toast.LENGTH_LONG).show(); break;
            case TelephonyManager.CALL_STATE_RINGING:
                Toast.makeText(context, "ringing", Toast.LENGTH_LONG).show(); break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                Toast.makeText(context, "offhook", Toast.LENGTH_LONG).show();
        }
    }
}
```

Dùng listener trên trong activity, cần cấp quyền android.permission.READ_PHONE_STATE trong tập tin manifest.

```
public class MainActivity extends Activity {
    TelephonyManager manager;
    PhoneReceiver listener;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listener = new PhoneReceiver(this);
        manager = ((TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE));
    }

    @Override public void onResume() {
        super.onResume();
        manager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
    }

    // loại bỏ listener
    @Override public void onPause() {
        super.onPause();
    }
}
```

```

        manager.listen(listener, PhoneStateListener.LISTEN_NONE);
    }
}

```

Ví dụ trên sử dụng lớp PhoneStateListener để kiểm soát trạng thái gọi. Tuy nhiên, nếu ứng dụng của bạn không đang chạy thì bạn sẽ không thể kiểm soát sự thay đổi trạng thái gọi. Do đó, tốt hơn bạn nên dùng BroadcastReceiver. Chỉ cần ứng dụng còn cài đặt trên thiết bị, dù không chạy, bạn vẫn có thể kiểm soát sự thay đổi trạng thái gọi.

Bạn tạo lớp PhoneStateReceiver thừa kế BroadcastReceiver. Trong lớp này, ta tạo lớp nội PhoneListener thừa kế lớp PhoneStateListener. Tiếp theo, viết lại phương thức onReceive() của lớp BroadcastReceiver, trong đó ta gọi phương thức listen() của lớp TelephonyManager để lắng nghe sự thay đổi trạng thái gọi với đối tượng listener là PhoneStateListener.

```

public class PhoneStateReceiver extends BroadcastReceiver {
    TelephonyManager manager;
    PhoneReceiver listener;
    static boolean alreadyListening = false;

    @Override public void onReceive(Context context, Intent intent) {
        listener = new PhoneReceiver(context);
        manager = ((TelephonyManager) context.getSystemService(Context.TELEPHONY_SERVICE));
        // không đăng ký listener quá một lần
        if (!alreadyListening) {
            manager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
            alreadyListening = true;
        }
    }

    class PhoneReceiver extends PhoneStateListener {
        Context context;
        public PhoneReceiver(Context context) {
            this.context = context;
        }

        @Override public void onCallStateChanged(int state, String incomingNumber) {
            super.onCallStateChanged(state, incomingNumber);
            switch (state) {
                case TelephonyManager.CALL_STATE_IDLE:
                    Toast.makeText(context, "idle", Toast.LENGTH_LONG).show(); break;
                case TelephonyManager.CALL_STATE_RINGING:
                    Toast.makeText(context, "ringing: " + incomingNumber, Toast.LENGTH_LONG).show(); break;
                case TelephonyManager.CALL_STATE_OFFHOOK:
                    Toast.makeText(context, "offhook", Toast.LENGTH_LONG).show();
            }
        }
    }
}

```

Khai báo BroadcastReceiver trong tập tin manifest, chặn android.intent.action.PHONE_STATE:

```

<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<application>
    <receiver android:name=".PhoneStateReceiver">
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" />
        </intent-filter>
    </receiver>
</application>

```

3. Khóa cuộc gọi đi

Dùng BroadcastReceiver, bạn có thể xác định có cuộc gọi đi trong phương thức onReceive(). Để khóa cuộc gọi đi với số điện thoại chỉ định, bạn chỉ cần gọi phương thức setResultData() và truyền cho nó tham số null.

```

@Override public void onReceive(Context context, Intent intent) {
    String outgoingNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER).toString();
    if (outgoingNumber.contentEquals("0908123456")) {
        setResultData(null);
        Toast.makeText(context, "This call is not allowed!", Toast.LENGTH_LONG).show();
    }
}

```

Trong tập tin manifest, khai báo BroadcastReceiver chặn android.intent.action.NEW_OUTGOING_CALL và cấp quyền android.permission.PROCESS_OUTGOING_CALLS.

```

<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<application>
    <receiver android:name=".OutgoingCallsReceiver" >

```

```
<intent-filter android:priority="0">
  <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
</intent-filter>
</receiver>
</application>
```

4. Tự động trả lời cuộc gọi đến

Để nhận cuộc gọi đến, bạn cần lớp BroadcastReceiver để kiểm soát sự thay đổi trạng thái của điện thoại. Khi phát hiện cuộc gọi đến, trong phương thức onReceive() của lớp này, gửi đối tượng Intent để mô phỏng hành động ấn nút trả lời trên headset.

```
public class IncomingCallsReceiver extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        if (!intent.getAction().equals("android.intent.action.PHONE_STATE")) return;
        String extraState = intent.getStringExtra(TelephonyManager.EXTRA_STATE);
        if (extraState.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
            String incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER);
            if (incomingNumber.contentEquals("0903759412")) {
                // giả lập nhấn nút headset
                Intent intent = new Intent(Intent.ACTION_MEDIA_BUTTON);
                intent.putExtra(Intent.EXTRA_KEY_EVENT,
                    new KeyEvent(KeyEvent.ACTION_UP, KeyEvent.KEYCODE_HEADSETHOOK));
                context.sendOrderedBroadcast(i, "android.permission.CALL_PRIVILEGED");
            }
        }
    }
}
```

5. Chuyển sang chế độ Airplane

Nếu muốn ngắt hết tất cả các kết nối không dây trên thiết bị Android, ta có thể lập trình chuyển sang chế độ Airplane. Để bật chế độ Airplane, dùng phương thức putInt() của lớp Settings.System và truyền vào một đối tượng ContentResolver, cùng với hằng AIRPLANE_MODE_ON và một một trị để xác định bật (1) hay tắt (0) chế độ Airplane. Tiếp theo, gửi quảng bá intent ACTION_AIRPLANE_MODE_CHANGED với extras state là true (để bật) hoặc false (để tắt).

Cần cấp quyền android.permission.WRITE_SETTINGS.

```
private void SetAirplaneMode(boolean enabled){
    Settings.System.putInt(getContentResolver(), Settings.System.AIRPLANE_MODE_ON, enabled ? 1 : 0);
    Intent intent = new Intent(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    intent.putExtra("state", enabled);
    sendBroadcast(intent);
}
```

Để kiểm tra xem thiết bị có đang trong chế độ Airplane hay không:

```
public boolean isAirplaneModeOn(){
    return Settings.System.getInt(getContentResolver(), Settings.System.AIRPLANE_MODE_ON, 0) != 0;
}
```

Nếu cần kiểm soát sự thay đổi của chế độ Airplane, tạo lớp BroadcastReceiver. Phương thức onReceive() của nó sẽ hoạt động khi chế độ Airplane thay đổi:

```
public class AirplaneModeReceiver extends BroadcastReceiver {
    @Override public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Service state changed", Toast.LENGTH_LONG).show();
    }
}
```

Sau đó đăng ký AirplaneModeReceiver chặn android.intent.action.SERVICE_STATE:

```
<receiver android:name=".AirplaneModeReceiver"
    android:enabled="true" >
    <intent-filter>
        <action android:name="android.intent.action.SERVICE_STATE" />
    </intent-filter>
</receiver>
```

6. Thẻ SIM và số IMEI

Ứng dụng của bạn có thể sẽ cần biết khi người dùng thay đổi thẻ SIM (Subscriber Identity Module). Ví dụ, khi ứng dụng được chạy lần đầu tiên, nó lưu ID của thẻ SIM. Mỗi khi điện thoại thay đổi từ AirplaneModeON sang AirplaneModeOFF, ứng dụng kiểm tra ID của thẻ SIM với cái đã lưu. Nếu phát hiện ID thay đổi, ứng dụng sẽ tự động gửi một SMS đến một điện thoại khác để thông báo cho bạn.

Ngoài thẻ SIM, IMEI cũng là một thông tin hữu ích. IMEI (International Mobile Equipment Identity) là dãy số dùng để nhận dạng một thiết bị di động. Xem IMEI bằng cách nhập mã *#06#.

Để lấy được các thông tin trên, ta dùng TelephonyManager và cần quyền android.permission.READ_PHONE_STATE.

```
public class MainActivity extends Activity {
```



```

@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TelephonyManager manager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
    // lấy ID thẻ SIM
    String simID = manager.getSimSerialNumber();
    if (simID != null) Toast.makeText(this, "SIM card ID: " + simID, Toast.LENGTH_LONG).show();
    // lấy số phone đang dùng
    String telNumber = manager.getLine1Number();
    if (telNumber != null) Toast.makeText(this, "Phone number: " + telNumber, Toast.LENGTH_LONG).show();
    // lấy số IMEI
    String IMEI = manager.getDeviceId();
    if (IMEI != null) Toast.makeText(this, "IMEI number: " + IMEI, Toast.LENGTH_LONG).show();
}
}

```

7. Hiển thị Call Log

Tất cả thông tin cuộc gọi của điện thoại đều được ghi vào content provider call_log. Do đó, nếu cần thông tin về cuộc gọi đến, cuộc gọi đi, cuộc gọi nhớ, ... bạn có thể truy cập content provider để lấy.

Để lấy thông tin cuộc gọi, ta truy xuất content provider call_log, duyệt kết quả trên đối tượng Cursor trả về. Ví dụ sau đây lấy tất cả thông tin cuộc gọi và sắp xếp theo ngày. Sau đó nó sẽ in ra thông tin của từng cuộc gọi: ID, số phone, ngày gọi, kiểu gọi, cuộc gọi có được người dùng biết đến hay chưa.

Để truy xuất content provider call_log, ta cần thêm quyền android.permission.READ_CONTACTS.

```

public class MainActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getCallLogs();
    }

    private void getCallLogs() {
        Cursor cursor = null;
        try {
            cursor = this.getContentResolver().query(Uri.parse("content://call_log/calls"),
                null, null, null, android.provider.CallLog.Calls.DATE + " DESC");
            while (cursor.moveToNext()) {
                String callLogID = cursor.getString(cursor.getColumnIndex(android.provider.CallLog.Calls._ID));
                String callNumber = cursor.getString(cursor.getColumnIndex(android.provider.CallLog.Calls.NUMBER));
                String callDate = cursor.getString(cursor.getColumnIndex(android.provider.CallLog.Calls.DATE));
                // 1-incoming; 2-outgoing; 3-missed
                String callType = cursor.getString(cursor.getColumnIndex(android.provider.CallLog.Calls.TYPE));
                // 0-call has been acknowledged; 1-call has not been acknowledge
                String isCallNew = cursor.getString(cursor.getColumnIndex(android.provider.CallLog.Calls.NEW));
                Log.d("", "callLogID: " + callLogID);
                Log.d("", "callNumber: " + callNumber);
                Log.d("", "callDate: " + callDate);
                Log.d("", "callType: " + callType);
                Log.d("", "isCallNew: " + isCallNew);
                // kiểm tra gọi nhớ không được biết đến
                if (Integer.parseInt(callType) == android.provider.CallLog.Calls.MISSED_TYPE &&
                    Integer.parseInt(isCallNew) > 0) {
                    Log.d("", "Missed Call Found: " + callNumber);
                }
            }
        } catch (Exception ex) {
            Log.d("", "ERROR: " + ex.toString());
        } finally {
            cursor.close();
        }
    }
}

```

Bluetooth

1. Kiểm tra hỗ trợ Bluetooth trên thiết bị

Để kiểm tra xem thiết bị có hỗ trợ Bluetooth hay không, ta tạo đối tượng BluetoothAdapter bằng cách gọi phương thức BluetoothAdapter.getDefaultAdapter(). Nếu thiết bị không hỗ trợ Bluetooth, đối tượng trả về là null.

```

BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();

private boolean bluetoothAvailable() {
    return adapter != null;
}

```

```
}

```

2. Bật chức năng Bluetooth

Nếu thiết bị của bạn hỗ trợ Bluetooth, bước tiếp theo là kiểm tra xem nó đã bật hay chưa bằng cách dùng phương thức `isEnabled()` của đối tượng `BluetoothAdapter`. Nếu chưa bật, ta gửi intent để bật nó lên. Đối tượng intent sẽ yêu cầu Android hiển thị activity nhắc người dùng bật chức năng Bluetooth. Để kiểm tra xem người dùng đã bật Bluetooth bằng activity đó chưa, gửi intent bằng phương thức `startActivityForResult()`, rồi viết lại phương thức `onActivityResult()` để nhận kết quả bật Bluetooth từ người dùng.

```
private void enableBluetooth() {
    if (bluetoothAvailable() && !adapter.isEnabled()) {
        Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(intent, REQUEST_ENABLE_BT);
    }
}

public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_ENABLE_BT) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this, "Bluetooth turned on!", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Cần cấp quyền `android.permission.BLUETOOTH`. Nếu bạn muốn ứng dụng của bạn tìm và thao tác thiết lập Bluetooth thay cho người dùng (không khuyến khích), cấp quyền `android.permission.BLUETOOTH_ADMIN`.

Để kiểm soát trạng thái của Bluetooth trên thiết bị, ta có thể dùng lớp `BroadcastReceiver` và đăng ký filter chặn intent có action `android.bluetooth.adapter.action.STATE_CHANGED`.

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override public void onReceive(Context context, Intent intent) {
        int prevState = intent.getExtras().getInt(BluetoothAdapter.EXTRA_PREVIOUS_STATE);
        int state = intent.getExtras().getInt(BluetoothAdapter.EXTRA_STATE);
        String status = "";
        switch (state) {
            case BluetoothAdapter.STATE_OFF:
                status = "Bluetooth off"; break;
            case BluetoothAdapter.STATE_TURNING_OFF:
                status = "Bluetooth turning off"; break;
            case BluetoothAdapter.STATE_ON:
                status = "Bluetooth on"; break;
            case BluetoothAdapter.STATE_TURNING_ON:
                status = "Bluetooth turning on";
        }
        Log.d(TAG, status);
    }
};

IntentFilter filter = new IntentFilter("android.bluetooth.adapter.action.STATE_CHANGED");
registerReceiver(receiver, filter);
```

3. Các thiết bị ghép nối

Ứng dụng phải đăng ký `BroadcastReceiver` chặn các intent `BluetoothDevice.ACTION_FOUND` cho từng thiết bị `BluetoothDevice` được ghép nối đến. Intent này có các extras `EXTRA_DEVICE` và `EXTRA_CLASS`, chứa thông tin về `BluetoothDevice` và `BluetoothClass` tương ứng.

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override public void onReceive(Context context, Intent intent) {
        if (BluetoothDevice.ACTION_FOUND.equals(intent.getAction())) {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            arrayAdapter.add(device.getName() + "\n" + device.getAddress());
        }
    }
};

IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(receiver, filter);
```

Để lấy danh sách các thiết bị đã ghép nối trước đó, gọi phương thức `getBondedDevices()`. Phương thức này trả về một tập (Set) các `BluetoothDevice` đã ghép nối trước đó.

```
private void listPairedDevices(View view) {
    Set<BluetoothDevice> pairedDevices = BA.getBondedDevices();
    ArrayList<String> list = new ArrayList<String>();
    list.addAll(pairedDevices);
    ArrayAdapter adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, list);
    listView.setAdapter(adapter);
}
```

}

4. Truyền dữ liệu qua Bluetooth

Giống như kết nối Socket thông qua kênh TCP, kết nối giữa hai thiết bị dùng Bluetooth sử dụng BluetoothSocket, hình thành kênh RFCOMM (Radio Frequency Communication). Tuy nhiên kết nối không được thực hiện trong UI thread.

a) Bluetooth Server

Trước tiên, lấy BluetoothServerSocket từ phương thức `listenUsingRfcommWithServiceRecord(String, UUID)`, String là tên bất kỳ và UUID là chuỗi ID 128-bit.

BluetoothServerSocket lắng nghe kết nối vào bằng phương thức `accept()`. Khi chấp nhận kết nối vào, phương thức này trả về BluetoothSocket được kết nối với BluetoothSocket phía client. Kết nối được chấp nhận khi và chỉ khi thiết bị từ xa gửi yêu cầu với UUID giống với UUID của server socket. Stream I/O lấy từ BluetoothSocket cho phép truyền thông giữa hai bên.

```
public class MainActivity extends Activity {
    private static final int DISCOVERABLE_REQUEST_CODE = 0x1;
    private boolean CONTINUE_READ_WRITE = true;
    private BluetoothSocket socket;
    private InputStream in;
    private OutputStreamWriter out;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // luôn bảo đảm Bluetooth server có thể quét thấy trong lúc đang lắng nghe
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivityForResult(discoverableIntent, DISCOVERABLE_REQUEST_CODE);
    }

    @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        new Thread(reader).start();
    }

    @Override protected void onDestroy() {
        super.onDestroy();
        if (socket != null) {
            try {
                in.close(); out.close(); socket.close();
            } catch (Exception e) { e.printStackTrace(); }
            CONTINUE_READ_WRITE = false;
        }
    }

    private Runnable reader = new Runnable() {
        @Override public void run() {
            BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
            UUID uuid = UUID.fromString("4e5d48e0-75df-11e3-981f-0800200c9a66");
            try {
                BluetoothServerSocket serverSocket = adapter.listenUsingRfcommWithServiceRecord("BTServer", uuid);
                socket = serverSocket.accept();
                in = socket.getInputStream();
                out = new OutputStreamWriter(socket.getOutputStream());
                new Thread(writer).start();
                int bufferSize = 1024;
                int bytesRead = -1;
                byte[] buffer = new byte[bufferSize];
                while (CONTINUE_READ_WRITE) {
                    final StringBuilder sb = new StringBuilder();
                    bytesRead = in.read(buffer);
                    if (bytesRead != -1) {
                        String result = "";
                        while ((bytesRead == bufferSize) && (buffer[bufferSize-1] != 0)) {
                            result += new String(buffer, 0, bytesRead - 1);
                            bytesRead = in.read(buffer);
                        }
                        result += new String(buffer, 0, bytesRead - 1);
                        sb.append(result);
                    }
                }
                runOnUiThread(new Runnable() { // hiển thị thông điệp UI thread
```

```

        @Override public void run() {
            Toast.makeText(MainActivity.this, sb.toString(), Toast.LENGTH_LONG).show();
        }
    }
} catch (IOException e) { e.printStackTrace(); }
}
};

private Runnable writer = new Runnable() {
    @Override public void run() {
        int index = 0;
        while (CONTINUE_READ_WRITE) {
            try {
                out.write("Message From Server" + (index++) + "\n");
                out.flush();
                Thread.sleep(2000);
            } catch (Exception e) { e.printStackTrace(); }
        }
    }
};
}
}

```

b) Bluetooth Client

Xác định thiết bị ghép nối từ xa (BluetoothDevice), dùng phương thức createRfcommSocketToServiceRecord(UUID) khởi tạo một BluetoothSocket sẽ kết nối đến BluetoothDevice đó. UUID truyền cho phương thức này phải giống với UUID của BluetoothServerSocket phía server. Hệ thống sẽ thực hiện giao thức SDP (Service Discovery Protocol) để tìm thiết bị có UUID phù hợp.

```

public class MainActivity extends Activity {
    private boolean CONTINUE_READ_WRITE = true;
    private BluetoothSocket socket;
    private InputStream in;
    private OutputStreamWriter out;
    private BluetoothDevice remoteDevice;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        registerReceiver(discoveryResult, new IntentFilter(BluetoothDevice.ACTION_FOUND));
        BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
        if (adapter != null && adapter.isDiscovering()) {
            adapter.cancelDiscovery();
        }
        adapter.startDiscovery();
    }

    @Override protected void onDestroy() {
        super.onDestroy();
        try {
            unregisterReceiver(discoveryResult);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (socket != null) {
            try {
                in.close(); out.close(); socket.close();
                CONTINUE_READ_WRITE = false;
            } catch (Exception e) { e.printStackTrace(); }
        }
    }

    private BroadcastReceiver discoveryResult = new BroadcastReceiver() {
        @Override public void onReceive(Context context, Intent intent) {
            unregisterReceiver(this);
            remoteDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            new Thread(reader).start();
        }
    };

    private Runnable reader = new Runnable() {
        @Override public void run() {

```

```

try {
    UUID uuid = UUID.fromString("4e5d48e0-75df-11e3-981f-0800200c9a66");
    socket = remoteDevice.createRfcommSocketToServiceRecord(uuid);
    socket.connect();
    out = new OutputStreamWriter(socket.getOutputStream());
    in = socket.getInputStream();
    new Thread(writer).start();
    int bufferSize = 1024;
    int bytesRead = -1;
    byte[] buffer = new byte[bufferSize];
    while (CONTINUE_READ_WRITE) {
        final StringBuilder sb = new StringBuilder();
        bytesRead = in.read(buffer);
        if (bytesRead != -1) {
            String result = "";
            while ((bytesRead == bufferSize) && (buffer[bufferSize-1] != 0)) {
                result += new String(buffer, 0, bytesRead - 1);
                bytesRead = in.read(buffer);
            }
            result += new String(buffer, 0, bytesRead - 1);
            sb.append(result);
        }
        runOnUiThread(new Runnable() { // hiển thị thông điệp trên UI thread
            @Override public void run() {
                Toast.makeText(MainActivity.this, sb.toString(), Toast.LENGTH_LONG).show();
            }
        });
    }
} catch (IOException e) { e.printStackTrace(); }
};

private Runnable writer = new Runnable() {
    @Override public void run() {
        int index = 0;
        while (CONTINUE_READ_WRITE) {
            try {
                out.write("Message From Client" + (index++) + "\n");
                out.flush();
                Thread.sleep(2000);
            } catch (Exception e) { e.printStackTrace(); }
        }
    }
};

```

Email

Giống như SMS, bạn cũng có thể gửi mail bằng hai cách: lập trình bên trong ứng dụng hoặc gọi ứng dụng email client của Android thông qua Intent. Để khởi động ứng dụng email client, bạn dùng intent với action ACTION_SEND, data là "mailto:" và type là "message/rfc822". Bạn có thể thêm các extras để gửi email vào intent:

Dữ liệu Extras	Mô tả
EXTRA_BCC	Mảng chuỗi lưu các địa chỉ email sẽ dùng trong BCC (blind carbon copied).
EXTRA_CC	Mảng chuỗi lưu các địa chỉ email sẽ dùng trong CC (carbon copied).
EXTRA_EMAIL	Mảng chuỗi lưu các địa chỉ email đích.
EXTRA_HTML_TEXT	Chuỗi hằng liên kết với intent ACTION_SEND để cung cấp văn bản định dạng HTML đến EXTRA_TEXT.
EXTRA_SUBJECT	Một chuỗi hằng chứa trong dòng subject của thông điệp.
EXTRA_TEXT	Chuỗi hằng liên kết với intent, dùng với intent ACTION_SEND hỗ trợ gửi dữ liệu chuỗi.
EXTRA_TITLE	Tiêu đề của hộp thoại cung cấp cho người dùng khi dùng với intent ACTION_CHOOSER.

```

public void onClick(View v) {
    String[] to = { "someguy@example.com", "anotherguy@example.com" };
    String[] cc = { "busybody@example.com" };
    sendEmail(to, cc, "Hello", "Hello my friends!");
}

private void sendEmail(String[] emailAddresses, String[] carbonCopies,
    String subject, String message) {
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    String[] to = emailAddresses;

```

```
String[] cc = carbonCopies;
emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
}
```


Networking

Android cho phép ứng dụng của bạn kết nối với Internet hay bất kỳ mạng cục bộ khác và cho phép bạn thực hiện các tác vụ mạng. Một thiết bị có thể có nhiều vài kiểu kết nối mạng: kết nối Wi-Fi hoặc kết nối mạng di động.

Kiểm tra kết nối mạng

Trước khi thực hiện các tác vụ mạng, bạn phải kiểm tra xem thiết bị có kết nối với Internet hoặc một mạng cục bộ nào không. Android cung cấp lớp `ConnectivityManager` để thực hiện việc này, trước hết bạn cần tạo một thực thể của lớp `ConnectivityManager`:

```
ConnectivityManager manager = (ConnectivityManager) this.context
    .getSystemService(Context.CONNECTIVITY_SERVICE);
```

Khi có thực thể của lớp `ConnectivityManager`, bạn có thể dùng phương thức `getAllNetworkInfo()` của nó để lấy thông tin của tất cả các mạng. Phương thức này trả về một mảng các đối tượng `NetworkInfo`:

```
NetworkInfo[] info = manager.getAllNetworkInfo();
```

Cuối cùng bạn duyệt mảng này để kiểm tra trạng thái kết nối của mạng:

```
for (int i = 0; i < info.length; ++i) {
    if (info[i].getState() == NetworkInfo.State.CONNECTED) {
        Toast.makeText(context, "Internet is connected!", Toast.LENGTH_SHORT).show();
    }
}
```

Các trạng thái của mạng bao gồm: `CONNECTED`, `CONNECTING`, `DISCONNECTED`, `DISCONNECTING`, `SUSPENDED`, `UNKNOWN`.

Trong tập tin manifest, cấp thêm các quyền sau:

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

Sau khi kiểm tra thấy thiết bị đang kết nối với Internet, bạn có thể thực hiện tác vụ mạng bất kỳ.

Các phương thức kết nối

Thiết bị được xem như phía client, Android hỗ trợ nhiều phương thức kết nối đến server.

1. Socket

Lập trình mạng bằng Socket trên Android tương tự với lập trình mạng bằng Java trên desktop. Các bước điển hình như sau:

- Tạo đối tượng Socket, kết nối đến một port chỉ định trên một host chỉ định.
- Lấy các stream nhập/xuất từ đối tượng Socket, lồng các stream thích hợp để sử dụng. Thường dùng `BufferedReader` với stream nhập và `PrintWriter` với stream xuất.
- Làm việc với các stream nhập/xuất để nhận/gửi dữ liệu từ/đến server.
- Đóng Socket.

```
public class NistTimeActivity extends Activity {
    private String mHost = "time-b.nist.gov";
    private int mPort = 13;
    private Button mButon;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mButon = (Button) findViewById(R.id.show_time);
        mButon.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                try {
                    Socket socket = new Socket(mHost, mPort);
                    BufferedReader in = SocketUtils.getReader(socket);
                    in.readLine(); // bỏ qua dòng trống đầu tiên
                    String timeResult = in.readLine();
                    Toast.makeText(getBaseContext(), timeResult, Toast.LENGTH_SHORT).show();
                    socket.close();
                } catch (UnknownHostException uhe) {
                    Toast.makeText(getBaseContext(), "Unknownhost: " + mHost, Toast.LENGTH_SHORT).show();
                } catch (IOException ioe) {
                    Toast.makeText(getBaseContext(), "IOException: " + ioe, Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

2. Apache HttpClient

Thư viện mã nguồn mở Apache HTTP là một phần của Android SDK từ API Level 3. Thư viện này tổ chức thành hai phần: `HttpCore` là tập các lớp cấp thấp xử lý kết nối HTTP, `HttpClient` là tập các lớp cấp cao, xây dựng trên `HttpCore`, dùng cài

đặt cho các ứng dụng sử dụng HTTP, hỗ trợ tốt SSL. Tuy nhiên, Apache HttpClient không chứa lớp MultipartEntity, nên bất tiện khi gửi thông điệp phức tạp.

Ví dụ sau dùng Apache HttpClient, tạo AsyncTask thực hiện tác vụ chạy nền, lấy trang web từ web server:

```
public class DownloadWebTask extends AsyncTask<String, Void, String> {
    private TextView mTextView;
    private Context context;

    public DownloadWebTask(Context context, TextView dataText) {
        this.context = context;
        this.mTextView = dataText;
    }

    @Override protected void onPreExecute() {
        // kiểm tra kết nối mạng
    }

    @Override protected String doInBackground(String... args) {
        try {
            HttpGet request = new HttpGet(args[0]);
            HttpResponse response = new DefaultHttpClient().execute(request);
            int statusCode = response.getStatusLine().getStatusCode();
            if (statusCode != HttpStatus.SC_OK) { // reply code 200
                return "Error: Failed getting web page";
            }
            return EntityUtils.toString(response.getEntity());
        } catch (Exception e) {
            return new String("Error: " + e.getMessage());
        }
    }

    @Override protected void onPostExecute(String result) {
        mTextView.setText(result);
    }
}
```

gọi AsyncTask từ onCreate() của activity:

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    contentText = (TextView) findViewById(R.id.content_id);
    new DownloadWebTask(this, contentText).execute("http://www.trainingwithexperts.com");
}
```

3. HttpURLConnection

Từ API Level 9, Android đề nghị sử dụng lớp HttpURLConnection và lớp URL để truy cập giao thức HTTP với hiệu suất cao và ổn định.

Đầu tiên, bạn cần tạo thực thể URL bằng cách cung cấp link đến website:

```
String urlString = "http://www.google.com";
URL url = new URL(urlString);
```

Gọi phương thức openConnection() của lớp URL và ép kiểu kết quả nhận được thành đối tượng HttpURLConnection. Tiếp theo, bạn gọi phương thức connect() của đối tượng này:

```
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.connect();
```

Cuối cùng, bạn lấy stream nhập của đối tượng HttpURLConnection bằng phương thức getInputStream(). Dùng stream này, bạn có thể đọc dữ liệu từ website:

```
private HttpURLConnection connection = null;

private InputStream OpenHttpConnection(String urlString) throws IOException {
    InputStream in = null;
    int response = -1;

    URL url = new URL(urlString);
    connection = (HttpURLConnection) url.openConnection();
    // Dùng với GET, với POST thêm: conn.setDoOutput(true);
    connection.setDoInput(true);
    connection.setAllowUserInteraction(true);
    connection.setInstanceFollowRedirects(true);
    connection.connect();
    response = connection.getResponseCode();
    if (response == HttpURLConnection.HTTP_OK)
```

```

    in = connection.getInputStream();
    return in;
}

```

Lớp `URLConnection` còn cung cấp nhiều phương thức khác:

`disconnect()`

Giải phóng kết nối, kết nối có thể dùng lại hoặc đóng.

`getRequestMethod()`

Trả về method của yêu cầu sẽ được sử dụng (GET, POST, ...), lệnh để server thực hiện.

`getResponseCode()`

Trả về mã đáp ứng từ server.

`setRequestMethod(String method)`

Thiết lập lệnh (GET, POST, ...) sẽ được gửi đến server.

`usingProxy()`

Cho biết kết nối có dùng máy chủ proxy hay không.

Ngoài ra, đối tượng `URLConnection` cung cấp phương thức tiện dụng `openStream()`, cho phép lấy được ngay stream nhập:

- Với dữ liệu chuỗi, lồng stream với `Reader/Writer` để dễ đọc ghi.
- Với dữ liệu hình ảnh, chuyển stream nhị phân thành `Bitmap`.

```

private Bitmap loadImage(String url) {
    Bitmap bitmap = null;
    InputStream in = null;
    try {
        in = new URL(url).openStream();
        bitmap = BitmapFactory.decodeStream(in);
        in.close();
    } catch (IOException ex) {
        Log.d("Networking", ex.getLocalizedMessage());
    }
    return bitmap;
}

```

Tác vụ tải từ mạng thường chiếm nhiều thời gian, nên đưa vào một `AsyncTask`, được tạo như lớp thành viên của activity. Ví dụ sau đọc thông tin từ mạng vào đối tượng POJO `Movie` tạo sẵn, đồng thời tải ảnh poster của `Movie` đó về.

```

private class BackgroundTask extends AsyncTask<String, Void, Movie> {
    private final ProgressDialog dialog;
    private HttpURLConnection connection = null;
    private Bitmap bitmap;
    private Context context;

    public BackgroundTask(Context context) {
        this.context = context;
        dialog = new ProgressDialog(context);
    }

    @Override protected void onPreExecute() {
        super.onPreExecute();
        ConnectivityManager manager = (ConnectivityManager)
            context.getSystemService(Context.CONNECTIVITY_SERVICE);

        if (manager == null) {
            Toast.makeText(context, "Not connected to internet", Toast.LENGTH_SHORT).show();
            this.cancel(true);
        } else {
            this.dialog.setMessage("Wait...");
            this.dialog.show();
        }
    }

    private Movie readAndParseJSON(String in) {
        Movie movie = null;
        // parse JSON thành đối tượng Movie (xem phần sau)
        return movie;
    }

    private Bitmap loadImage(String url) {
        // tải ảnh từ kết nối HttpURLConnection
    }

    @Override protected Movie doInBackground(String... args) {
        Movie movie = null;
        try {
            InputStream stream = new URL(args[0]).openStream();
            if (stream != null) {

```

```

        BufferedReader in = new BufferedReader(new InputStreamReader(stream));
        movie = readAndParseJSON(in.readLine());
        in.close();
        stream.close();
        if (movie != null)
            bitmap = loadImage(movie.getImageURL());
    }
} catch (IOException e) {
    e.printStackTrace();
}
return movie;
}

@Override protected void onPostExecute(Movie movie) {
    if (movie != null) {
        ImageView mImage = (ImageView) findViewById(R.id.image_id);
        mImage.setImageBitmap(bitmap);
        TextView mTextView = (TextView) findViewById(R.id.text_id);
        mTextView.setText(Html.fromHtml(movie.toString()));
    } else {
        Toast.makeText(context, "Sorry! Movie not found!", Toast.LENGTH_SHORT).show();
    }
    if (connection != null) connection.disconnect();
    if (this.dialog.isShowing()) this.dialog.dismiss();
}
}

```

Cuối cùng, gọi AsyncTask từ onCreate() của activity:

```

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    String url = "http://www.omdbapi.com/?s=%s";
    String title = "Spirited Away";
    try {
        String urlString = String.format(url, "t", URLEncoder.encode(title, "UTF-8"));
        new HandleJSON(this).execute(urlString);
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
}

```

REST API

Ứng dụng của bạn thường cần truy cập một RESTful API thông qua HTTP để tương tác với web services cung cấp bởi máy từ xa. REST viết tắt của Representational State Transfer, là một kiểu kiến trúc phổ biến cho web services hiện nay. RESTful API được xây dựng sao cho có thể dùng HTTP chuẩn tạo yêu cầu (request) gửi đến máy chứa tài nguyên và nhận đáp ứng (response) trả về trong các định dạng tài liệu có cấu trúc phổ biến như XML, JSON, CSV.

Chúng tôi giới thiệu OMDb (Open Movie Database) API được phát triển bởi Brian Fritz, được sử dụng rộng rãi để lấy thông tin từ IMDb. IMDb (Internet Movie Database) là cơ sở dữ liệu thông tin phim lớn nhất trên Internet, có hơn 2 triệu tựa phim.

Tham số dùng cho OMDb API đơn giản:

Tham số	Trị	Mô tả
s	chuỗi (tùy chọn)	Tựa phim để tìm kiếm.
i	chuỗi (tùy chọn)	Định danh IMDb hợp lệ.
t	chuỗi (tùy chọn)	Tựa phim sẽ trả về thông tin chi tiết.
y	năm (tùy chọn)	Năm sản xuất phim.
r	json, xml	Kiểu dữ liệu trả về (mặc định là JSON).
plot	short, full	Tóm tắt nội dung (mặc định là short).
callback	tên (tùy chọn)	Tên của JSON callback.
tomatoes	true (tùy chọn)	Thêm thông tin đánh giá của trang Rotten Tomatoes.

Ví dụ:

```
http://www.omdbapi.com/?t=Spirited+Away&r=xml
```

Bạn cũng có thể chọn lấy API khác, tuy nhiên thường phải đăng ký API_KEY. Ví dụ:

```
https://api.themoviedb.org/3/search/movie?api_key=624645327f33f7866355b7b728f9cd98&query=Spirited+Away
```

JSON nhận được:

```

{
  "page": 1,
  "results": [
    {
      "adult": false,
      "backdrop_path": "/mnpRKVSXBx6jb56nabvmGKA0Wig.jpg",
      "id": 129,

```

```

        "original_title": "千と千尋の神隠し",
        "release_date": "2001-07-20",
        "poster_path": "/dL11DBPcRhWwNjCfXl9A07MrqTI.jpg",
        "popularity": 8.97512289897908,
        "title": "Spirited Away",
        "vote_average": 7.9,
        "vote_count": 1001
    }
],
"total_pages": 1,
"total_results": 1
}

```

đường dẫn đầy đủ để lấy poster tại:

<http://image.tmbd.org/t/p/w500/dL11DBPcRhWwNjCfXl9A07MrqTI.jpg>

Một số API:

- Google Search:

https://www.googleapis.com/customsearch/v1?key=AIzaSyBbW-W1SHCK4eW0kK74VGMLJj_b-byNzkI&cx=008212991319514020231:1mkouq8yagw&q=Android

key là KEY_API, cx là Custom Search Engine ID.

- Thông tin thời tiết:

<http://api.openweathermap.org/data/2.5/weather?q=Hanoi&mode=xml>

hoặc thông tin thời tiết cho Asia/Ho Chi Minh:

<https://api.forecast.io/forecast/e4b286e91fb6fa9a7fe5efef584adc06/10.8142,106.6438>

Bạn có thể dùng các tiện ích REST client để kiểm tra trước:

- RESTClient, Add-on của FireFox, tại: <http://restclient.net/>

- Advanced REST client, extension của Chrome, tại: restforchrome.blogspot.com

Truy cập REST API, bạn sẽ nhận được dữ liệu trả về có cấu trúc JSON hoặc XML, cần phải phân tích (parse) để sử dụng.

SOAP WebServices

Dịch vụ SOAP (Simple Object Access Protocol) thường định nghĩa một contract, viết bằng WSDL (Web Services Description Language). Contract liên quan đến mọi cấu trúc dữ liệu, phương thức gọi dịch vụ của SOAP và được công bố cho người sử dụng web services (thường gọi là consumers).

SOAP sử dụng XML để đóng gói dữ liệu cho các yêu cầu/đáp ứng.

Bạn thử tìm hiểu sử dụng web services bằng cách dùng WebService Studio, tải về tại: <http://webservicesstudio.codeplex.com/>,

sử dụng web services cung cấp thông tin chứng khoán tại: <http://www.webservice.net/stockquote.aspx>

Chạy WebService Studio, nhập vào WSDL End Point: <http://www.webservice.net/stockquote.aspx?WSDL> rồi nhấn Get.

Trong tab Invoke, phần Input, chú ý phương thức được gọi là GetQuote có tham số là symbol. Chọn symbol rồi nhập chuỗi các mã chứng khoán MSFT ORCL AMZN AAPL GOOG vào trường Value, trong phần Value. Nhấn Invoke. Bạn sẽ nhận được kết quả trả về dưới định dạng XML trong phần Output. Để thấy rõ dữ liệu yêu cầu/đáp ứng, đóng gói trong định dạng XML, vào tab Request/Response.

Bạn cũng có thể dùng add-on SOAP Client của FireFox: <https://addons.mozilla.org/en-US/firefox/addon/soa-client/?src=api>

1. Phân tích dữ liệu XML bằng SAX

Dữ liệu chứa trong mỗi element <Stock> nhận được sẽ được phân tích rồi đưa vào một POJO StockQuote:

```

public class StockQuote {
    private String symbol;
    private String name;
    private Double quote;
    // constructors và getters/setters
}

```

Trước đây ta đã thảo luận việc phân tích dữ liệu XML bằng XMLPullParser. Trong phần này, ta dùng SAX để phân tích dữ liệu XML nhận được, trong lớp StockQuoteHandler:

```

public class StockQuoteHandler extends DefaultHandler {
    private ArrayList<StockQuote> quotes = new ArrayList<StockQuote>();
    private StockQuote currentQuote = new StockQuote();
    private String currentNodeText;

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException {
        // Tạo một StockQuote mới với mỗi node <Stock> trong tài liệu XML
        if (localName.equalsIgnoreCase("Stock"))
            currentQuote = new StockQuote();
    }

    @Override public void characters(char[] ch, int start, int length) throws SAXException {
        // Lấy nội dung TEXT của node đang xét
    }
}

```

```

        currentNodeText = new String(ch, start, length);
    }

    @Override public void endElement(String uri, String localName, String qName) throws SAXException {
        if (localName.equalsIgnoreCase("Symbol")) {
            currentQuote.setSymbol(currentNodeText);
        } else if (localName.equalsIgnoreCase("Name")) {
            currentQuote.setName(currentNodeText);
        } else if (localName.equalsIgnoreCase("Last")) {
            currentQuote.setQuote(Double.parseDouble(currentNodeText));
        } else if (localName.equalsIgnoreCase("Stock")) {
            // Gặp tag đóng </Stock>, đưa StockQuote vừa tạo vào mảng quotes kết quả
            quotes.add(currentQuote);
        }
    }

    public ArrayList<StockQuote> getQuotes() {
        return quotes;
    }
}

```

2. Truy cập SOAP WebServices

Android không cung cấp thư viện hỗ trợ truy cập SOAP. Tuy nhiên, bạn có thể dùng thư viện bổ sung kSOAP2, tải về tại:

<https://code.google.com/p/ksoap2-android/>

Đưa gói ksoap2-android-assembly-2.5.4-jar-with-dependencies.jar tải về vào thư mục libs của project. Thư viện kSOAP2 cung cấp:

- Tạo đối tượng chứa yêu cầu (request) là SoapObject.
- Thiết lập các thuộc tính cho SoapObject bằng phương thức addProperty(). Thuộc tính là đối tượng PropertyInfo, cung cấp tham số cho phương thức gọi web services.
- Đóng gói SoapObject trong SoapSerializationEnvelope để gửi đi cùng với phương thức cần gọi.
- Kết quả trả về được đóng gói trong đối tượng SoapPrimitive.
- Phân tích (parse) kết quả trả về bằng SAX, với phương thức xử lý là StockQuoteHandler đã viết ở trên.

```

public class StockQuoteFetcher {
    private final String NAMESPACE = "http://www.webserviceX.NET/";
    private final String METHOD_NAME = "GetQuote";
    private final String SOAP_ACTION = "http://www.webserviceX.NET/GetQuote";
    private final String URL = "http://www.webservices.net/stockquote.asmx";

    private final SoapSerializationEnvelope envelope;

    public StockQuoteFetcher(String quotes) {
        SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
        PropertyInfo quotesProperty = new PropertyInfo();
        quotesProperty.setName("symbol");
        quotesProperty.setValue(quotes);
        quotesProperty.setType(String.class);
        request.addProperty(quotesProperty);

        envelope = new SoapSerializationEnvelope(SoapEnvelope.VERSION1);
        envelope.dotNet = true;
        envelope.setOutputSoapObject(request);
    }

    public List<StockQuote> fetch() {
        HttpTransportSE httpRequest = new HttpTransportSE(URL);
        StockQuoteHandler quoteParser = new StockQuoteHandler();
        try {
            httpRequest.call(SOAP_ACTION, envelope);
            SoapPrimitive response = (SoapPrimitive) envelope.getResponse();
            Xml.parse(response.toString(), quoteParser);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return quoteParser.getQuotes();
    }
}

```

Chuẩn bị một adapter tùy biến, constructor nhận mảng các đối tượng StockQuote, chuyển dữ liệu của mỗi đối tượng vào một mục của ListView.

```

public class StockQuoteAdapter extends ArrayAdapter<StockQuote> {
    private final Activity activity;

```



```

private final List<StockQuote> stocks;

public StockQuoteAdapter(Activity activity, List<StockQuote> objects) {
    super(activity, R.layout.stock_quote_list_item, objects);
    this.activity = activity;
    this.stocks = objects;
}

@Override public View getView(int position, View convertView, ViewGroup parent) {
    View rowView = convertView;
    StockQuoteView sqView = null;
    if (rowView == null) {
        LayoutInflater inflater = activity.getLayoutInflater();
        rowView = inflater.inflate(R.layout.list_item, parent, false);
        sqView = new StockQuoteView();
        sqView.ticker = (TextView) rowView.findViewById(R.id.ticker_symbol);
        sqView.quote = (TextView) rowView.findViewById(R.id.ticker_price);
        rowView.setTag(sqView);
    } else {
        sqView = (StockQuoteView) rowView.getTag();
    }
    // Chuyển dữ liệu từ POJO vào view
    StockQuote currentStock = stocks.get(position);
    sqView.ticker.setText(currentStock.getSymbol() + " (" + currentStock.getName() + ")");
    sqView.quote.setText(currentStock.getQuote().toString());
    return rowView;
}

protected static class StockQuoteView {
    protected TextView ticker;
    protected TextView quote;
}
}

```

Activity chính:

```

public class MainActivity extends ListActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);

        StockQuoteFetcher sqf = new StockQuoteFetcher("MSFT ORCL AMZN AAPL GOOG");
        List<StockQuote> quoteResult = sqf.fetch();
        setListAdapter(new StockQuoteAdapter(this, quoteResult));
    }
}

```

cần thêm quyền trong tập tin manifest:

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

Location

Location Based Services

Location Based Services cung cấp tập API trong gói android.location dùng định vị cho phép bạn dễ dàng xây dựng các ứng dụng dựa trên nhận biết vị trí (location-aware), mà không cần quan tâm chi tiết đến công nghệ định vị bên dưới. Điều này giúp ứng dụng của bạn có thể theo dõi tự động vị trí, geofencing².

Ví dụ: xây dựng ứng dụng tìm bệnh viện hoặc nhà hàng gần nhất, định vị sự kiện và con người trên mạng xã hội, cung cấp thông tin di chuyển giữa hai địa điểm.

1. Geocoder

Nếu làm việc với vị trí, bạn thường phải chuyển một địa chỉ thành cặp latitude/longitude (vĩ độ/kinh độ), khái niệm này gọi là forward geocoding. Ngược lại, chuyển đổi từ một cặp latitude/longitude thành một danh sách các địa chỉ, gọi là reverse geocoding. Android cung cấp lớp android.location.Geocoder giúp bạn thực hiện các chuyển đổi này.

Các phương thức của lớp Geocoder:

```
List<Address> getFromLocation(double latitude, double longitude, int maxResults)
List<Address> getFromLocationName(String locationName, int maxResults,
    double lowerLeftLatitude, double lowerLeftLongitude,
    double upperRightLatitude, double upperRightLongitude)
List<Address> getFromLocationName(String locationName, int maxResults)
```

```
// lat/long của Đại học FPT
int latitude = (int) (10.8529544 * 1000000);
int longitude = (int) (106.6292685 * 1000000);
GeoPoint point = new GeoPoint(latitude, longitude);
// lấy lat/long từ bản đồ của một MapActivity, thường thực hiện trong thread nền
List<Address> addressList = new Geocoder(this).getFromLocationName("fpt university saigon", 3);
if (addressList != null && !addressList.isEmpty()) {
    latitude = (int) (addressList.get(0).getLatitude() * 1000000);
    longitude = (int) (addressList.get(0).getLongitude() * 1000000);
}
// reverse geocoding
addressList = new Geocoder(this).getFromLocation(latitude, longitude, 1);
if (addressList != null && !addressList.isEmpty()) {
    String Address1 = addressList.get(0).getAddressLine(0);
    String Address2 = addressList.get(0).getAddressLine(1);
    String State = addressList.get(0).getAdminArea();
    String Zipcode = addressList.get(0).getPostalCode();
    String Country = addressList.get(0).getCountryName();
}
```

Tham số cuối của phương thức getFromLocationName() và getFromLocation() là số kết quả đề nghị trả về nếu có thể.

2. LocationManager và LocationProvider

Giống như sensor, đầu tiên cần lấy đối tượng LocationManager từ getSystemService(LOCATION_SERVICE). Service này cho phép truy cập bên cung cấp thông tin vị trí LocationProvider để đăng ký cập nhật vị trí.

Thiết bị Android có sẵn vài LocationProvider và bạn có thể chọn một trong chúng:

Hằng	Mô tả
NETWORK_PROVIDER	Dùng mạng điện thoại di động hoặc WiFi để định vị. Chính xác hơn GPS khi sử dụng trong nhà.
GPS_PROVIDER	Sử dụng GPS (Global Positioning System) trong thiết bị Android để định vị thông qua vệ tinh. Hoạt động kém và thiếu chính xác trong nhà và nơi bị che khuất, cản trở.
PASSIVE_PROVIDER	Sử dụng thông tin định vị được cập nhật bởi các thành phần khác.

Để LocationProvider hoạt động, cần cấp các quyền sau trong tập tin manifest:

- Cho phép ứng dụng truy cập vị trí gần đúng của người dùng, thông tin lấy từ mạng di động hoặc WiFi:

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

- Cho phép ứng dụng truy cập vị trí chính xác của người dùng, thông tin dựa trên sự kết hợp của hệ thống GPS, các nguồn vị trí mạng và WiFi:

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

- Cho phép ứng dụng truy cập Internet:

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

Do người dùng có thể bật/tắt GPS hoặc WiFi, LocationManager cung cấp phương thức isProviderEnabled() cho phép ứng dụng truy vấn trạng thái của LocationManager. Ngoài ra, khi người dùng thay đổi provider, hệ thống phát intent quảng bá với action android.location.PROVIDERS_CHANGED, ứng dụng có thể nhận thông điệp này để có hành động phù hợp:

```
private final BroadcastReceiver providerChangedListener = new BroadcastReceiver() {
    @Override public void onReceive(Context context, Intent intent) {
        LocationManager manager = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
        if (manager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
```

² Geofencing là phần mềm dùng hệ thống định vị toàn cầu (GPS - Global Positioning System) hoặc nhận dạng tần số vô tuyến (RFID - Radio Frequency Identification) để xác định vị trí địa lý.

```

        // dùng GPS provider
    } else {
        // dùng provider khác
    }
};
...
registerReceiver(providerChangeListener, new IntentFilter(LocationManager.PROVIDERS_CHANGED_ACTION));

```

3. Cập nhật vị trí

LocationProvider cho phép ứng dụng nhận cập nhật vị trí bằng hai cách: cập nhật vị trí liên tục và cảnh báo tiệm cận.

a) Cập nhật vị trí liên tục (continuous location updates)

Ứng dụng đăng ký một LocationListener để nhận cập nhật vị trí khi vị trí của người dùng thay đổi.

```

private final LocationListener listener = new LocationListener() {
    @Override public void onLocationChanged(Location location) {
        // cập nhật vị trí từ Location được truyền
    }
    @Override public void onStatusChanged(String provider, int status, Bundle extras) {
        // trạng thái provider có thay đổi
    }
    @Override public void onProviderEnabled(String provider) {
        // provider được kích hoạt
    }
    @Override public void onProviderDisabled(String provider) {
        // tắt provider
    }
};

manager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,    // tên của LocationProvider
    1000,                             // thời gian giữa hai lần cập nhật tối thiểu 1 sec
    1,                                // cập nhật khi khoảng cách thay đổi tối thiểu 1m
    listener);

```

Giống như khi dùng cảm biến, cập nhật vị trí liên tục tiêu thụ nhiều pin, khi không cần thiết nên tắt bằng:

```
manager.removeUpdates(listener);
```

Ứng dụng cũng có thể yêu cầu một cập nhật vị trí đơn, thiết bị sẽ cung cấp một cập nhật vị trí đơn rồi tắt.

```
manager.requestSingleUpdate(LocationManager.GPS_PROVIDER, listener, Looper.myLooper());
```

Bạn có thể dùng pending intent để đăng ký một cập nhật vị trí đơn.

b) Cảnh báo tiệm cận (proximity alert)

Một ứng dụng có thể đăng ký để được thông báo khi người dùng vào vùng tiệm cận của một vị trí địa lý cho trước. Vùng tiệm cận là một hình tròn có tâm là vị trí (vĩ độ, kinh độ) cho trước, bán kính (tính bằng m).

```

manager.addProximityAlert(
    37.3688,        // vĩ độ tâm vùng tiệm cận
    -122.0363,     // kinh độ tâm vùng tiệm cận
    100,            // bán kính vùng tiệm cận (m)
    -1,            // thời gian hết hạn tùy chọn
    PendingIntent.getActivity(this, 0, new Intent(this, LocationActivity.class), 0)
);

```

Cảnh báo tiệm cận được phân phối bằng pending intent (tham số cuối), pending intent này chứa một extra kiểu Boolean, KEY_PROXIMITY_ENTERING, cho biết người dùng đang vào hoặc ra vùng tiệm cận.

Khi không cần đến nữa, ứng dụng có thể tắt cảnh báo tiệm cận bằng:

```
manager.removeProximityAlert(pendingIntent);
```

c) Định vị nhanh

Thời gian nhận thông tin để hiệu chỉnh vị trí lần đầu có thể lớn, nhất là khi dùng GPS. Trong khi chờ cập nhật vị trí chính xác hơn, ứng dụng có thể đơn giản dùng phương thức getLastKnownLocation() của LocationManager để lấy vị trí cuối cùng được biết:

```
Location location = manager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
```

d) Ví dụ

```

public class LocationActivity extends Activity implements LocationListener {
    private TextView mLatitude;
    private TextView mLongitude;
    private LocationManager manager;
    private String provider;

```

```

@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mLatitude = (TextView) findViewById(R.id.latitude);
    mLongitude = (TextView) findViewById(R.id.longitude);
    manager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_COARSE);
    provider = manager.getBestProvider(criteria, false);
    Location location = manager.getLastKnownLocation(provider);
    if (location != null) {
        Toast.makeText(this, "Provider " + provider + " has been selected.", Toast.LENGTH_SHORT);
        mLatitude.setText(location.getLatitude() + "");
        mLongitude.setText(location.getLongitude() + "");
    } else {
        mLatitude.setText("Provider not available");
        mLongitude.setText("Provider not available");
    }
}

@Override protected void onResume() {
    super.onResume();
    manager.requestLocationUpdates(provider, 400, 1, this);
}

@Override protected void onPause() {
    super.onPause();
    manager.removeUpdates(this);
}

@Override public void onLocationChanged(Location location) {
    mLatitude.setText(location.getLatitude() + "");
    mLongitude.setText(location.getLongitude() + "");
}

@Override public void onStatusChanged(String provider, int status, Bundle extras) { }

@Override
public void onProviderEnabled(String provider) {
    Toast.makeText(this, "Enabled provider " + provider, Toast.LENGTH_SHORT).show();
}

@Override public void onProviderDisabled(String provider) {
    Toast.makeText(this, "Disabled provider " + provider, Toast.LENGTH_SHORT).show();
}
}

```

4. Location

a) Đối tượng Location

Đối tượng Location thể hiện một vị trí địa lý, có thể gồm: vĩ độ (latitude), kinh độ (longitude), nhãn thời gian (timestamp) và các thông tin khác như bearing³, cao độ, vận tốc. Các phương thức quan trọng của đối tượng Location:

```

float distanceTo(Location dest)
    Trả về khoảng cách tương đối tính bằng m giữa vị trí đang đứng đến vị trí chỉ định.

float getAccuracy()
    Lấy độ chính xác ước tính cho vị trí đang đứng, tính bằng m.

double getAltitude()
    Lấy cao độ nếu được, tính bằng m so với mặt nước biển WGS 84 (World Geodetic System).

float getBearing()
    Lấy bearing, tính bằng độ.

double getLatitude()
    Lấy vĩ độ, tính bằng độ.

double getLongitude()
    Lấy kinh độ, tính bằng độ.

float getSpeed()
    Lấy tốc độ nếu được, tính bằng m/s.

boolean hasAccuracy()
    Trả về true nếu vị trí đang đứng tính được độ chính xác.

```

³ Bearing là góc (theo chiều kim đồng hồ) giữa chính Bắc và đường thẳng hướng đến đích.

```

boolean hasAltitude()
    Trả về true nếu vị trí đang đúng có cao độ
boolean hasBearing()
    Trả về true nếu vị trí đang đúng có bearing.
boolean hasSpeed()
    Trả về true nếu vị trí đang đúng có tốc độ.
void reset()
    Xóa thông tin vị trí.
void setAccuracy(float accuracy)
    Thiết lập độ chính xác, tính bằng m.
void setAltitude(double altitude)
    Thiết lập cao độ, tính bằng m so với mặt nước biển.
void setBearing(float bearing)
    Thiết lập bearing, tính bằng độ.
void setLatitude(double latitude)
    Thiết lập vĩ độ, tính bằng độ.
void setLongitude(double longitude)
    Thiết lập kinh độ, tính bằng độ.
void setSpeed(float speed)
    Thiết lập tốc độ, tính bằng m/s.
String toString()
    Trả về chuỗi chứa mô tả ngắn gọn của đối tượng Location.

```

b) Chất lượng dịch vụ định vị

Đối tượng `LocationRequest` được dùng để yêu cầu chất lượng dịch vụ (QoS – Quality of Service) cho cập nhật vị trí từ `LocationClient`. Các phương thức sau dùng để kiểm soát QoS:

```

setExpirationDuration(long millis)
    Thiết lập thời gian yêu cầu, tính bằng miligiây.
setExpirationTime(long millis)
    Thiết lập thời gian quá hạn cho yêu cầu, tính bằng miligiây.
setFastestInterval(long millis)
    Thiết lập khoảng thời gian cập nhật vị trí nhanh nhất, tính bằng miligiây.
setInterval(long millis)
    Thiết lập khoảng thời gian cập nhật vị trí, tính bằng miligiây.
setNumUpdates(int numUpdates)
    Thiết lập số cập nhật vị trí.
setPriority(int priority)
    Thiết lập độ ưu tiên của yêu cầu định vị.

```

Ví dụ, nếu ứng dụng của bạn muốn định vị chính xác cao, nó phải tạo một yêu cầu định vị với `setPriority(int)` thiết lập thành `PRIORITY_HIGH_ACCURACY` và `setInterval(long)` là 5 giây.

Activity phải chú ý loại bỏ các yêu cầu định vị khi activity vào nền, ví dụ gọi `onPause()`, hoặc ít nhất giảm chất lượng yêu cầu, chuyển thành `priority PRIORITY_LOW_POWER` và thời gian `interval` lớn hơn.

Google Maps

1. Dùng Google Maps cài đặt sẵn

Bạn đơn giản gửi intent chứa thông tin geocoding cho ứng dụng Google Maps cài đặt sẵn trên thiết bị:

```

String geoCode = "geo:0,0?q=fpt+university+saigon";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(geoCode));
startActivity(intent);

```

Tìm địa điểm theo tọa độ:

```

String geoCode = "geo:10.8529544,106.6292685";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(geoCode));
startActivity(intent);

```

Đường đi giữa hai địa điểm:

```

String query = "http://maps.google.com/maps?saddr=10.8529544,106.6292685&daddr=10.85145123,106.63729191";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(query));
startActivity(intent);

```

Google Street View:

```

String geoCode = "google.streetview:cbll=41.5020952,-81.6789717&cbp=1,270,,45,1&mz=1";
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(geoCode));
startActivity(intent);

```

2. Google Maps API v2

Từ 2012, Google giới thiệu Google Maps API v2 cung cấp nhiều tính năng mới, thay thế phiên bản v1 đã lạc hậu. Bạn có thể tích hợp Google Maps API v2 vào ứng dụng của bạn, thực hiện theo các bước sau:

a) Chuẩn bị sử dụng Google Maps API trên AVD Genymotion

Bạn cần chú ý bảo đảm mọi thứ tương thích với phiên bản Android bạn đang làm việc.

Trên máy ảo Genymotion, bạn cần cài đặt:

- ARM Translation Installer v1.1 (<http://goo.gl/tfjiMt> hoặc <http://goo.gl/ZLKeKj>).
 - Tải về GApps (Google Applications) là gói ứng dụng cho nhiều thiết bị Android, tại http://wiki.rootzwiki.com/Google_Apps. Chú ý sử dụng phiên bản tương thích. Tên gói có dạng gapps-jb-yyyyMMdd-signed.zip.
- Giả sử bạn đã có tài khoản Google, kéo thả lượt hai gói trên vào màn hình Home của máy ảo Genymotion, nếu có thông báo nhắc, nhấn Ok. Sau khi cài đặt từng gói, khởi động lại máy ảo (đóng và chạy lại).
- Nếu cài đặt thành công, trên máy ảo sẽ có thêm các icon ứng dụng của Google như: Google+, Google Settings, v.v...

b) Google APIs và Google Play Services, Google Maps API key

Để tích hợp Google Maps vào ứng dụng của bạn, bạn cần cài đặt trước trong Android SDK:

- Google APIs tương thích với phiên bản đang phát triển.
- Google Play Services trong phần Extras của Android SDK Manager.

Những yêu cầu API từ ứng dụng trên thiết bị Android của bạn sẽ được gửi trực tiếp đến Google. Google xác thực nguồn gốc của yêu cầu từ chuỗi nhận dạng SHA-1 của giấy phép phát triển (certificate SHA-1 fingerprint) và tên package của ứng dụng. Bạn cần cung cấp thông tin trên và nhận về Google Maps API key, đưa vào ứng dụng của bạn.

Android Studio cung cấp cách đơn giản giúp sử dụng Google API v2. Theo các bước sau:

- Tạo project với Google Maps Activity, hoặc từ menu tắt của project, chọn New > Google > Google Maps Activity. Theo wizard, Android Studio sẽ chuyển bạn đến tập tin google_maps_api.xml.

Android Studio cũng đưa các dependency cần thiết vào project, giảm nhẹ công việc cấu hình bạn phải làm.

- Tập tin google_maps_api.xml chứa hướng dẫn rất cụ thể: certificate SHA-1 fingerprint và link để nhận về Google Maps API key. Thông tin đăng ký lấy từ khóa lưu trữ tại thư mục C:/Users/<User name>/ .android đưa vào tập tin này.

Theo đúng hướng dẫn, bạn sẽ nhận được Google Maps API Key, ví dụ: AIzaSyD7G-GonRp-VRHoJzeF5L3ij6IwwahquAA

Thay thế YOUR_KEY_HERE trong tập tin google_maps_api.xml với API Key nhận được.

c) MapFragment và GoogleMap

Android Studio tạo FragmentActivity với khung code có sẵn các phương thức: setUpMap() và setUpMapIfNeeded() để bạn thêm vào code cấu hình GoogleMap tùy ý muốn.

Google cung cấp MapFragment (hoặc SupportMapFragment) như một container tích hợp bản đồ vào ứng dụng Android.

Khai báo trong tập tin layout XML:

```
<fragment xmlns:android=http://schemas.android.com/apk/res/android
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Khi bạn cấu hình tập tin google_maps_api.xml, một số quyền và Google Maps API key được *tự động thêm* vào tập tin manifest, xem tập tin manifest để thấy những thay đổi:

```
<!--Permissions-->
<uses-permission android:name="com.twe.examples.permission.MAPS_RECEIVE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<!--Không cần để dùng Google Maps API v2, nhưng được đề nghị-->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<!--OpenGL ES v2, cần cho Google Maps API v2-->
<uses-feature android:glEsVersion="0x00020000"
    android:required="true" />

<application android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <!--API Key-->
    <meta-data android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyD7G-GonRp-VRHoJzeF5L3ij6IwwahquAA" />
</application>
```


Bạn cũng có thể thêm MapFragment vào activity bằng code, bằng phương thức add() của FragmentTransaction. Trước hết cần tạo một điểm nhập cho MapFragment:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

Thêm MapFragment trong onCreate() của activity. Từ MapFragment có được, bạn lấy đối tượng quan trọng nhất, thuộc lớp GoogleMap. Bạn cũng có thể thiết lập listener cho GoogleMap bằng cách cài đặt GoogleMap.OnMapClickListener.

```
public class MainActivity extends Activity implements GoogleMap.OnMapClickListener {
    private MapFragment mapFragment;
    private GoogleMap mMap;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // nếu lấy từ tập tin layout XML
        // mapFragment = (MapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment = MapFragment.newInstance();
        getSupportFragmentManager().beginTransaction().add(R.id.container, mapFragment).commit();
        mMap = mapFragment.getMap();
        mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
        mMap.setOnMapClickListener(this);
    }

    @Override protected void onResume() {
        super.onResume();
        int resultCode = GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
        if (resultCode == ConnectionResult.SUCCESS) {
            Toast.makeText(getApplicationContext(), "Service available!", Toast.LENGTH_LONG).show();
        } else {
            GooglePlayServicesUtil.getErrorDialog(resultCode, this, -1);
        }
    }

    // xử lý sự kiện tap lên bản đồ
    @Override public void onMapClick(LatLng point) {
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(point, 14));
    }
}
```

3. Tùy biến GoogleMap

a) Khởi tạo các tùy chọn hiển thị

Bạn có thể khởi tạo các tùy chọn hiển thị bằng các thuộc tính trong tập tin layout XML:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto" <!-- namespace cần cho map -->
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    map:cameraBearing="45" <!-- hướng lên trên tạo với chính Bắc góc 45°, xem địa bàn -->
    map:cameraTargetLat="10.8529544"
    map:cameraTargetLng="106.6292685"
    map:cameraTilt="0" <!-- nhìn vuông góc xuống bản đồ -->
    map:cameraZoom="14"
    map:mapType="normal"
    map:uiCompass="true" <!-- các UI control -->
    map:uiRotateGestures="true"
    map:uiScrollGestures="true"
    map:uiTiltGestures="true"
    map:uiZoomControls="true"
    map:uiZoomGestures="true" /> <!-- kích hoạt zoom gesture -->
```

hoặc thiết lập bằng code thông qua đối tượng GoogleMapOptions và CameraPosition:

```
GoogleMapOptions options = new GoogleMapOptions();
options.mapType(GoogleMap.MAP_TYPE_SATELLITE)
    .compassEnabled(true)
    .rotateGesturesEnabled(false)
    .tiltGesturesEnabled(false);
mapFragment = MapFragment.newInstance(options);
```

```
CameraPosition cameraPosition = new CameraPosition.Builder()
    .target(new LatLng(10.8529544, 106.6292685))
    .zoom(14).bearing(45).tilt(0).build();
mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
```

b) Thay đổi các tùy chọn hiển thị

Sau khi có đối tượng GoogleMap, bạn có thể thay đổi các tùy chọn hiển thị bằng cách gọi các phương thức của nó. Ví dụ:

- Thiết lập kiểu bản đồ bằng phương thức `setMapType()`. Có các kiểu, định nghĩa như hằng của lớp `GoogleMap`:

- + normal: `MAP_TYPE_NORMAL`, hiển thị các tuyến đường, còn gọi là kiểu road map.
- + hybrid: `MAP_TYPE_HYBRID`, ảnh vệ tinh với các tuyến đường vẽ thêm vào.
- + satellite: `MAP_TYPE_SATELLITE`, địa hình mặt đất từ ảnh vệ tinh.
- + terrain: `MAP_TYPE_TERRAIN`, dữ liệu địa hình (topographic), bao gồm màu phân vùng, các đường viền, nhãn.
- + none: `MAP_TYPE_NONE`, lưới rỗng không lát.

```
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

- Google Map API cung cấp các control tạo sẵn, tương tự như ứng dụng Google Maps trên thiết bị Android. Bạn có thể bật/tắt chúng bằng các phương thức của đối tượng `UiSettings`, lấy từ `GoogleMap.getUiSettings()`. Hầu hết các tùy chọn này cũng được cấu hình thông qua tập tin layout XML hoặc `GoogleMapOptions` như trình bày trong phần trên.

Mỗi control có một vị trí xác định trước, liên quan đến các cạnh của bản đồ, nhưng bạn có thể di chuyển chúng bằng cách đệm thêm với `GoogleMap.setPadding()`.

Các UI control gồm:

- Zoom** công cụ thu nhỏ/phóng to, mặc định ẩn, hiển thị bằng: `UiSettings.setZoomControlsEnabled(true)`.
- Compass** la bàn xuất hiện khi camera được định hướng với bearing hoặc tilt khác 0, khi tap lên compass, camera dịch chuyển cho bearing và title trở về 0 và compass biến mất.
- My Location** xuất hiện khi kích hoạt layer My Location bằng `GoogleMap.setMyLocationEnabled(true)`. Click lên nút sẽ kích hoạt listener `GoogleMap.OnMyLocationButtonClickListener`. Bạn có thể ẩn nút này bằng cách gọi `UiSettings.setMyLocationButtonEnabled(false)`. Nếu dùng Genymotion, trước đó cần bật GPS và tạo vị trí giả cho thiết bị bằng MAP trong GPS.
- Level picker** xuất hiện khi người dùng xem một indoor map.
- Map toolbar** xuất hiện khi tap lên marker, tap lên icon của toolbar sẽ gửi intent đến ứng dụng Google Maps trên thiết bị.

```
mMap.setMyLocationEnabled(true);
mMap.setOnMyLocationChangeListener(new GoogleMap.OnMyLocationChangeListener() {
    @Override public void onMyLocationChange(Location arg0) {
        mMap.addMarker(new MarkerOptions().position(new LatLng(arg0.getLatitude(), arg0.getLongitude()))
            .title("It's Me!"));
    }
});
```

- Đặt một marker với văn bản kèm theo để đánh dấu vị trí của bạn trên bản đồ:

```
final LatLng myPoint = new LatLng(10.8529544, 106.6292685);
mMap.addMarker(new MarkerOptions().position(myPoint).title("FPT University"));
```

Muốn đánh dấu nhiều vị trí, chỉ cần gọi `addMaker()` nhiều lần với những vị trí ta muốn đặt. Để xóa các vị trí đánh dấu, gọi phương thức `GoogleMap.clear()`.

Một số phương thức khác của `GoogleMap`:

Phương thức	Mục đích
<code>addCircle(CircleOptions options)</code>	Thêm một vòng tròn trên bản đồ.
<code>addPolygon(PolygonOptions options)</code>	Thêm một đa giác trên bản đồ.
<code>addTileOverlay(TileOverlayOptions options)</code>	Thêm lớp lát trên bản đồ.
<code>animateCamera(CameraUpdate update)</code>	Di chuyển bản đồ theo cập nhật với hình ảnh động.
<code>clear()</code>	Loại bỏ mọi thứ trên bản đồ.
<code>getMyLocation()</code>	Trả về vị trí người dùng đang được hiển thị.
<code>moveCamera(CameraUpdate update)</code>	Định vị lại camera theo hướng dẫn định nghĩa trong update.
<code>setTrafficEnabled(boolean enabled)</code>	Bật/tắt lớpTraffic.
<code>snapshot(GoogleMap.SnapshotReadyCallback callback)</code>	Lấy một bản chụp bản đồ.
<code>stopAnimation()</code>	Dừng hình ảnh động của camera.

c) Map gestures

`GoogleMap` cũng hỗ trợ một số gesture, giống như UI control, bạn có thể bật/tắt gesture thông qua lớp `UiSettings`.

Các gesture gồm:

- Zoom** có các gesture sau: double-tap để phóng đại (zoom in) một cấp; tap cùng lúc hai ngón để thu nhỏ (zoom out) một cấp; pinch/stretch để thu nhỏ/phóng to; zoom 1 ngón bằng cách double-tap rồi giữ luôn ngón khi tap lần hai, sau đó lướt lên/xuống để phóng to/thu nhỏ.
Tắt bằng `UiSettings.setZoomGesturesEnabled(false)`.
- Scroll (pan)** mặc định, dùng kéo xem các vùng bản đồ, tắt bằng `UiSettings.setScrollGesturesEnabled(false)`.
- Tilt** đặt hai ngón tay lên bản đồ và chuyển chúng lên xuống để tăng/giảm góc tilt (góc nhìn vuông góc bản đồ). Tắt bằng `UiSettings.setTiltGesturesEnabled(false)`.
- Rotate** đặt hai ngón tay lên bản đồ và xoay chúng cùng chiều (khó thực hiện trên thiết bị giả lập) để xoay bản đồ. Tắt bằng `UiSettings.setRotateGesturesEnabled(false)`.

4. GoogleApiClient

Đa số tác vụ bạn có thể thực hiện với sự hỗ trợ của `com.google.android.gms.maps.GoogleMap` và các listener của nó. Tuy nhiên, Google Maps API v2 còn cung cấp `com.google.android.gms.common.api.GoogleApiClient` cũng tiện dụng cho định vị.

`GoogleApiClient` tạo từ `GoogleApiClient.Builder()`. Builder sẽ cấu hình `GoogleApiClient` bằng cách thiết lập hai interface callback: `GoogleApiClient.ConnectionCallbacks` và `GoogleApiClient.OnConnectionFailedListener`.

Các phương thức cần cài đặt cho hai interface:

`void onConnected(Bundle bundle)`

Được gọi khi dịch vụ định vị đã kết nối thành công với `GoogleApiClient`.

`void onDisconnected(int arg0)`

Được gọi khi client ngắt kết nối. Bạn dùng `disconnect()` để ngắt kết nối từ `GoogleApiClient`.

`void onConnectionFailed(ConnectionResult result)`

Được gọi khi có lỗi kết nối từ `GoogleApiClient` đến dịch vụ định vị.

`GoogleApiClient` kết nối với dịch vụ định vị `LocationServices` bằng phương thức `connect()`, thường gọi trong `onResume()` của activity. Sau đó gọi phương thức `LocationServices.FusedLocationApi.getLastLocation()` để trả về vị trí cuối của thiết bị dưới hình thức một đối tượng `Location`, chứa vĩ độ, kinh độ và các thông tin khác.

Khi bạn có đối tượng `Location`, bạn có thể dùng phương thức `Geocoder.getFromLocation()` để lấy địa chỉ. Phương thức này là đồng bộ, có thể chiếm nhiều thời gian, vì vậy bạn nên gọi từ phương thức `doInBackground()` của một lớp `AsyncTask`.

```
public class MapsActivity extends FragmentActivity implements
    GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener {
    protected GoogleApiClient mClient;
    protected GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();
        mMap.setMyLocationEnabled(true);
        buildGoogleApiClient();
    }

    protected synchronized void buildGoogleApiClient() {
        mClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    }

    @Override
    protected void onResume() {
        super.onResume();
        mClient.connect();
    }

    @Override
    public void onConnected(Bundle bundle) {
        Location location = LocationServices.FusedLocationApi.getLastLocation(mClient);
        if (location != null) {
            LatLng myPoint = new LatLng(location.getLatitude(), location.getLongitude());
            mMap.animateCamera(CameraUpdateFactory.newLatLng(myPoint));
        } else {
            Toast.makeText(this, "No location detected!", Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onConnectionSuspended(int arg0) {
        Toast.makeText(this, "Connection suspended", Toast.LENGTH_SHORT).show();
        mClient.connect();
    }

    @Override
    public void onConnectionFailed(ConnectionResult result) {
        Toast.makeText(this, "Connection failed: " + result.getErrorCode(), Toast.LENGTH_SHORT).show();
    }
}
```

Hardware

Sensor

1. Sensor

Đa số các thiết bị Android đều được trang bị các sensor (cảm biến) có sẵn, bao gồm ba loại:

- Cảm biến chuyển động: cảm biến gia tốc (accelerometer), cảm biến trọng lực (gravity), con quay hồi chuyển (gyroscope), cảm biến vector quay. Các cảm biến này cung cấp thông tin chuyển động của thiết bị.
 - Cảm biến môi trường: khí áp kế (barometer), quang kế (photometer) và nhiệt kế (thermometer). Các cảm biến này cung cấp các thông số môi trường như nhiệt độ, ánh sáng.
 - Cảm biến vị trí: cảm biến định hướng (orientation) và từ trường (magnetometer). Chúng cung cấp vị trí vật lý của thiết bị.
- Android cho phép bạn nhận dữ liệu đo lường từ các cảm biến và sử dụng dữ liệu này trong ứng dụng của bạn. Ví dụ bạn có thể xây dựng ứng dụng trò chơi, lấy thông tin chuyển động của thiết bị chuyển thành thông tin điều khiển đối tượng trong trò chơi. Android cung cấp một số lớp và giao diện (sensor framework) giúp bạn thực hiện một loạt các vấn đề liên quan đến cảm biến:
- Xác định số cảm biến có sẵn trên thiết bị.
 - Xác định khả năng của một cảm biến cụ thể, như nhu cầu pin, độ phân giải, phạm vi hoạt động.
 - Nhận dữ liệu đo lường từ cảm biến.
 - Đăng ký và hủy đăng ký các listener lắng nghe và theo dõi thay đổi từ cảm biến.

Danh sách các cảm biến có trên thiết bị được lấy bằng phương thức `getSensorList()` của `SensorManager`:

```
public class SensorListActivity extends ListActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SensorManager manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ALL);
        List<String> types = new ArrayList<String>();
        for (Sensor sensor : sensors)
            types.add(sensor.getName());
        setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, types));
        getListView().setTextFilterEnabled(true);
    }
}
```

Các cảm biến được truy cập theo một cách chung:

- Lấy `SensorManager` từ `getSystemService(SENSOR_SERVICE)`.
- Các hằng thuộc lớp `Sensor` mô tả kiểu cảm biến, được truyền như tham số đến phương thức `getDefaultSensor()` của lớp `SensorManager`, để lấy cảm biến mặc định cho kiểu cảm biến đó. Nếu kiểu cảm biến đó có nhiều hơn một cảm biến, dùng phương thức `getSensorList()` với tham số là kiểu cảm biến để lấy danh sách các cảm biến cùng kiểu.
- Khi lấy được cảm biến, ứng dụng bắt đầu lắng nghe sự kiện từ cảm biến. Mỗi cảm biến báo (report) sự kiện theo cách khác nhau. Trước khi lắng nghe cảm biến, phải kiểm tra chế độ báo sự kiện của cảm biến và chọn chiến lược lắng nghe thích hợp. Mỗi cảm biến chỉ có một chế độ báo sự kiện, có thể lấy bằng phương thức `getReportingMode()` của `Sensor`.

```
int reportingMode = sensor.getReportingMode();
```

Lớp `Sensor` cung cấp các hằng mô tả các chế độ báo sự kiện:

Hằng	Mô tả
REPORTING_MODE_CONTINUOUS	Sự kiện được báo với thời gian định kỳ được thiết lập bởi ứng dụng.
REPORTING_MODE_ON_CHANGE	Sự kiện được báo chỉ khi đo lường thay đổi.
REPORTING_MODE_ONE_SHOT	Sự kiện được báo trong chế độ one-shot. Khi phát hiện sự kiện, cảm biến tự tắt.
REPORTING_MODE_SPECIAL_TRIGGER	Sự kiện được báo với kích hoạt chỉ định trong mô tả của cảm biến.

+ Với chế độ continuous và on-change, ứng dụng cần đăng ký `SensorEventListener` để nhận sự kiện. Phải cài đặt hai phương thức callback xử lý sự kiện: `onAccuracyChanged()` và `onSensorChanged()`. `SensorEventListener` được đăng ký với kiểu cảm biến chỉ định cho `SensorManager`, bằng phương thức `registerListener()`. Tham số cuối truyền cho phương thức này là thời gian trễ đề nghị để màn hình thay đổi phù hợp, có thể là trị (miligiây), hoặc các hằng khai báo trong lớp `SensorManager`:

Hằng	Mô tả
SENSOR_DELAY_NORMAL	Thời gian trễ mặc định.
SENSOR_DELAY_UI	Thời gian trễ phù hợp cho mục tiêu hiển thị.
SENSOR_DELAY_GAME	Thời gian trễ phù hợp cho trò chơi.
SENSOR_DELAY_FASTEST	Thời gian trễ nhanh nhất.

Khi ứng dụng không còn lắng nghe cảm biến, hủy đăng ký listener bằng cách gọi phương thức `unregisterListener()` của `SensorManager`. Điều này cần thiết do cảm biến tiêu thụ pin nhiều.

```
public class MainActivity extends Activity implements SensorEventListener {
    private final SensorManager manager;
    private final Sensor sensor;

    public onCreate() {
        manager = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensor = manager.getDefaultSensor(Sensor.TYPE_GENERIC);
    }

    protected void onResume() {
        super.onResume();
    }
}
```

```

manager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL);
}

protected void onPause() {
    super.onPause();
    manager.unregisterListener(this);
}

public void onAccuracyChanged(Sensor sensor, int accuracy) { }
public void onSensorChanged(SensorEvent event) { }
}

```

Phương thức callback xử lý sự kiện nhận đối tượng SensorEvent chứa thông tin sự kiện. Lớp này có các thành viên: accuracy, sensor (nguồn của sự kiện), timestamp (nhãn thời gian, tính bằng nanogiây) và values (dữ liệu đo lường). values của SensorEvent là một mảng có cấu trúc khác nhau tùy theo kiểu cảm biến, nguồn phát ra SensorEvent. Ví dụ, với TYPE_GRAVITY, values chứa 3 trị tương ứng với trọng lực theo 3 trục (x, y, z) tính bằng m/s²; với TYPE_PRESSURE, values chứa 1 trị là áp suất không khí tính bằng hPa hoặc mbar.

+ Với chế độ one-shot và trigger, cảm biến không đo liên tục mà được kích hoạt bởi một sự kiện nào đó. Ví dụ, cảm biến chuyển động đủ lớn (significant motion) báo khi một chuyển động vượt ngưỡng xảy ra, sau đó cảm biến tự động tắt.

Với chế độ báo này, ứng dụng cần đăng ký listener TriggerEventListener và cài đặt phương thức callback onTrigger().

```

public class OneShotListener extends TriggerEventListener {
    @Override public void onTrigger(TriggerEvent triggerEvent) { }
}

```

Do cảm biến kiểu này không luôn chạy, ứng dụng phải yêu cầu dữ liệu đo lường từ cảm biến khi cần, bằng cách gọi phương thức requestTriggerSensor() của SensorManager, phương thức này nhận TriggerEventListener và Sensor làm tham số.

Nếu sự kiện vẫn chưa xuất hiện mà không cần đo lường nữa, gọi phương thức cancelTriggerSensor() của SensorManager, với cùng tham số như trên.

```

manager.requestTriggerSensor(oneShotListener, sensor);
manager.cancelTriggerSensor(oneShotListener, sensor);

```

2. Giả lập sensor trên AVD

Nói chung, các AVD hiện nay không hỗ trợ giả lập các cảm biến, bạn phải dùng chương trình giả lập cảm biến OpenIntents Sensor Simulator (<http://code.google.com/p/openintents/wiki/SensorSimulator>). Đây là công cụ mã nguồn mở cho phép mô phỏng dữ liệu cảm biến, hỗ trợ giả lập gia tốc, từ trường, định hướng, nhiệt độ, ánh sáng, áp lực, gia tốc tuyến tính, vector quay và cảm biến con quay hồi chuyển. Sensor Simulator chạy như một server, dữ liệu giả lập do người dùng tạo ra sẽ được chuyển đến thiết bị kết nối đến nó.

- Tải OpenIntents Sensor Simulator từ địa chỉ trên, giải nén và chạy /bin/sensorsimulator-xxx.jar. Dùng mouse điều khiển mô hình thiết bị, bạn sẽ thấy các thông số giả lập được tạo ra. Khi ứng dụng của bạn chạy, nó kết nối đến server giả lập này để nhận các thông số giả lập cho cảm biến của nó.

- Truy cập Sensor Simulator từ ứng dụng:

+ Chọn tool window Project, chọn chế độ xem là Project. Kéo thả /lib/sensorsimulator-xxx.jar đến thư mục app/libs của project.

+ Do ứng dụng phải kết nối đến server giả lập, cần thêm quyền trong tập tin manifest:

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

Chú ý là ứng dụng của bạn phải kết nối đến server giả lập, vì vậy dùng StrictMode. Với thiết bị thật, không cần điều này.

+ Thay thế các đối tượng trong sensor framework của Android bằng các đối tượng tương ứng trong thư viện trên.

Trong ví dụ minh họa sau, chú ý các khối chú thích là để dùng cho thiết bị thật:

```

// SENSOR SIMULATOR
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
import android.os.StrictMode;
// REAL DEVICE
// import android.hardware.Sensor;
// import android.hardware.SensorEvent;
// import android.hardware.SensorEventListener;
// import android.hardware.SensorManager;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity implements SensorEventListener {
    // SENSOR SIMULATOR
    private SensorManagerSimulator manager;
    // REAL DEVICE
    // private SensorManager manager;
    TextView mTextView;
}

```



```

@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTextView = (TextView) findViewById(R.id.text_id);
    // SENSOR SIMULATOR
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
    manager = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
    manager.connectSimulator();
    // REAL DEVICE
    // manager = (SensorManager) getSystemService(SENSOR_SERVICE);
}

@Override protected void onResume() {
    super.onResume();
    // SENSOR SIMULATOR
    manager.registerListener(this, manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManagerSimulator.SENSOR_DELAY_NORMAL);
    manager.registerListener(this, manager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManagerSimulator.SENSOR_DELAY_NORMAL);
    manager.registerListener(this, manager.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
        SensorManagerSimulator.SENSOR_DELAY_NORMAL);

    // REAL DEVICE
    // manager.registerListener(this,
    //     manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER), SensorManager.SENSOR_DELAY_NORMAL);
    // manager.registerListener(this,
    //     manager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD), SensorManager.SENSOR_DELAY_FASTEST);
    // manager.registerListener(this,
    //     manager.getDefaultSensor(Sensor.TYPE_GYROSCOPE), SensorManager.SENSOR_DELAY_FASTEST);
}

@Override protected void onPause() {
    manager.unregisterListener(this);
    super.onPause();
}

public void onAccuracyChanged(Sensor sensor, int accuracy) {
}

@Override public void onSensorChanged(SensorEvent event) {
    synchronized (this) {
        // SENSOR SIMULATOR
        int sensor = event.type;
        // REAL DEVICE
        // int sensor = event.sensor.getType();
        float[] values = event.values;
        switch (sensor) {
            case Sensor.TYPE_ACCELEROMETER:
                mTextView.setText(values[0] + " : " + values[1] + " : " + values[2]);
                break;
            case Sensor.TYPE_MAGNETIC_FIELD: break;
            case Sensor.TYPE_GYROSCOPE: break;
            default:
        }
    }
}
}
}

```

Media

1. Audio

a) AudioManager

Android cung cấp audio thông qua đối tượng android.media.AudioManager, bạn lấy thực thể AudioManager bằng cách yêu cầu service hệ thống Context.AUDIO_SERVICE:

```
AudioManager manager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
```

b) Thiết bị audio

Khi có được AudioManager, bạn có thể quản lý được các thiết bị audio, có sẵn như speakerphone và microphone hoặc gắn thêm bên ngoài như bluetooth headset. AudioManager cung cấp các phương thức để kiểm tra trạng thái hiện hành của chúng, kích hoạt hoặc bất hoạt chúng.

- Microphone

Khi dùng AVD, kiểm tra microphone có tồn tại không:

```
private boolean hasMicrophone() {
    PackageManager manager = this.getPackageManager();
    return manager.hasSystemFeature(PackageManager.FEATURE_MICROPHONE);
}
```

Microphone cho phép thiết bị Android nhận audio vào. Khi không cần audio vào, microphone có thể tắt (muted).

```
if (!manager.isMicrophoneMute()) {
    manager.setMicrophoneMute(true);
}
```

- Speakerphone cho phép audio ra để nghe từ khoảng cách gần mà không cần dùng tai nghe. Ứng dụng có thể kiểm tra audio đang phát thông qua speakerphone và bật hoặc tắt nó.

```
if (!manager.isSpeakerphoneOn()) {
    manager.setSpeakerphoneOn(true);
}
```

c) Audio stream

Android phân biệt các audio stream dựa vào mục đích của nó, Android cho phép mỗi nhóm audio stream được điều khiển riêng. Điều này cho phép người dùng thiết lập âm lượng khác nhau cho các loại audio khác nhau như âm thanh báo thức, tiếng chuông điện thoại, âm nhạc.

Lớp AudioManager cung cấp các hằng cho từng loại audio được hỗ trợ:

Hằng	Audio stream
STREAM_ALARM	Âm thanh báo thức.
STREAM_DTMF	Âm DTMF (dual-tone multi-frequency) khi nhấn các phím điện thoại.
STREAM_MUSIC	Âm nhạc.
STREAM_NOTIFICATION	Âm thanh cảnh báo.
STREAM_RING	Âm thanh chuông điện thoại.
STREAM_SYSTEM	Âm thanh hệ thống.
STREAM_VOICE_CALL	Âm thanh gọi điện thoại.
USE_DEFAULT_STREAM_TYPE	Âm thanh mặc định.

Âm lượng (volume) của audio stream có thể lấy và hiệu chỉnh thông qua lớp AudioManager:

```
int currentVolume = manager.getStreamVolume(AudioManager.STREAM_MUSIC);
int maxVolume = manager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
manager.setStreamVolume(AudioManager.STREAM_MUSIC, 10,
    AudioManager.FLAG_SHOW_UI | AudioManager.FLAG_PLAY_SOUND);
```

Ứng dụng hiệu chỉnh âm lượng bằng cách cung cấp một trị giữa 0 và maxVolume lấy được, trong ví dụ trên là trị 10. Phương thức thiết lập âm lượng nhận tham số cuối là một tập các cờ:

Flags	Mô tả
FLAG_ALLOW_RINGER_MODE	Bao gồm cả chế độ chuông (ringer mode) khi thay đổi âm lượng.
FLAG_PLAY_SOUND	Hiệu ứng âm thanh khi thay đổi âm lượng.
FLAG_REMOVE_SOUND_AND_VIBRATE	Ngắt âm thanh hoặc rung khi thay đổi âm lượng.
FLAG_SHOW_UI	Hiển thị Toast chứa âm lượng hiện tại.
FLAG_VIBRATE	Rung nếu thiết lập đi vào chế độ rung (vibrate mode).

Thay vì hạ thấp âm lượng của audio stream về 0, có thể tắt trực tiếp audio stream:

```
manager.setStreamMute(AudioManager.STREAM_NOTIFICATION, true);
```

Để phát một audio stream mà không bị nhiễu bởi các stream audio khác, AudioManager cho phép phát audio stream chỉ định trong lúc cấm các audio stream khác:

```
if (!manager.isMusicActive()) {
    manager.setStreamSolo(AudioManager.STREAM_VOICE_CALL, true);
}
```

d) Phát audio

Android cung cấp lớp MediaPlayer cho phép phát dữ liệu đa phương tiện như audio và video.

- Trước hết, nạp nội dung media từ các nguồn dữ liệu chỉ định như từ một URL hoặc trực tiếp từ tập tin. Cấu hình cho MediaPlayer, ví dụ chỉ định loại audio stream của nội dung media đó. Cuối cùng gọi phương thức prepare() để chuẩn bị đệm (buffering) cho nội dung media

```
MediaPlayer player = new MediaPlayer();
player.setDataSource("http://www.example.com/sound.mp3");
player.setAudioStreamType(AudioManager.STREAM_MUSIC);
player.prepare();
```

- Khi MediaPlayer chuẩn bị đệm thành công, bắt đầu phát audio bất cứ lúc nào bằng cách gọi phương thức start() của nó. Sau đó, có thể dừng bất cứ lúc nào bằng cách gọi phương thức stop(). Khi dừng, trước khi phát lại cần chuẩn bị (prepare) lần nữa.

- Khi không phát nữa, giải phóng tài nguyên cho MediaPlayer bằng cách gọi phương thức release().

Nếu nội dung media được tải từ mạng hoặc từ SD card trên UI thread, ứng dụng có thể không đáp ứng tương tác. Android cung cấp AsyncPlayer, cũng dựa trên MediaPlayer để phát. Tuy nhiên, AsyncPlayer chạy MediaPlayer trong một thread riêng, nó giữ vai trò như một cầu nối giữa ứng dụng và MediaPlayer.

Sử dụng AsyncPlayer đơn giản hơn MediaPlayer nhiều. Bạn có thể phát trực tiếp, không quan tâm đến các vấn đề của MediaPlayer:

```
AsyncPlayer player = new AsyncPlayer("Audio Player");
player.play(this, Uri.parse("http://www.example.com/sound.mp3"), false, AudioManager.STREAM_MUSIC);
```

Nếu cần phát nhiều mẫu audio, thường xuyên và yêu cầu độ trễ thấp, Android cung cấp SoundPool. SoundPool có thể nạp một tập hợp các mẫu audio từ tài nguyên của ứng dụng vào bộ nhớ. Sau khi nạp, ta phát chúng bất cứ lúc nào, với độ trễ thấp. Dùng SoundPool là giải pháp tốt nhất cho việc tích hợp hiệu ứng âm thanh vào ứng dụng Android.

- Tạo SoundPool

Từ API Level 20 trở về trước:

```
SoundPool soundPool = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);
```

Thủ tục tạo SoundPool đã thay đổi từ API Level 21, dùng lớp SoundPool.Builder. Lớp buider này dùng lại dùng lớp AudioAttributes.Builder cho phép ứng dụng chỉ định thêm thông tin cho audio stream.

```
SoundPool soundPool = new SoundPool.Builder()
    .setMaxStreams(2)
    .setAudioAttributes(
        new AudioAttributes.Builder()
            .setContentType(AudioAttributes.CONTENT_TYPE_MOVIE)
            .setUsage(AudioAttributes.USAGE_MEDIA)
            .build()
    )
    .build();
```

- Nạp các mẫu audio vào SoundPool

Lớp SoundPool cung cấp một tập các phương thức cho phép nạp các mẫu audio từ nhiều nguồn khác nhau: tài nguyên raw của ứng dụng, tài nguyên assets của ứng dụng, tập tin. Các phương thức nạp này trả về một định danh duy nhất, gọi là sound ID, cho mỗi mẫu audio nạp vào SoundPool. Định danh này dùng để tham chiếu đến mẫu audio cụ thể cần phát. Nếu có sự cố khi nạp mẫu audio vào SoundPool, định danh trả về sẽ là 0.

```
int soundId1 = soundPool.load(this, R.raw.sound, 1);
int soundId2 = soundPool.load(getAssets().openFd("sound.mp3"), 1);
int soundId3 = soundPool.load("/sdcard/sound.mp3", 1);
```

Nếu mẫu audio nào không còn được dùng, ứng dụng dùng phương thức unload() với sound ID chỉ định để loại mẫu audio đó ra khỏi SoundPool. Nếu SoundPool không còn được dùng, có thể giải phóng nó bằng phương thức release().

- Phát audio từ SoundPool

Khi các mẫu audio đã nạp vào SoundPool, phát chúng bất kỳ lúc nào bằng phương thức play(), với tham số là sound ID của mẫu audio chỉ định. Phương thức này cũng cung cấp thêm nhiều tham số, cho phép những tùy chỉnh như âm lượng, độ ưu tiên của stream, chế độ lặp, tốc độ phát.

```
soundPool.play(
    soundId1, // sound ID
    1,        // âm lượng loa trái thiết lập tối đa
    1,        // âm lượng loa phải thiết lập tối đa
    0,        // audio stream có độ ưu tiên thấp
    0,        // không lặp
    1         // tốc độ phát mặc định
);
```

Nếu phương thức play() thành công, nó trả về một stream ID khác 0. Stream ID có thể được dùng để điều chỉnh một số tham số khi audio đang phát. Ví dụ, nếu chế độ lặp được yêu cầu, ứng dụng dùng phát bằng cách triệu gọi phương thức stop() và cung cấp stream ID đó cho stop().

e) Ghi audio

Android cung cấp chức năng ghi âm, dùng lớp MediaRecorder và cần cấp quyền android.permission.RECORD_AUDIO.

- Trước hết, cần cấu hình nguồn audio để ghi âm.

```
MediaRecorder recorder = new MediaRecorder();
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

Lớp MediaRecorder.AudioSource chứa các hằng định nghĩa cho các nguồn audio:

Hằng	Nguồn audio
CAMCORDER	Nguồn audio từ microphone đi cùng camera nếu thiết bị có hỗ trợ.
DEFAULT	Nguồn audio mặc định.
MIC	Nguồn audio từ microphone.
REMOTE_SUBMIX	Nguồn audio trộn từ nhóm (submix) các audio stream.
VOICE_CALL	Nguồn voice uplink và downlink.
VOICE_COMMUNICATION	Microphone được điều chỉnh cho liên lạc voice.
VOICE_DOWNLINK	Nguồn voice downlink.
VOICE_RECOGNITION	Microphone được điều chỉnh cho nhận dạng voice.
VOICE_UPLINK	Nguồn voice uplink.

- Cấu hình audio xuất cho ghi âm.

Âm thanh thu được sẽ được nén và mã hóa bằng cách dùng audio encoder (bộ mã hóa âm thanh). Tiếp theo, âm thanh sau khi mã hóa sẽ được lưu vào tập tin với định dạng lưu trữ (container format) thích hợp.

Lớp `MediaRecorder.AudioEncoder` cung cấp các hằng cho các audio encoder được hỗ trợ.

Lớp `MediaRecorder.OutputFormat` cung cấp các hằng cho các định dạng lưu trữ được hỗ trợ.

```
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setOutputFile(Environment.getExternalStorageDirectory().getAbsolutePath() + "/myaudio.3gp");
```

Phải thêm các quyền sau vào tập tin manifest:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- Tiến hành ghi âm.

Sau khi cấu hình `MediaRecorder`, chuẩn bị ghi âm bằng phương thức `prepare()`, sau đó tiến hành ghi âm bằng phương thức `start()`.

```
recorder.prepare();
recorder.start();
```

- Dừng ghi âm bất cứ lúc nào bằng cách gọi phương thức `stop()`. Gọi phương thức `reset()` trước khi ghi âm tiếp (`prepare`) hoặc khi không cần ghi âm nữa, giải phóng tài nguyên cho `MediaRecorder` bằng cách gọi phương thức `release()`.

2. Video

`MediaPlayer` dùng phát audio cũng hỗ trợ phát video với chuỗi lời gọi API tương tự. Phát video cũng cần một đối tượng `Surface` để vẽ các frame video. Đơn thể UI `SurfaceView`, tích hợp vào UI layout, sẽ cung cấp `Surface`:

```
<SurfaceView android:id="@+id/surface"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

`SurfaceView` giữ chỗ cho một đối tượng `Surface`. Ứng dụng đăng ký `SurfaceView`, trong `onCreate()` của activity, để nhận các dữ kiện trong vòng đời của `Surface`, thông qua đối tượng `SurfaceHolder`.

```
SurfaceView surfaceView = (SurfaceView) findViewById(R.id.surface);
surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() {
    @Override public void surfaceCreated(SurfaceHolder surfaceHolder) {
        startPlayback(surfaceHolder);
    }
    @Override public void surfaceChanged(SurfaceHolder surfaceHolder, int i, int i2, int i3) { }
    @Override public void surfaceDestroyed(SurfaceHolder surfaceHolder) {
        stopPlayback();
    }
});
```

a) Phát video

Bên trong phương thức callback `surfaceCreate()`, ứng dụng lấy đối tượng `Surface` từ `SurfaceHolder` được cung cấp bằng phương thức `getSurface()`. Thiết lập `Surface` có được cho `MediaPlayer` rồi phát video.

```
private void startPlayback(SurfaceHolder surfaceHolder) {
    player = new MediaPlayer();
    player.setSurface(surfaceHolder.getSurface());
    player.setScreenOnWhilePlaying(true);
    try {
        player.setDataSource("http://www.example.com/movie.mp4");
    } catch (IOException e) {
        e.printStackTrace();
        releasePlayer();
        return;
    }
    player.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
        @Override public void onPrepared(MediaPlayer mediaPlayer) {
            mediaPlayer.start();
        }
    });
    player.prepareAsync();
}
```

Có thể dừng video đang phát bất cứ lúc nào, giống như dừng audio:

```
private void stopPlayback() {
    if (mediaPlayer != null) {
        mediaPlayer.stop();
        releasePlayer();
    }
}

private void releasePlayer() {
    mediaPlayer.release();
    mediaPlayer = null;
}
```

b) Ghi video

Giống như ghi audio, ghi video được xử lý bởi MediaRecorder, MediaRecorder cũng cần đối tượng Surface để hiển thị xem trước (preview) trong khi ghi video, cách lấy Surface giống với phát video.

```
recorder.setPreviewDisplay(surfaceHolder.getSurface());
```

MediaRecorder ghi trực tiếp từ camera của thiết bị hoặc từ một Surface khác. Lớp MediaRecorder.VideoSource cung cấp các hằng cho từng nguồn video, dùng thiết lập nguồn video cho MediaRecorder:

```
recorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

Video thu được sẽ được nén và mã hóa với một video encoder chỉ định. Lớp MediaRecorder.VideoEncoder cung cấp các hằng cho từng video encoder được hỗ trợ, dùng thiết lập video encoder cho MediaRecorder:

```
recorder.setVideoEncoder(MediaRecorder.VideoEncoder.MPEG_4_SP);
```

c) Phát video bằng VideoView và MediaController

Cách đơn giản nhất để phát video trong ứng dụng Android là dùng lớp android.widget.VideoView. VideoView là một đơn thể UI có thể thêm vào layout, cung cấp một bề mặt (surface) để phát video trên đó. VideoView có một tập phong phú các phương thức hỗ trợ phát video:

setVideoPath(String path)

Chỉ định đường dẫn của video, có thể là URL chỉ đến video từ xa hoặc đường dẫn chỉ đến tập tin video trên thiết bị.

setVideoUri(Uri uri)

Giống setVideoPath() nhưng dùng với tham số là Uri.

start()

Bắt đầu phát video.

stopPlayback()

Ngừng video đang phát.

pause()

Tạm dừng video đang phát.

isPlaying()

Trả về trị Boolean cho biết video có đang phát hay không.

setOnPreparedListener(MediaPlayer.OnPreparedListener listener)

Thiết lập listener sẽ gọi phương thức callback khi video sẵn sàng phát.

setOnErrorListener(MediaPlayer.OnErrorListener listener)

Thiết lập listener sẽ gọi phương thức callback khi có lỗi xuất hiện trong thời gian phát video.

setOnCompletionListener(MediaPlayer.OnCompletionListener listener)

Thiết lập listener sẽ gọi phương thức callback khi video kết thúc phát.

getDuration()

Trả về thời lượng của video, tính bằng miligiây, thường gọi bên trong OnPreparedListener.

getCurrentPosition()

Trả về trị nguyên chỉ vị trí đang phát trong video.

setMediaController(MediaController controller)

Thiết lập MediaController cho phép điều khiển phát video.

Để hỗ trợ phát video, lớp android.widget.MediaController cung cấp một tập các điều khiển cho phép người dùng điều khiển quá trình phát (play, pause, dịch chuyển anchor). Một số phương thức của lớp này:

setAnchorView(View view)

Thiết lập anchor chỉ vị trí đang phát trong video. Người dùng có thể dịch chuyển anchor.

show()

Hiển thị các control. Mặc định, tap vào video mới hiển thị các control.

show(int timeout)

Các control hiển thị sau một khoảng thời gian, tính bằng miligiây.

hide()

Ẩn các control từ người dùng.

isShowing()

Trả về trị Boolean chỉ các control đang hiện hoặc ẩn.

```
public class PlayVideoActivity extends ActionBarActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final VideoView mVideoView = (VideoView) findViewById(R.id.videoview_id);
        mVideoView.setVideoPath("http://www.example.com/movie.mp4");
        MediaController controller = new MediaController(this);
        controller.setAnchorView(mVideoView);
        mVideoView.setMediaController(controller);
        mVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
            @Override public void onPrepared(MediaPlayer mp) {
                Toast.makeText(getBaseContext(), "Duration: " + mVideoView.getDuration(), Toast.LENGTH_SHORT).show();
            }
        });
        mVideoView.start();
    }
}
```

Phải thêm quyền sau vào tập tin manifest:

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

d) Ghi video bằng MediaStore

Cách đơn giản nhất để ghi video là gửi intent đến MediaStore.ACTION_VIDEO_CAPTURE. Vị trí lưu tập tin video sẽ ghi có thể kèm theo intent như một extra có khóa MediaStore.EXTRA_OUTPUT.

Bạn cũng có thể dùng cách tương tự để chụp ảnh: gửi intent đến MediaStore.ACTION_IMAGE_CAPTURE.

```
public class RecordVideoActivity extends ActionBarActivity {
    private static final int VIDEO_CAPTURE = 101;
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mButton = (Button) findViewById(R.id.button_id);
        if (!hasCamera()) mButton.setEnabled(false);
    }

    private boolean hasCamera() {
        return getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA_ANY);
    }

    public void startRecording(View view) {
        File file = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/video.mp4");
        Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
        URI uri = Uri.fromFile(file);
        intent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
        startActivityForResult(intent, VIDEO_CAPTURE);
    }

    @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == VIDEO_CAPTURE) {
            switch (resultCode) {
                case RESULT_OK:
                    Toast.makeText(this, "Video has been saved to:\n" + data.getData(), Toast.LENGTH_LONG).show();
                    break;
                case RESULT_CANCELED:
                    Toast.makeText(this, "Video recording cancelled.", Toast.LENGTH_LONG).show();
                    break;
                default:
                    Toast.makeText(this, "Failed to record video", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

Phải thêm quyền sau vào tập tin manifest:

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

- Chạy ứng dụng trên Andy OS đã kích hoạt camera.
- Nhấn nút trong màn hình ứng dụng để chuyển sang ghi video, nhấn nút đỏ để ghi.
- Khi muốn dừng, nhấn nút đỏ lần nữa. Xem video vừa ghi bằng nút Play giữa màn hình.
- Cuối cùng, nhấn nút Done để quay về màn hình ứng dụng, toast hiển thị sẽ báo vị trí ghi tập tin video.

Camera

Thiết bị Android được trang bị một hay nhiều camera, mặt trước hoặc mặt sau thiết bị, cho phép ứng dụng chụp hình ảnh hoặc thu video. Android cũng cung cấp một tập API giúp tương tác với các camera này.

Để làm việc với camera, cần quyền android.permission.CAMERA và khai báo dùng android.hardware.camera2:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera2" android:required="false" />
```

Thuộc tính android:required="false" cho thấy ứng dụng có thể dùng camera, nhưng camera không phải là yêu cầu bắt buộc khi cài đặt ứng dụng. Nếu true, Google Play sẽ ngăn cài đặt ứng dụng lên thiết bị không có camera.

1. CameraManager

Từ Android 5, lớp CameraManager và các lớp hỗ trợ trong gói android.hardware.camera2 cung cấp camera API cho Android. Bạn lấy thực thể CameraManager bằng cách yêu cầu service hệ thống Context.CAMERA_SERVICE:

```
CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);
```

2. Cấu hình cho camera

a) Camera ID

Do thiết bị có thể có nhiều camera, ứng dụng có thể dùng phương thức getCameraIdList() để lấy danh sách ID của các camera có sẵn:


```
try {
    String[] cameraIds = manager.getCameraIdList();
} catch (CameraAccessException e) {
    // xử lý exception
}
```

b) Tính năng của camera

Dựa trên camera ID, ứng dụng lấy các tính năng của camera thông qua phương thức `getCameraCharacteristics()`, thực thể `CameraCharacteristics` trả về chứa thông tin camera trong các cặp key/value.

```
String[] cameraIds = manager.getCameraIdList();
for (String id : cameraIds) {
    CameraCharacteristics characters = manager.getCameraCharacteristics(id);
    if (CameraCharacteristics.LENS_FACING_FRONT == characters.get(CameraCharacteristics.LENS_FACING)) {
        // tìm thấy camera mặt trước
    }
}
```

Lớp `CameraCharacteristics` cung cấp tập các hằng, chính là key của các tính năng chuẩn của camera, lấy bằng các phương thức `get()`. Một số tính năng đáng chú ý:

Hằng	Mô tả
<code>LENS_FACING</code>	<code>LENS_FACING_FRONT</code> cho camera trước, <code>LENS_FACING_BACK</code> cho camera sau.
<code>SCALER_STREAM_CONFIGURATION_MAP</code>	Trả về <code>StreamConfigurationMap</code> chứa cấu hình các stream hỗ trợ.

3. Mở camera

Camera được mở bằng phương thức `openCamera()` của lớp `CameraManager`. Khi gọi phương thức này, giao diện `StateCallback` chuyển kết quả đến ứng dụng. Các phương thức callback sau phải được cài đặt trong ứng dụng:

Phương thức callback	Mô tả
<code>onOpened()</code>	Được gọi khi camera mở xong.
<code>onError()</code>	Được gọi khi camera có sự cố: <div> <div>ERROR_CAMERA_DEVICE</div> <div>camera có sự cố nghiêm trọng.</div> </div> <div> <div>ERROR_CAMERA_DISABLED</div> <div>camera không mở do chính sách thiết bị.</div> </div> <div> <div>ERROR_CAMERA_IN_USE</div> <div>camera đang dùng bởi ứng dụng khác.</div> </div> <div> <div>ERROR_CAMERA_SERVICE</div> <div>dịch vụ camera gặp sự cố nghiêm trọng.</div> </div> <div> <div>ERROR_MAX_CAMERAS_IN_USE</div> <div>nhiều camera đang mở.</div> </div>
<code>onDisconnect()</code>	Được gọi khi không còn kết nối camera.
<code>onClosed()</code>	Được gọi khi camera đã đóng.

Phương thức `openCamera()` nhận các tham số: ID của camera, thực thể `StateCallback` và thực thể handler dùng để triệu gọi giao diện callback, xử lý các yêu cầu camera không đồng bộ:

```
manager.openCamera(cameraId, new CameraDevice.StateCallback() {
    @Override public void onOpened(CameraDevice camera) {
        // Camera opened
    }
    @Override public void onDisconnected(CameraDevice camera) {
        // Camera disconnected
    }
    @Override public void onError(CameraDevice camera, int error) {
        // Camera error
    }
}, null);
```

4. Chụp ảnh từ camera

Để chụp ảnh (capturing) từ camera, cần một thực thể `Surface`, giống như khi làm việc với video. `Surface` và camera phải được cấu hình trước khi lấy ảnh.

- Lấy kích thước xuất của camera.

Trước hết, bạn cần lấy các kích thước xuất từ camera thông qua đối tượng `CameraCharacteristics`, bằng cách dùng khóa `SCALER_STREAM_CONFIGURATION_MAP`. Trị trả về là một đối tượng thuộc lớp `StreamConfigurationMap`, phương thức `getOutputSizes()` của nó trả về một danh sách các kích thước xuất được hỗ trợ. Thiết lập một trong những kích thước này cho đối tượng `Surface` xem trước (preview) của camera.

```
StreamConfigurationMap streamConfigurationMap =
    characters.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
Size[] outputSizes = streamConfigurationMap.getOutputSizes(SurfaceHolder.class);
surfaceHolder.setFixedSize(outputSizes[0].getWidth(), outputSizes[0].getHeight());
```

- Tạo phiên chụp ảnh (capture session)

Sau khi `Surface` được tạo và thiết lập kích thước xuất, ứng dụng có thể tạo phiên chụp ảnh bằng cách gọi phương thức `createCaptureSession()` của thực thể `CameraDevice`. Phương thức này nhận tham số gồm một danh sách các thực thể `Surface` sẽ được dùng trong phiên chụp ảnh, lớp con của `CameraCaptureSession.StateCallback` và một handler.

```
Surface surface = surfaceHolder.getSurface();
cameraDevice.createCaptureSession(Arrays.asList(surface), new CameraCaptureSession.StateCallback() {
```



```
@Override public void onConfigured(CameraCaptureSession session) { }
@Override public void onConfigureFailed(CameraCaptureSession session) { }
}, null);
```

Kết quả gọi phương thức được chuyển đến ứng dụng thông qua các phương thức được cung cấp bởi lớp con của CameraCaptureSession.StateCallback. Để tạo lớp con của lớp này, bạn phải cài đặt các phương thức:

onConfigured Phiên chụp ảnh được tạo thành công, có thể bắt đầu xử lý yêu cầu chụp.

onConfigurationFailed Phiên chụp ảnh không thể bắt đầu.

- Tạo một yêu cầu chụp (capture request)

Khi bạn đã tạo được phiên chụp ảnh, bạn có thể xử lý nhiều yêu cầu chụp của phiên. Mỗi yêu cầu chụp được tạo bằng phương thức createCaptureRequest() của đối tượng CameraDevice. Với các yêu cầu chụp thường gặp, Android cung cấp các template:

Hằng	Mô tả
TEMPLATE_MANUAL	Ứng dụng trực tiếp điều khiển trực yêu cầu chụp.
TEMPLATE_PREVIEW	Yêu cầu phù hợp cho camera preview.
TEMPLATE_RECORD	Yêu cầu phù hợp cho ghi video.
TEMPLATE_STILL_CAPTURE	Yêu cầu phù hợp cho chụp tiếp.
TEMPLATE_VIDEO_SNAPSHOT	Yêu cầu phù hợp cho chụp tiếp trong lúc ghi video.
TEMPLATE_ZERO_SHUTTER_LAG	Yêu cầu phù hợp cho chụp nhanh, giảm trễ giữa kích hoạt màn trập và lấy ảnh.

Phương thức createCaptureRequest() nhận một kiểu template và trả về một thực thể CaptureRequest.Builder. Mỗi yêu cầu chụp cần một Surface đích, là một trong các Surface được cung cấp khi tạo phiên chụp. Gọi phương thức addTarget() của thực thể CaptureRequest.Builder và truyền cho nó Surface đích.

Bạn có thể cấu hình thêm cho đối tượng builder này. Ví dụ thiết lập auto-focus

```
CaptureRequest.Builder builder = cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
builder.addTarget(surfaceHolder.getSurface());
builder.set(CaptureRequest.CONTROL_AF_MODE, CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_VIDEO);
```

Danh sách yêu cầu chụp được cung cấp như các hằng thuộc lớp CaptureRequest, danh sách được thiết bị hỗ trợ lấy được bằng phương thức getAvailableCaptureRequestKeys() của CameraCharacteristics.

Hằng	Mô tả
COLOR_CORRECTION_MODE	Cách dữ liệu ảnh chuyển đổi từ không gian màu của cảm biến camera thành không gian màu sRGB.
CONTROL_AE_MODE	Chế độ tự động phơi sáng (auto-exposure).
CONTROL_AF_MODE	Chế độ tự động lấy nét (auto-focus).
CONTROL_AWB_MODE	Chế độ tự động cân bằng trắng (auto-whitebalance).
CONTROL_EFFECT_MODE	Áp dụng hiệu ứng màu sắc đặc biệt.
FLASH_MODE	Chế độ dùng đèn flash.
JPEG_QUALITY	Chất lượng nén cho ảnh JPEG.
SCALER_CROP_REGION	Co giãn vùng ảnh được cắt.
SENSOR_SENSITIVITY	Độ nhạy của cảm biến trước khi xử lý.

Khi đã cấu hình đầy đủ yêu cầu chụp, ứng dụng có thể triệu gọi phương thức build() của CaptureRequest.Builder để tạo ra yêu cầu chụp thực sự:

```
CaptureRequest previewRequest = builder.build();
```

- Gửi (submit) yêu cầu chụp

Lớp CameraCaptureSession cung cấp một tập các phương thức cho phép gửi thực thể CameraCaptureRequest đi xử lý:

Hằng	Mô tả
capture()	Gửi yêu cầu chụp một ảnh duy nhất từ camera.
captureBurst()	Gửi danh sách các yêu cầu để chụp như một nhóm ảnh.
setRepeatingRequest()	Gửi yêu cầu cho chụp ảnh lặp đi lặp lại.
setRepeatingBurst()	Gửi danh sách các yêu cầu chụp lặp đi lặp lại nhóm ảnh.

Các phương thức này nhận một thực thể CameraCaptureSession.CaptureCallback để báo với ứng dụng trạng thái của yêu cầu.

```
session.setRepeatingRequest(previewRequest, new CameraCaptureSession.CaptureCallback() {
    @Override public void onCaptureCompleted(CameraCaptureSession session,
        CaptureRequest request, TotalCaptureResult result) {
        super.onCaptureCompleted(session, request, result);
        // capture completed
    }
}, null);
```

Yêu cầu chụp lặp đi lặp lại có thể dừng bằng cách gọi phương thức stopRepeating() của CameraCaptureSession.

Design Patterns và Framework

Android UI Design Patterns

Các ứng dụng di động thành công phải dễ sử dụng. Người dùng không mất nhiều thời gian để tìm hiểu chức năng, và cũng không mất nhiều nỗ lực để sử dụng ứng dụng. Đồng thời, bạn cũng muốn một ứng dụng mà mang lại giá trị hữu ích cho một số khía cạnh của cuộc sống của bạn. Để có thể cân bằng được hai phẩm chất này trong việc thiết kế một ứng dụng trên thiết bị di động, bạn cần tìm hiểu các mẫu thiết kế giao diện người dùng, gọi là các UI design pattern. Các UI design pattern được rút ra từ kinh nghiệm thực tế của các nhà thiết kế, phát triển và quản lý sản phẩm. Chúng thường được sử dụng để giải quyết vấn đề thiết kế chung. Hiểu biết về UI design pattern, giúp thiết kế của bạn trở nên hữu ích, trực quan, dễ tiếp cận với người dùng. Thực tế, ứng dụng Android được "lên khung" trước bằng một phần mềm mockup, storyboard hoặc sketch nào đó. Từng tác vụ trong kịch bản sẽ được thiết kế dựa trên UI design pattern. Đây là vấn đề rất rộng, thiên về UI và UX, nên chúng tôi không thảo luận trong tài liệu này. Bạn có thể tham khảo trong:

- [1] Theresa Neil – **Mobile Design Pattern Gallery, Second Edition** – O'Reilly Media, Inc., 2014. ISBN: 978-1-4493-6363-5
- [2] Greg Nudelman – **Android™ Design Patterns** John Wiley & Sons, Inc., 2013. ISBN: 978-1-118-41755-3

Android Software Design Patterns

Bản thân Android API được thiết kế theo nhiều design pattern. Ví dụ:

- Các lớp cơ sở Activity và Fragment được thiết kế theo design pattern Template Method. Chúng có vòng đời tạo từ chuỗi các phương thức callback. Khi sử dụng, bạn có thể lựa chọn định nghĩa lại các phương thức callback trong chuỗi tạo nên hành vi linh động cho Activity hoặc Fragment đó.
 - View và cây phân cấp của nó là một ví dụ điển hình cho design pattern Composition, kết hợp các đối tượng đơn (View) và đối tượng phức hợp (ViewGroup) vào một cây phân cấp, để xử lý chúng tương tự nhau.
 - BaseAdapter và các lớp con được thiết kế theo design pattern Adapter, tiếp hợp tốt giữa AdapterView với nguồn dữ liệu lấy từ mảng hoặc cơ sở dữ liệu. Các adapter tạo nên cầu nối giữa nguồn dữ liệu và UI.
- Ngoài các design pattern của GoF, khi xây dựng ứng dụng Android, bạn có thể áp dụng nhiều design pattern khác như: MVC, MVP, MVVM, Transfer Object, v.v... Phần này chỉ giới thiệu một số design pattern đặc thù cho Android.

1. View Holder Design Pattern

Thông thường, khi tạo một adapter tùy biến, phương thức getView() được viết như sau:

```
@Override public View getView(int position, View convertView, ViewGroup parent) {
    if (convertView == null) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        convertView = inflater.inflate(R.layout.list_item, parent, false);
    }

    ImageView icon = (ImageView) convertView.findViewById(R.id.icon);
    TextView text = (TextView) convertView.findViewById(R.id.text);

    Item item = mList.get(position); // một mục của ListView
    if (item != null) {
        Bitmap bitmap = getBitmapFromAsset(item.getPath()); // lấy Bitmap từ assets
        if (bitmap != null) {
            icon.setVisibility(View.VISIBLE);
            icon.setImageBitmap(bitmap);
        } else {
            icon.setVisibility(View.GONE);
        }
        text.setText(item.getContent());
    }
    return convertView;
}
```

Khi bạn cuộn ListView chứa nhiều mục, phương thức getView() được gọi nhiều lần, kéo theo việc gọi lại nhiều lần phương thức findViewById() không cần thiết. Ý tưởng là lưu (cache) view để dùng lại.

Ngoài ra, có thể dùng một AsyncTask để nạp ảnh trong một thread chạy nền rồi hiển thị trong UI khi kết thúc nạp ảnh. Bạn cũng có thể hiển thị một progress spinner tại vị trí đặt ảnh trong lúc nó đang nạp.

```
@Override public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;

    if (convertView == null) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        convertView = inflater.inflate(R.layout.list_item, parent, false);
        holder = new ViewHolder();
        holder.icon = (ImageView) convertView.findViewById(R.id.icon);
        holder.text = (TextView) convertView.findViewById(R.id.text);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
}
```

```

Item item = mList.get(position);
if (item != null) {
    holder.position = position;
    new LoadImageTask(position, items.getPath()).execute(holder);
    holder.text.setText(item.getContent());
}
return convertView;
}

private static class LoadImageTask extends AsyncTask<ViewHolder, Void, Bitmap> {
    private int position;
    private String filename;
    private ViewHolder holder;

    public LoadImageTask(int position, String filename) {
        this.position = position;
        this.filename = filename;
    }

    @Override protected Bitmap doInBackground(ViewHolder... params) {
        holder = params[0];
        return getBitmapFromAsset(filename);
    }

    @Override protected void onPostExecute(Bitmap result) {
        super.onPostExecute(result);
        holder.progress.setVisibility(View.GONE);
        if (result != null && holder.position == position) {
            holder.image.setVisibility(View.VISIBLE);
            holder.image.setImageBitmap(result);
        } else {
            holder.image.setVisibility(View.GONE);
        }
    }

    private Bitmap getBitmapFromAsset(String strName) {
        // phương thức công cụ, lấy Bitmap từ assets
    }
}

static class ViewHolder {
    ImageView icon;
    TextView text;
    int position;
    ProgressBar progress;
}

```

2. Delegate Design Pattern

Design pattern Delegate là một mẫu thiết kế trong lập trình hướng đối tượng. Trong mẫu thiết kế này, một đối tượng thay vì thực hiện công việc, nó lại giao (ủy thác) công việc đó cho một đối tượng khác thực hiện thay. Đối tượng giao việc gọi là delegator, đối tượng thực hiện công việc gọi là delegatee.

Chương trình minh họa đọc RSS từ iTunes. Chương trình sẽ gửi yêu cầu lấy về RSS, bản chất là tập tin XML. Để hiển thị nội dung cho người dùng, chương trình phải phân tích XML và lấy những thông tin cần thiết, sau đó hiện lên màn hình.

Yêu cầu thiết kế:

- Tác vụ chiếm nhiều thời gian là phân tích XML được chuyển cho một worker thực hiện, worker này hoạt động trong một thread chạy nền.
- Khi tác vụ chạy nền hoàn thành, cập nhật UI thread bằng:
 - + Dùng phương thức sendMessage() hoặc post() của lớp Handler, xem phần Tác vụ chạy nền.
 - + Dùng delegate, worker sẽ nhận một delegatee và giao việc cập nhật UI thread cho delegatee này.

```

public class MainActivity extends Activity implements ParserListener {
    LinearLayout layout;
    MainActivity activity = this;

    public void setTitle(final String s) {          // callback của ParserListener
        runOnUiThread(new Runnable() {
            public void run() {
                TextView title = new TextView(activity);
                title.setText(s);
                layout.addView(title);
            }
        });
    }
}

```

```

    }
    });
}

@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    setContentView(layout);

    try {
        URL url = new URL("http://ax.phobos.apple.com.edgesuite.net/WebObjects/" +
            "MZStore.woa/wpa/MRSS/topsongs/limit=10/rss.xml");
        new MainActivity.ITunesSAX(url, this);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

class ITunesSAX extends DefaultHandler {
    boolean itemFound = false;
    ParserListener delegatee;
    String element = "";

    public ITunesSAX(final URL url, ParserListener delegatee) {
        this.delegatee = delegatee;
        new Thread(new Runnable() {
            public void run() {
                SAXParserFactory factory = SAXParserFactory.newInstance();
                try {
                    InputStream in = url.openStream();
                    SAXParser saxParser = factory.newSAXParser();
                    saxParser.parse(in, ITunesSAX.this);
                } catch (Throwable t) {
                    t.printStackTrace();
                }
            }
        }).start();
    }

    public void startDocument() throws SAXException { }

    public void endDocument() throws SAXException { }

    public void startElement(String namespaceURI, String sName, String qName, Attributes attrs)
        throws SAXException {
        if (sName.equals("item")) itemFound = true;
        element = "";
    }

    public void endElement(String namespaceURI, String sName, String qName) throws SAXException {
        if (itemFound && sName.equals("title"))
            delegatee.setTitle(element);
    }

    public void characters(char[] buf, int offset, int length) throws SAXException {
        if (length > 0)
            element += new String(buf, offset, length);
    }
}

interface ParserListener {
    public void setTitle(final String s);
}

```

Activity đóng vai trò UI, đồng thời là delegatee. Lớp ITunesSAX đóng vai trò delegator, nhận và giao công việc cho delegatee trong sự kiện endElement, công việc này là hiển thị nội dung RSS (top 10 bài hát từ iTunes) vào TextView.

Phụ lục - Android Studio

[1.2.0]

Source Code Editing

1. Định dạng code thường xuyên khi lập trình là một thói quen tốt. Trong khi lập trình, bạn có thể định dạng code tự động bằng **Ctrl+Alt+L**. Nếu bạn muốn một định dạng code theo phong cách riêng, tùy biến auto-format trong File > Settings... Trong hộp thoại Settings, phần Project Settings > Code Style > Java, chọn các tab và tùy biến.

2. Bạn thường dùng comment (chú thích) để "khóa" tạm một đoạn code hoặc uncomment (mở chú thích) để cho "chạy" đoạn code đã comment. Nếu thêm/xóa vài dòng comment bằng `//`, chọn các dòng và dùng: **Ctrl+/***. Nếu thêm/xóa một khối comment bằng `/**/`, chọn khối và dùng **Ctrl+Shift+/***.

3. Bạn có thể xóa nhanh một dòng bằng **Ctrl+X** hoặc **Ctrl+Y**.

4. Để sao chép một dòng hoặc một khối, đặt con trỏ tại dòng cần sao chép hoặc chọn khối, nhấn **Ctrl+D**.

Có thể nối hai dòng liên tiếp bằng cách đặt con trỏ tại vị trí bất kỳ trong dòng trên rồi nhấn **Ctrl+Shift+J**, hai dòng sẽ nối thành một dòng, các space không cần thiết được loại bỏ.



5. Di chuyển khối hoặc dòng trong Android Studio rất thuận tiện:

- Để di chuyển một dòng/khối, đặt con trỏ tại dòng hoặc chọn khối cần di chuyển, rồi dùng **Alt+Shift+↓** để di chuyển dòng/khối xuống phía dưới, hoặc **Alt+Shift+↑** để di chuyển dòng/khối lên phía trên.
- Để di chuyển một phương thức, đặt con trỏ tại tiêu đề của phương thức cần di chuyển, rồi dùng **Ctrl+Shift+↓** để di chuyển khối phương thức xuống phía dưới, hoặc **Ctrl+Shift+↑** để di chuyển khối phương thức lên phía trên.

6. Kết thúc dòng lệnh bằng **Ctrl+Shift+Enter** là một thao tác thuận tiện trong Android Studio:

- Với dòng lệnh đơn, dấu `;` tự động thêm vào cuối dù con trỏ đang ở giữa dòng lệnh.
- Với các khối `if`, `while`, `for`, ... tự động thêm vào `() { }`

7. Bạn muốn bọc khối code (surround with) đã viết thành thân của các phát biểu `if/else`, `do/while`, `for`, `try/catch`, ... Chọn khối code đó rồi nhấn **Ctrl+Alt+T**.

8. Thanh bên phải, gọi là validation & marker sidebar, có khả năng phân tích code "on-the-fly". Lỗi và cảnh báo hiển thị như các vạch đỏ và vàng trên thanh này, di chuyển mouse đến các vạch để xem nội chi tiết rồi giải quyết nhanh bằng **Alt+Enter** để hiển thị menu tắt chứa các giải pháp gợi ý. Nếu code hoàn chỉnh, icon  đầu thanh sẽ chuyển thành .

9. Tổ hợp phím "vạn năng" **Alt+Enter** đặc biệt hữu dụng:

- Đặt con trỏ tại một tên biến, nhấn **Alt+Enter**, xuất hiện cửa sổ đề nghị một số tùy chọn xử lý biến đó.
- Chọn một tên lớp, nhấn **Alt+Enter**, bạn có thể sinh test case hoặc subclass cho lớp đó.
- Khi nhập một biểu thức chính quy (regular expression), nhấn **Alt+Enter**, chọn Edit RegExp Fragment để nhập và kiểm tra.

10. Đổi tên (rename) lớp, tên phương thức hoặc tên biến hàng loạt bằng chức năng tự động sửa tại các vị trí dùng đến tên đó. Đặt dấu nhắc tại tên cần đổi rồi nhấn **Shift+F6**, nhập tên mới vào cửa sổ popup xuất hiện hoặc chọn một trong các tên được đề nghị, rồi nhấn **Enter**.

11. Để xem nhanh các định danh cần theo dõi, bạn đặt con trỏ lên định danh đó để gạch chân tất cả các vị trí của chúng trong phương thức. Hữu ích nhất là đặt con trỏ lên định danh tại điểm trả về của phương thức để xác định những thay đổi của định danh đó trước khi trả về.

Code Assist

12. Để xem nhanh tài liệu về lớp hoặc phương thức đang quan tâm, đặt con trỏ ngay tên phương thức, nhấn **Ctrl+Q**.

Để xem nhanh các tham số của phương thức đang quan tâm, đặt con trỏ vào giữa cặp `()`, nhấn **Ctrl+P**.

Để xem nhanh định nghĩa của một phương thức, đặt con trỏ ngay tên phương thức, nhấn **Ctrl+Shift+I**.

13. Để tự động import, chọn File > Settings... Trong hộp thoại Settings, phần IDE Settings > Editor > Auto Import, chọn ô Optimize imports on the fly. Chú ý phần Insert imports on paste là Ask, nghĩa là khi chèn code khác vào, những import cần thiết sẽ được hỏi trước khi chèn. Để tổ chức lại import, dùng **Ctrl+Alt+O**.

14. Khi dùng chức năng hoàn chỉnh code (code completion) để chọn một đề nghị trả về boolean, nếu chọn bằng dấu `!`, kết quả là biểu thức có `!` phía trước. Ví dụ: `if (stack.[chọn isEmpty()] bằng !)`, kết quả: `if (!stack.isEmpty())`
 Khi dùng hoàn chỉnh code, nếu bạn muốn thay thế một phương thức hoặc biến đã chọn (nhằm) trước đó, thay vì dùng **Enter** để chấp nhận đề nghị được chọn, hãy nhấn phím **Tab**.

15. Một cách hoàn chỉnh code đặc biệt là hoàn chỉnh code kiểu postfix, được áp dụng ngược từ "phải sang trái". Bạn nhập một biểu thức `expr`, nhập dấu `.` rồi nhấn **Ctrl+J**, sẽ xuất hiện nhiều tùy chọn hữu ích giúp bạn hoàn chỉnh code. Ví dụ: `name.isEmpty()[nhập .][Ctrl+J][chọn else]` cho kết quả `if (!name.isEmpty())`

16. Ngoài chức năng hoàn chỉnh code cơ bản bằng **Ctrl+Space**, Android Studio còn cung cấp thêm chức năng hoàn chỉnh code SmartType giúp tìm các phương thức, biến, tham số phù hợp với ngữ cảnh hiện hành. Bạn chỉ cần nhập ký tự đầu và nhấn **Ctrl+Shift+Space**, Android Studio sẽ đề nghị 5 kết quả phù hợp nhất.

17. Hoàn chỉnh code SmartType có thể dùng sau toán tử **new** để khởi tạo đối tượng với kiểu dự kiến. Sau khi nhập **new**, nhấn **Ctrl+Shift+Space**.

18. Chức năng sinh code tự động đặc biệt hữu dụng, nó cho phép tạo nhanh constructor, getters/setters, override methods, toString, .v.v... Nhấn **Alt+Insert** để truy cập menu sinh code tự động.

Nếu bạn chỉ muốn viết lại (override) các phương thức thừa kế từ lớp cơ sở, nhấn **Ctrl+O**.

Để cài đặt các phương thức của interface mà lớp hiện tại đang cài đặt (hoặc của lớp abstract mà lớp hiện tại đang thừa kế), nhấn **Ctrl+I**.

19. Chức năng Live Templates giúp bạn sinh nhanh các đoạn code thường viết. Danh sách đầy đủ của Live Templates xem trong Settings... > IDE Settings > Live Templates. Chúng đặc biệt hữu dụng, một số thường gặp: **fori** (vòng lặp for với biến đếm), **iter** (vòng lặp foreach), **sout** (System.out.println), **psfi** (private static final int), **St** (String), **ifn** (if null), ... Nếu bạn không nhớ chuỗi viết tắt, nhập một phần rồi nhấn **Ctrl+J**.

20. Chức năng Refactoring không thể thiếu trong IDE hiện đại. Sau khi code, lập trình viên dùng chức năng này để kiến trúc lại code. Trong Android Studio bạn chọn khối code cần kiến trúc lại, nhấn **Ctrl+Alt+Shift+T** để gọi chức năng Refactor This... rồi chọn tùy chọn refactoring theo ý muốn.

Một số tùy chọn refactoring bạn nên gọi trực tiếp từ phím tắt riêng. Ví dụ, với tùy chọn extract, vào Refactor > Extract để xem các phím tắt. Chúng cho phép đơn giản các phát biểu phức tạp để dễ xem code, ví dụ **Ctrl+Alt+V** để trích xuất biến.

Navigation


21. Android Studio hỗ trợ tìm kiếm rất mạnh, bạn nhấn nhanh hai lần phím **Shift**, cửa sổ tìm kiếm "khắp mọi nơi" (Search Everywhere) hiển thị cho phép bạn nhập và tìm kiếm mọi thứ trong Android Studio. Từ khóa nhập vào có thể theo kiểu "camel". Ví dụ, nhập "GeDe" để tìm "GestureDetector", nhập "oCr" để tìm "onCreate". Cách nhập từ khóa theo kiểu "camel" này cũng áp dụng nhiều nơi, ví dụ cho chức năng hoàn chỉnh code.

22. Di chuyển nhanh trong Android Studio:

- Tìm đến lớp quan tâm: **Ctrl+N** rồi nhập một phần tên lớp.
- Tìm đến tập tin quan tâm: **Ctrl+Shift+N** rồi nhập một phần tên tập tin.

23. **Ctrl+Click** vào tab của tập tin đang biên soạn bạn sẽ mở cửa sổ nhỏ File Path. File Path trình bày đường dẫn đến tập tin đó theo thứ tự ngược từ dưới lên. Khi bạn chọn một phần của File Path, vị trí tương ứng sẽ mở trong Explorer.

24. Navigation bar là thanh ngay dưới toolbar, còn gọi là breadcrumb trail, cho phép nhận biết đường dẫn đến tập tin bạn đang làm việc. Bạn có thể đóng Navigation bar, từ menu View, để thêm không gian làm việc. Khi cần đến, nhấn **Alt+Home**, dùng các phím mũi tên để di chuyển trên breadcrumb trail. Chú ý rằng từng phần của breadcrumb trail là một danh sách thả xuống cho phép điều hướng nhanh đến các phần trong project (tập tin, thư mục).

25. Bạn đang trong cửa sổ biên soạn và muốn nhảy nhanh đến tập tin chứa code đang biên soạn trong tool window Project. Hãy click vào icon "Scroll from Source"  trên đầu tool window Project.

26. Bạn muốn nhảy nhanh đến nơi định nghĩa phương thức (source), có hai cách:

- Nếu con trỏ đang ngay tên phương thức, nhấn **F4**.
- Nhấn giữ phím **Ctrl** để tên phương thức chuyển thành link, click lên link.

Để nhảy nhanh đến nơi khai báo, dùng **Ctrl+B**. Để chuyển đến nơi cài đặt một phương thức abstract, đặt con trỏ lên tên của nó trong khai báo rồi nhấn **Ctrl+Alt+B**.

27. Bạn nhảy nhanh từ phương thức này sang phương thức sau/trước nó, bằng **Alt+↓ / Alt+↑**.


28. Bạn muốn chuyển nhanh qua lại giữa cửa sổ công cụ (tool window) và cửa sổ biên soạn (editor):

- Khi đang trong tool window, nhấn **Esc** để chuyển về editor.
- Nếu nhấn **Shift+Esc**, sau khi chuyển về editor, tool window sẽ đóng.
- Khi đang trong editor, nhấn **F12** để chuyển đến tool window được sử dụng gần nhất.

29. Dùng **Ctrl+Tab** để chuyển qua lại giữa các tập tin và các tool window đang mở. Khi cửa sổ Switcher xuất hiện, nhấn giữ phím **Ctrl**, rồi dùng các phím: mũi tên, **Tab**, **Shift+Tab**, **Alt** để di chuyển giữa các mục. Dùng **Delete** hoặc **BackSpace** để đóng tập tin đang biên soạn hoặc tool window được chọn.


30. Bạn đã viết một lớp, phương thức hoặc biến, được dùng trong project. Bây giờ, bạn muốn biết chúng được dùng ở đâu trong project và bạn muốn nhảy nhanh đến những vị trí dùng chúng. Hãy đặt con trỏ lên tên của chúng và nhấn **Ctrl+Alt+F7**, chức năng Show Usages sẽ giúp bạn.

Workspace

31. Khi bạn đang mở tool window Project (**Alt+1**), chú ý có nhiều chế độ quản lý project, liệt kê trong danh sách thả xuống ngay góc trên trái, trong đó ba chế độ quan trọng nhất là:  Android (hiển thị mặc định),  Project (lưu trữ thật của project) và  Packages.

Ví dụ, bạn cần đưa một ảnh vào drawable, chuyển chế độ của tool Window Project thành Project, giã nhánh app/src/main/res/drawable, kéo thả ảnh vào thư mục drawable.

32. Android Studio có chức năng Smart Rendering, khi gặp vấn đề về giao diện đang thiết kế, Android Studio hiển thị các link cho phép sửa nhanh lỗi hiển thị. Ví dụ, bạn thêm Button đến layout mà quên chỉ định các thuộc tính width và height, Android sẽ hiển thị link "Automatically add all missing attributes" để bạn click vào và sửa lỗi.

33. Biên xung quanh Android Studio gọi là *tool window bar*, chứa nhiều tool window. Chúng tiện dụng nên thường xuyên được gọi đến bằng cách click lên chúng hoặc nhấn **Alt+n** với n là số trước tên tool window; ví dụ, 1: Project (**Alt+1**). Dùng phím tắt này một lần nữa để đóng tool window vừa gọi. Click vào icon  góc dưới trái để đóng tool window bar, mở rộng thêm không gian làm việc. Khi đưa mouse lên icon này, một menu xuất hiện cho phép bạn chọn nhanh tool window cần thiết.

34. Khi cần sao chép, di chuyển một khối code lớn, hoặc đơn giản để code thoáng hơn, bạn hãy dùng chức năng *code folding*. Click dấu [-] để thu khối code lại, như vậy khối code chỉ nằm trên một dòng, dễ dàng được chọn để sao chép.

Dùng **Ctrl+-** để thu (collapse) thành viên của lớp hiện tại hoặc **Ctrl+Shift+-** để thu tất cả (collapse all). **Ctrl++** (Numpad) để mở rộng, và **Ctrl+Shift++** để mở rộng tất cả.

Một đặc tính thú vị là lớp nội vô danh đơn giản (có một phương thức) khi thu lại có hình thức giống biểu thức lambda.

35. Bạn đang code trong một cửa sổ và cần mở rộng tối đa kích thước cửa sổ để có thêm không gian làm việc. Bạn có thể double-click lên tiêu đề của cửa sổ hiện hành. Tuy nhiên, nhanh nhất là dùng **Shift+Esc**.

36. Bạn đặt con trỏ tại tên lớp và nhấn **Ctrl+H**. Tool window Type Hierarchy sẽ mở, cung cấp cho bạn thông tin hữu ích về lớp đó trong cây phân cấp lớp.

37. Bạn đặt con trỏ tại tên phương thức và nhấn **Ctrl+Alt+H**. Tool window Hierarchy Callers sẽ mở, cung cấp "đường dẫn" có thể từ nơi khai báo phương thức đến nơi triệu gọi nó.

38. Để đóng tất cả các tab, ngoại trừ tab đang quan tâm, bạn nhấn giữ phím **Alt** và click vào dấu × của tab đang quan tâm. Để đóng một tab, click phím giữa mouse lên tab.

39. Để di chuyển qua lại giữa các tab, nhấn **Alt+→** hoặc **Alt+←**.

40. Bạn có thể đánh dấu dòng code bằng Bookmark (**F11**) hoặc Bookmark Mnemonic (**Ctrl+F11**). Dùng **Shift+F11** hiển thị cửa sổ Bookmarks, để xem và xóa đánh dấu nếu cần (nhấn dấu – góc trên trái).

Time Savers

41. Bạn nên lên kế hoạch tự ép sử dụng các phím tắt để thuộc dần, cố gắng thay thế thói quen cũ. Sau một thời gian áp dụng phím tắt, tốc độ code của bạn sẽ tăng lên đáng kể. Danh sách các phím tắt mặc định của Android Studio cũng là danh sách các phím tắt của IntelliJ IDEA, có thể xem bằng cách vào Help > Default Keymap Reference.

42. Nhấn **Alt+Shift+F10**, bạn có thể truy cập nhanh đến menu Run/Debug mà không cần dùng mouse.

43. Bạn có thể triệu gọi mục menu hoặc hành động bất kỳ trong Android Studio bằng chức năng "tìm action" (command lookup), nhấn **Ctrl+Shift+A** và nhập action cần tìm. Nếu action đó có phím tắt, phím tắt sẽ hiển thị ngay bên cạnh tên action tìm thấy.

44. Với những phương thức cần viết tiếp, cần xem lại, cần tối ưu sau, ... bạn tạo một chú thích với **TODO**, nhập **/**[Enter]** rồi nhập **TODO <nội dung>**. Android Studio tạo ra một vạch xanh tương ứng trên thanh bên phải để nhắc. Truy cập các TODO này bằng tool window TODO, nằm góc dưới-trái.

45. Tô chọn (highlight) mọi thứ trong Android Studio, nó cung cấp nhiều thông tin hữu ích:

- Tô chọn return và throw trong phương thức cũng tô chọn tất cả điểm thoát của phương thức.
- Tô chọn extends và implements trong định nghĩa lớp cũng tô chọn các phương thức nó viết lại (override) hoặc cài đặt.
- Tô chọn import cũng tô chọn nơi dùng đến nó.

46. Tìm kiếm nhanh được tích hợp sẵn trong tất cả các tool window hiển thị dạng cây. Trong khi đang chọn tool window, gõ trực tiếp từ cần tìm vào bàn phím là bạn định vị được ngay mục cần tìm.

47. Bạn có thể chọn nhanh một phương thức bằng cách đặt con trỏ lên tên phương thức, nhấn **Ctrl+W** hai lần. Nếu nhấn thêm, vùng chọn sẽ mở rộng dần.

48. Bạn có thể nhìn thấy lịch sử thay đổi (history) của một tập tin, bằng cách gọi Local History > Show History trong menu tắt của tập tin đó. Nó cho phép bạn duyệt qua các phiên bản thay đổi của tập tin, xem khác biệt giữa các phiên bản và khôi phục lại một phiên bản bất kỳ.

49. Android Studio cho phép chế độ nhiều con trỏ. Nhấn giữ **Alt+Shift**, click mouse tại các vị trí muốn đặt con trỏ để đặt chúng. Như vậy, bạn có thể nhập cùng lúc tại các con trỏ. Nhấn **Esc** để loại các con trỏ bổ sung.

50. Tăng bộ nhớ heap của Android Studio, mở tập tin <Android Studio>/bin/studio64.exe.vmoptions hoặc studio.exe.vmoptions tùy phiên bản bạn dùng, tìm hai dòng: -Xms128m và -Xmx750m và tăng các trị này một cách hợp lý (tùy dung lượng RAM), ví dụ, -Xms256m và -Xmx1024m. Bạn sẽ nhận thấy Android Studio khởi động nhanh hơn nhiều.

Phụ lục - Gradle

Hệ thống build tự động của Android Studio dựa trên Gradle tích hợp vào Android Studio. Gradle cho phép các project được xây dựng, cấu hình, quản lý thông qua một tập hợp các tập tin cấu hình build. Các tập tin này định nghĩa cách project được build, các dependency cần đưa vào project, những gì là kết quả cuối của quá trình build, .v.v...

Các plugin Android giúp Gradle chạy độc lập, nghĩa là bạn có thể build ứng dụng Android từ Android Studio hay từ giao diện dòng lệnh trên máy của bạn nếu không có cài đặt Android Studio.

Gradle sử dụng DSL (Declarative Domain-Specific Language) để mô tả và cấu hình thông qua cú pháp Groovy thay vì dùng XML như Ant hay Maven. Groovy là một ngôn ngữ động được dùng để định nghĩa tùy biến các thao tác build và tương tác với các thành phần cụ thể trong Android.

Bạn có thể xem tài liệu hướng dẫn tại: <http://tools.android.com/tech-docs/new-build-system/user-guide>.

Gradle cung cấp một số tính năng mạnh mẽ cho việc build các project Android:

- Gradle có một tập định nghĩa trước các thiết lập cấu hình mặc định hợp lý, được tự động sinh ra khi tạo project mới. Các cấu hình mặc định này sẽ được sử dụng, trừ phi bạn phủ quyết chúng bằng cách thiết lập trong các tập tin build. Vì vậy, bạn chỉ cần cấu hình tối thiểu là có thể build. Những thay đổi trong các tập tin build chỉ cần thiết khi cấu hình mặc định không đủ đáp ứng yêu cầu build của bạn.

- Mỗi project Android Studio liên kết với một tập tin AndroidManifest.xml chứa cấu hình chi tiết của ứng dụng. Một số mục chỉ định trong tập tin build của Gradle sẽ tự động sinh thành các section XML tương ứng trong tập tin manifest khi project được build. Khả năng này cũng bổ sung cho việc tạo các biến thể build.

- Gradle hỗ trợ APK signing trong đoạn signingConfigs { } và dùng ProGuard khi chuẩn bị phát hành ứng dụng Android.

Các tập tin build của Gradle

Một project trong Android Studio bao gồm module hiện hành (app) và một hay nhiều module phụ thuộc. Một module là một phần của ứng dụng có thể build, test, hoặc debug một cách độc lập. Có nhiều loại module:

- Android Module (module ứng dụng): bao gồm src/main, AndroidManifest.xml và build.gradle. Nói chung tất cả những gì bạn cần để sinh gói APK. Dùng plugin com.android.application

- Library Module (module thư viện): là Android Module chuẩn, nhưng khi tạo module dùng kiểu module là Android Library. Hệ thống build sẽ sinh gói AAR (Android Archive) để tái sử dụng. Dùng plugin com.android.library

- App Engine Module: các module cung cấp bởi bên thứ ba, như App Engine Backend with Google Cloud Messaging (GCM), App Engine Java Endpoints Module, App Engine Java Servlet Module.

- Java Library Module (module thư viện Java): thư viện tái sử dụng. Hệ thống build sẽ sinh gói JAR cho module này.

Một project chứa:

- Tập tin build Gradle cấp cao nhất (top-level): build.gradle (Project: <project name>) có nội dung:

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

- Tập tin build Gradle cấp module: build.gradle (Module: <module name>). Một project Android Studio được tạo thành từ một hoặc nhiều module, mỗi module yêu cầu tập tin build Gradle riêng, có nội dung:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "21.1.2"
    defaultConfig {
        // manifest entries
        applicationId "com.twe.samples.storageproject"
        minSdkVersion 11
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.0.0'
}
```

Dependency là các thư viện, tập tin JAR, project, module mà module hiện hành phụ thuộc (depends) vào chúng để biên dịch và chạy. Các dependency khai báo trong đoạn, bao gồm:

- Module dependency, danh sách các module mà module hiện hành phụ thuộc. Khi build module hiện hành, hệ thống build sẽ tập hợp các module phụ thuộc lại.
- Remote dependency, tham chiếu đến các dependency trên máy chủ từ xa, thường gọi là repository. Các dependency tự động tải về từ repository và tham gia quá trình build, bạn không cần thực hiện thủ công. Các repository được hỗ trợ bao gồm jcenter, Maven Central hoặc Ivy.
- Local dependency, tham chiếu đến các dependency có trên hệ thống tập tin cục bộ.

Trong Android Studio, bạn thêm dependency bằng cách: trong menu tắt của app chọn Open Module Settings (F4). Trong cửa sổ Project Structure, chọn tab Dependencies. Click dấu (+) và chọn loại dependency cần thêm bằng cách chỉ đến chúng. Cuối cùng, kiểm tra đoạn dependencies { }.

Các tác vụ build

Mỗi project trong Android Studio cũng chứa một tool cho phép các tác vụ của Gradle được triệu gọi từ dòng lệnh. Tuy nhiên, Android Studio có tool window Gradle (góc trên phải) chứa một danh sách các tác vụ Gradle phân thành nhóm (android, build, install, other, verification) double-click lên tác vụ để chạy tác vụ đó.

Các task cấp cao nhất (top-level) là các task nếu chạy nghĩa là chạy tất cả các task phụ thuộc. Chúng bao gồm:

- build.assemble: build đầu ra của project.
- verification.check: chạy kiểm tra và kiểm thử.
- build.build: chạy cả assemble và check.
- build.clean: dọn dẹp các tập tin sinh ra trong quá trình build.

Build variants

Gradle cung cấp các biến thể build (build variants) hỗ trợ cho các project Android Studio. Điều này cho phép nhiều biến thể của một ứng dụng được build từ một project duy nhất. Android chạy trên nhiều thiết bị khác nhau, có kiểu vi xử lý và màn hình khác nhau. Nếu bạn chọn nhiều thiết bị mục tiêu, bạn sẽ cần các biến thể build tương ứng.

Build variants là sự kết hợp giữ productFlavors và buildTypes. productFlavor là phiên bản của sản phẩm mà quá trình build sẽ tạo ra như là bản full hay bản demo, bản free hay bản paid, bản phone hay bản tablet. buildType thể hiện phiên bản đóng gói mà quá trình build sẽ sinh ra trong từng gói ứng dụng như là debug hay release. Hệ thống build sẽ sinh ra các gói APK theo từng cặp, kết hợp giữa productFlavor và buildType.

Để thêm productFlavor ta cần phải:

- Định nghĩa productFlavor trong tập tin build cấp module: build.gradle.

```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
    debug {
        debuggable true
    }
}
productFlavors {
    paid {
        applicationId "com.twe.gradle.samples.paid"
        versionName "1.0-paid"
    }
    free {
        applicationId "com.twe.gradle.samples.free"
        versionName "1.0-free"
    }
}
```

- Tạo các thư mục nguồn bổ sung cho từng productFlavor.

```
app
├── src
│   ├── main                // dùng chung cho các biến thể
│   │   ├── java
│   │   └── res
│   ├── free
│   │   ├── java
│   │   └── res
│   └── paid
│       ├── java
│       └── res
```

Các build variant tạo được: freeDebug, freeRelease, paidDebug, paidRelease.

Tài liệu tham khảo

(Theo năm xuất bản)

- [1] Onur Cinar – **Android Quick APIs Reference** – Apress, 2015. ISBN: 978-1-4842-0523-5
- [2] Neil Smyth – **Android Studio Development Essentials** – Amazon Digital Services, Inc., 2015. ISBN: 1-500-61386-X
- [3] Marko Gargenta, Masumi Nakamura – **Learning Android, Second Edition** – O'Reilly Media, Inc., 2014. ISBN: 978-1-449-31923-6
- [4] Erik Hellman – **Android Programming, Pushing the Limits** – John Wiley & Sons Ltd, 2014. ISBN: 978-1-118-71730-1
- [5] Bill Phillips & Brian Hardy – **Android Programming – The Big Nerd Ranch Guide** – Big Nerd Ranch, Inc., 2013. ISBN: 0-321-80433-3
- [6] Corinne Hoisington – **Android Boot Camp for Developers** – Cengage Learning, 2013. ISBN: 978-1-133-59720-9
- [7] Mark L. Murphy – **The Busy Coder's Guide to Android Development** – CommonsWare, LLC., 2013. ISBN: 978-0-9816780-0-9
- [8] Zigurd Mednieks, Laird Dornin, G. Blake Meike, and Masumi Nakamura – **Programming Android, Second Edition** – O'Reilly Media, Inc., 2012. ISBN: 978-1-449-31664-8
- [9] Satya Komatineni, Dave MacLean – **Pro Android 4** – Apress, 2012. ISBN: 978-1-4302-3931-4