Giới thiệu

Tài liệu **Lập trình Android cơ bản** là bản dịch chọn lọc nội dung từ trang ***developer.android.com*** - trang Web do chính Google xây dựng và chia sẻ miễn phí cho các lập trình viên Android. Với mong muốn để sinh viên học tập được tốt nhất môn này, bộ phận Bản quyền và Xuất bản Đại học FPT đã thực hiện chuyển ngữ tài liệu sang tiếng Việt và cung cấp miễn phí cho sinh viên FPT Polytechnic dưới dạng file PDF lưu trên trang Web ***lms.poly.edu.vn***.

Vì đây là lần chuyển ngữ đầu tiên tài liệu **Lập trình Android cơ bản** này, nên dù đã rất cố gắng trong các khâu biên tập, hiệu đính, chế bản, song thiếu sót là điều không thể tránh khỏi. Bộ phận Bản quyền và Xuất bản rất mong nhận được ý kiến đóng góp của quý độc giả gần xa để hoàn thiện bản dịch trong những lần tới.

Mọi ý kiến đóng góp xin vui lòng gửi về:
Bộ phận Bản quyền và Xuất bản - Đại học FPT
Địa chỉ: Tầng 2, nhà F, tòa nhà Việt Úc (VAS), khu đô thị Mỹ Đình I, Từ Liêm, Hà Nội.
Email: publishing@fpt.edu.vn

*FPT Polytechnic tin tưởng rằng, tiêu chí **"Thực học - Thực nghiệp"** sẽ là mục đích cao đẹp nhất của một đơn vị đào tạo. Chúng ta hãy cùng chia sẻ mục đích này để có được một môi trường đào tạo tốt, xây dựng được những giá trị thiết thực, giúp cho mỗi sinh viên vững vàng trên con đường sự nghiệp của mình.*

Tháng 04 năm 2014

Mục lục

1. Android, nền tảng di động phổ biến nhất thế giới

Android là nền tảng của hàng trăm triệu thiết bị di động tại hơn 190 quốc gia trên thế giới. Trong các nền tảng di động (mobile platform), đây là nền tảng được cài đặt nhiều nhất và có tốc độ phát triển rất nhanh - mỗi ngày, hàng triệu người dùng bật thiết bị Android của họ lên lần đầu tiên rồi bắt đầu tìm kiếm các ứng dụng, trò chơi cùng những nội dung số khác.

Android cung cấp cho bạn một nền tảng tốt nhất toàn cầu nhằm tạo ra các ứng dụng và trò chơi cho người dùng Android ở mọi nơi, cùng với đó là một thị trường mở để phân phối chúng ngay tức thì.

Sự tăng trưởng của Android thể hiện qua số lượng thiết bị đã được kích hoạt

Quan hệ đối tác và nền tảng cài đặt toàn cầu

Từ việc xây dựng trên sự đóng góp của cộng đồng mã nguồn mở Linux (open-source Linux community) cùng hơn 300 phần cứng, phần mềm và các nhà mạng đối tác, Android đã mau chóng trở thành hệ điều hành di động phát triển nhanh nhất.

Mỗi ngày có hơn 1 triệu thiết bị Android mới được kích hoạt trên toàn thế giới.

Tính mở của Android khiến người tiêu dùng yêu thích nền tảng này và các nhà phát triển ứng dụng cũng vậy, điều đó đã thúc đẩy sự tăng trưởng mạnh trong lĩnh vực tiêu thụ ứng dụng. Người dùng Android tải hơn 1,5 tỷ ứng dụng và trò chơi từ Google Play mỗi tháng.

Cùng với các đối tác, Android liên tục mở rộng ranh giới giữa phần cứng với phần mềm, nhằm mang lại những tính năng mới cho người dùng và các nhà phát triển. Đối với nhà phát triển, sự đổi mới của Android cho phép bạn xây dựng những ứng dụng mạnh mẽ, khác biệt, dùng công nghệ di động mới nhất.

Framework[1] phát triển mạnh mẽ

Rất dễ để tối ưu hóa một hệ nhị phân cho điện thoại, máy tính bảng (tablet) và các thiết bị khác.

Android cung cấp mọi thứ bạn cần để xây dựng và trải nghiệm ứng dụng ở mức tốt nhất. Nó cũng cung cấp một mô hình ứng dụng đơn, cho phép bạn triển khai rộng rãi ứng dụng của mình tới hàng trăm triệu người dùng trên một loạt thiết bị - từ điện thoại cho tới máy tính bảng và còn hơn thế nữa.

Android cũng đưa ra những công cụ tạo ứng dụng có giao diện đẹp mắt và tận dụng lợi thế từ khả năng phần cứng có sẵn trên từng thiết bị. Nó cũng tự động thích nghi với giao diện người dùng (user interface - UI) để đạt được sự tối ưu nhất trên từng thiết bị, trong khi vẫn đưa ra nhiều điều khiển (control) như bạn muốn, thông qua giao diện người dùng trên các loại thiết bị khác nhau.

Ví dụ, bạn có thể tạo một ứng dụng hệ nhị phân sao cho ứng dụng này được tối ưu hóa cho cả điện thoại lẫn các dạng máy tính bảng. Bạn có thể khai báo giao diện người dùng của mình một cách gọn nhẹ trong tập các tài nguyên XML, một tập dành cho những thành phần chung đối với tất cả các dạng, trong khi những tập khác dùng cho việc tối ưu hóa theo đặc trưng của điện thoại hoặc máy tính bảng. Khi chạy, Android áp dụng đúng tập tài nguyên dựa trên kích thước màn hình, mật độ, vị trí,…

Nhằm giúp bạn phát triển hiệu quả, các công cụ phát triển Android (Android Development Tools - ADT) cung cấp cho bạn một môi trường phát triển tích hợp (Integrated Development Environment - IDE) đầy đủ cho Java với những tính năng tiên tiến để phát triển, gỡ lỗi (debugging) và đóng gói (packing) các ứng dụng Android. Sử dụng IDE, bạn có thể phát triển ứng dụng trên mọi thiết bị Android có sẵn, hoặc tạo các thiết bị ảo (virtual device) giả lập bất kỳ cấu hình phần cứng nào.

Có 1,5 tỷ lượt tải về mỗi tháng và con số này vẫn tiếp tục tăng. Hãy đưa ứng dụng của bạn ra trước mặt hàng triệu người dùng theo quy mô của Google.

Thị trường mở để phân phối các ứng dụng của bạn

Google Play là thị trường hàng đầu cho việc bán và phân phối các ứng dụng Android. Khi phát hành một ứng dụng trên Google Play, bạn đã chạm được vào nền tảng khổng lồ được cài đặt của Android.

[1] Tập các tài nguyên tạo nên khung nền cho việc phát triển hệ thống.

Là một thị trường mở, Google Play cho phép bạn kiểm soát cách thức bán các sản phẩm của mình. Bạn có thể phát hành ở mọi thời điểm bạn muốn, với tần suất tùy ý và tới những khách hàng bạn quan tâm. Bạn cũng có thể phân phối rộng khắp trên mọi thị trường và thiết bị hay tập trung vào những phân khúc, thiết bị cụ thể, hoặc theo phạm vi của khả năng phần cứng.

Bạn có thể thu lợi nhuận theo cách hợp lý nhất cho doanh nghiệp của mình - mất phí hoặc miễn phí, với các sản phẩm nhúng trong ứng dụng (in-app) hoặc đăng ký theo dõi (subscription) - đảm bảo đạt được sự cam kết và thu nhập tốt nhất. Bạn cũng có thể kiểm soát hoàn toàn giá cả cho ứng dụng của mình và các sản phẩm nhúng trong ứng dụng, đồng thời có thể thiết lập hoặc thay đổi giá với bất cứ loại tiền tệ được hỗ trợ vào mọi thời điểm.

Ngoài sự phát triển của nền tảng khách hàng, Google Play giúp bạn xây dựng tầm nhìn và cam kết thông qua ứng dụng và thương hiệu của bạn. Khi ứng dụng của bạn dần trở nên phổ biến, Google Play xếp cho chúng vị trí cao hơn trong biểu đồ xếp hạng "đứng đầu" mỗi tuần, tiến hành xếp hạng và đảm bảo những vị trí tốt nhất cho ứng dụng trong kho.

Được cài đặt sẵn trong hàng trăm triệu thiết bị Android trên thế giới, Google Play có thể là một phương tiện phát triển cho doanh nghiệp của bạn.

KHỞI ĐỘNG

Tất cả những gì bạn cần để bắt đầu phát triển ứng dụng cho Android đều có ở đây, trên trang developer.android.com. Bạn sẽ tìm thấy mọi thứ trên trang Web này, từ những SDK (bộ công cụ phát triển phần mềm - Software Developement Kit), tài liệu hướng dẫn API, chỉ dẫn thiết kế, thông tin về chiều xoay (landscape) của thiết bị hiện tại cho tới cách để bạn có thể phân phối và thu lợi nhuận từ các ứng dụng.

Các ứng dụng đều được xây dựng theo những cách khác nhau, nhưng chúng tôi đã cấu trúc những thông tin cần thiết giúp bạn xây dựng một ứng dụng thành ba mục chính dưới đây, đại diện cho quy trình phát triển ứng dụng chung:

Thiết kế

Trước khi viết một dòng mã, bạn cần thiết kế giao diện người dùng và làm cho nó phù hợp với trải nghiệm người dùng trên Android. Mặc dù bạn có thể biết người dùng sẽ làm gì với ứng dụng của mình, song bạn vẫn nên dừng lại để tập trung vào cách người dùng sẽ *tương tác với* nó. Thiết kế của bạn nên có style đẹp, đơn giản, mạnh mẽ và được điều chỉnh hướng tới trải nghiệm trên Android.

Vì vậy, bất kể bạn là một cửa hàng nhỏ chỉ có một người hay là một nhóm lớn nhiều người thì cũng nên nghiên cứu các chỉ dẫn "Design" ("Thiết kế") trước tiên.

Phát triển

Một khi thiết kế của bạn đã hoàn thiện, tất cả những gì bạn cần là các công cụ để biến ý tưởng thiết kế ứng dụng đó thành hiện thực. Framework của Android cung cấp cho bạn các giao diện lập trình ứng dụng (Application Programming Interface - API) để xây dựng những ứng dụng có thể khai thác toàn bộ lợi thế của phần cứng thiết bị, thiết bị phụ kiện kết nối (connected accessory device), mạng Internet, các tính năng phần mềm và nhiều hơn nữa. Với sức mạnh của Android, sức mạnh mà ứng dụng của bạn có thể đem lại là không giới hạn.

Mọi kiến thức bạn cần để học về framework và các công cụ phát triển ứng dụng đều có trong mục (Develop) ("Phát triển") của tài liệu hướng dẫn.

Phân phối

Hiện giờ, ứng dụng của bạn đã hoàn tất. Chúng ta đã xây dựng ứng dụng đó để hỗ trợ nhiều kích cỡ và mật độ màn hình, đồng thời đã kiểm thử (test) nó trên trình giả lập (emulator) Android cũng như trên các thiết bị thật. Bạn đã sẵn sàng để đem ứng dụng của mình đi phân phối.

Bước tiếp theo phụ thuộc vào nhiều yếu tố, chẳng hạn như chiến lược tiền tệ và loại thiết bị hỗ trợ ứng dụng của bạn. Mọi thông tin bạn cần để bắt đầu quá trình này đều có trong mục (Distribute) ("Phân phối").

2. Intent và bộ lọc Intent

Bộ ba trong các thành phần chính (core component) của một ứng dụng - activity (một thành phần của ứng dụng cung cấp giao diện để người dùng tương tác nhằm thực hiện một công việc nào đó), service (luồng dịch vụ chạy ngầm trong Android) và broadcast receiver (trình thu nhận các thông tin bên ngoài gửi tới) - được kích hoạt thông qua bản tin (message), gọi là các *intent*. Việc gửi và nhận intent là cơ sở cho việc liên kết khi chạy về sau này giữa những thành phần trong cùng một hoặc các ứng dụng khác nhau. Bản thân đối tượng `Intent` là một cấu trúc dữ liệu thụ động chứa mô tả trừu tượng về một activity được thực hiện - hoặc, trong trường hợp của broadcast thì đó thường là mô tả về điều gì đó đã xảy ra và được công bố. Có nhiều cơ chế riêng biệt để gửi Intent cho mỗi loại thành phần của hệ thống:

- Một đối tượng Intent được truyền tới `Context.startActivity()` hoặc `Activity.startActivityForResult()` nhằm khởi động một activity hoặc lấy activity hiện có để làm việc mới. (Intent cũng có thể được truyền tới `Activity.setResult()` để trả về thông tin cho activity gọi `startActivityForResult()`).
- Một đối tượng Intent được truyền tới `Context.startService()` để khởi tạo một service hoặc gửi đi những chỉ lệnh mới tới một service đang chạy. Tương tự, một intent có thể được truyền tới `Context.bindService()` để thiết lập một kết nối giữa thành phần gọi và service đích. Intent có thể tùy ý khởi tạo service nếu như service chưa chạy sẵn.
- Các đối tượng Intent truyền tới bất cứ phương thức broadcast nào (chẳng hạn như `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()` hay `Context.sendStickyBroadcast()`) được gửi tới tất cả các broadcast receiver cùng hệ thống. Nhiều loại tin broadcast như vậy bắt nguồn từ mã hệ thống (system code).

Trong mỗi trường hợp, hệ thống Android sẽ tìm activity, service hoặc tập các broadcast receiver thích hợp để phản hồi intent và khởi tạo chúng, nếu cần. Không có sự chồng chéo bên trong các hệ thống nhắn tin này: Intent broadcast chỉ được gửi tới broadcast receiver chứ không bao giờ được gửi tới activity hay service. Một intent truyền tới `startactivity()` chỉ được gửi tới một activity chứ không bao giờ được gửi tới một service hay broadcast receiver, và cứ như vậy.

Tài liệu này bắt đầu với một mô tả về đối tượng Intent. Sau đó, tài liệu đi vào mô tả những quy tắc mà Android sử dụng để kết nối các intent với các thành phần - cách phân tích đâu là những thành phần nên nhận thông điệp intent. Đối với các intent không chỉ rõ một thành phần đích một cách tường minh, quá trình này liên quan đến việc kiểm thử đối tượng intent dựa vào *bộ lọc intent (intent filter)* liên kết với những thành phần đích có khả năng.

Các đối tượng Intent

Một đối tượng `Intent` là một tập thông tin. Đối tượng này chứa thông tin về đặc điểm của thành phần nhận intent (như action được thực hiện và dữ liệu để thực hiện action đó), cùng với đó là thông tin về đặc điểm của hệ thống Android (như danh mục của thành phần xử lý intent và các lệnh khởi chạy một activity đích). Phần lớn đối tượng `Intent` có thể bao gồm:

Tên thành phần

Tên của thành phần xử lý intent. Trường này là một đối tượng `ComponentName` - một sự kết hợp tên lớp (class) đầy đủ của thành phần đích (chẳng hạn như "`com.example.project.app.FreneticActivity`") với tên gói (package) đặt trong file kê khai (manifest file) của ứng dụng, nơi chứa các thành phần (ví dụ như "`com.example.project`"). Phần thông tin tên gói trong ComponentName và tên gói được thiết lập trong file kê khai không nhất thiết phải trùng nhau.

Tên thành phần là không bắt buộc. Nếu được thiết lập, đối tượng Intent được gửi tới một thể hiện (instance) của lớp đã chỉ định. Còn nếu chưa được thiết lập, Android sử dụng thông tin khác trong đối tượng Intent để đặt thành phần đích thích hợp - xem mục "Phân tích Intent" ("Intent Resolution") sẽ được trình bày ở phần sau của tài liệu này.

Tên thành phần được thiết lập bằng cách sử dụng `setComponent()`, `setClass()` hoặc `setClassName()` và được đọc bởi `getComponent()`.

Action

Một chuỗi (string) chứa tên của action được thực hiện - hoặc, trong trường hợp các intent broadcast, đây sẽ là action đã xảy ra và đang được báo cáo. Lớp Intent định nghĩa một vài hằng số action, gồm:

Hằng số	Thành phần đích	Action
`ACTION_CALL`	activity	Khởi tạo một cuộc gọi điện thoại.
`ACTION_EDIT`	activity	Hiển thị dữ liệu cho người dùng để thực hiện chỉnh sửa.
`ACTION_MAIN`	activity	Bắt đầu với activity khởi tạo của một nhiệm vụ và không có dữ liệu đầu vào (input), đồng thời không trả về dữ liệu đầu ra (output).
`ACTION_SYNC`	activity	Đồng bộ dữ liệu trên máy chủ (server) với dữ liệu trên thiết bị di động.
`ACTION_BATTERY_LOW`	broadcast receiver	Một cảnh báo rằng pin sắp hết.
`ACTION_HEADSET_PLUG`	broadcast receiver	Một bộ tai nghe đã được gắn vào thiết bị, hoặc được rút ra từ thiết bị.
`ACTION_SCREEN_ON`	broadcast receiver	Màn hình đã được bật lên.
`ACTION_TIMEZONE_CHANGED`	broadcast receiver	Các thông số thiết lập cho múi giờ (time zone) đã thay đổi.

Xem mô tả lớp `Intent` để tìm hiểu danh sách các hằng số có sẵn cho những action chung. Các action khác được định nghĩa ở đâu đó trong Android API. Bạn cũng có thể tự định nghĩa các chuỗi action để kích hoạt những thành phần có mặt trong ứng dụng của mình. Các chuỗi mà bạn tạo ra nên bao gồm tên gói ứng dụng như một tiền tố - ví dụ: "`com.example.project.SHOW_COLOR`".

Action quyết định phần lớn cách mà các thông tin còn lại của intent được cấu trúc - ngoại trừ các trường `data` và `extras` - cũng nhiều như việc một tên phương thức (method) quyết định tập các tham số và giá trị trả về. Do vậy, nên sử dụng những tên action cụ thể nhất trong khả năng, đồng thời kết hợp chặt chẽ chúng với các trường khác của intent. Nói cách khác, thay vì định nghĩa một action riêng lẻ, hãy định nghĩa cả một giao thức (protocol) cho những đối tượng intent mà các thành phần trong ứng dụng của bạn có thể xử lý.

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

Dữ liệu

Thành phần URI (Uniform Resource Identifier - Định dạng tài nguyên thống nhất) của dữ liệu được thực thi và loại MIME (Multipurpose internet mail extension - Mở rộng thư tín Internet đa năng) của dữ liệu đó. Những action khác nhau được kết hợp với các loại đặc tả dữ liệu khác nhau thành từng cặp. Ví dụ, nếu trường action là `ACTION_EDIT`, trường dữ liệu sẽ chứa URI của tài liệu được hiển thị để chỉnh sửa. Nếu hành động là `ACTION_CALL`, trường dữ liệu sẽ là tel: URI với số điện thoại để gọi. Tương tự, nếu action là `ACTION_VIEW` và trường dữ liệu là http: URI, activity nhận được sẽ được gọi để tải về, đồng thời hiển thị bất cứ dữ liệu nào mà URI trỏ tới.

Khi khớp một intent với một thành phần có khả năng xử lý dữ liệu, điều quan trọng cần biết là loại của dữ liệu (loại MIME) cùng với URI của dữ liệu ấy. Ví dụ, không nên gọi một thành phần có khả năng hiển thị dữ liệu hình ảnh để chạy một file âm thanh.

Trong nhiều trường hợp, loại dữ liệu có thể được suy đoán từ URI - ngoại trừ các content: URI cho thấy rằng dữ liệu được định vị trên thiết bị và do một Content Provider (trình cung cấp nội dung) kiểm soát (xem mục "Content provider" bên dưới). Tuy nhiên, loại dữ liệu có thể được thiết lập một cách tường minh trong đối tượng Intent. Phương thức `setData()` định rõ rằng dữ liệu chỉ như một URI, `setType()` định rõ rằng nó chỉ như một loại MIME, còn `setDataAndType()` cho biết nó vừa là một URI vừa là một loại MIME. Để đọc URI sử dụng `getData()`, đọc loại dữ liệu sử dụng `getType()`.

Hạng mục

Một chuỗi chứa thông tin bổ sung về những loại thành phần xử lý intent. Đối tượng intent có thể chứa một số lượng không giới hạn các mô tả hạng mục (category). Tương tự như cách đã làm với action, lớp Intent định nghĩa một vài hằng số hạng mục, bao gồm:

Hằng số	Ý nghĩa
CATEGORY_BROWSABLE	Activity đích có thể được trình duyệt (browser) kích hoạt (invoke) một cách an toàn để hiển thị dữ liệu do một đường dẫn (link) trỏ tới - ví dụ như một bức ảnh hoặc một tin nhắn e-mail (thư điện tử).
CATEGORY_GADGET	Activity có thể được nhúng vào bên trong activity khác và là activity có thể làm nền cho các gadget (những ứng dụng nhỏ được thiết kế với tính "lắp ráp" được).
CATEGORY_HOME	Activity hiển thị màn hình chính (home screen), màn hình đầu tiên mà người dùng thấy khi bật thiết bị lên hoặc khi button (nút) *Home* được nhấn.
CATEGORY_LAUNCHER	Activity có thể là activity khởi tạo của một tác vụ (task) và được liệt kê ở mức cao nhất của màn hình chính ứng dụng (application launcher).
CATEGORY_PREFERENCE	Activity đích là một bảng thiết lập tùy chỉnh cá nhân (preference panel).

Xem mô tả lớp Intent để có được danh sách đầy đủ về các hạng mục.

Phương thức addCategory() đặt một hạng mục vào trong một đối tượng Intent, removeCategory() xóa một hạng mục đã thêm vào trước đó, trong khi getCategories() lấy ra tập tất cả các hạng mục hiện có của đối tượng.

Các thành phần phụ

Các cặp khóa-giá trị (key-value) bổ sung thông tin sẽ được gửi tới thành phần xử lý intent. Tương tự một vài action được gắn với những loại URI dữ liệu cụ thể, một số intent sẽ được gắn với các thành phần phụ (extra). Ví dụ, một intent ACTION_TIMEZONE_CHANGED có thành phần phụ "time-zone" để nhận diện các múi giờ mới, còn ACTION_HEADSET_PLUG có thành phần phụ "state" (trạng thái) định rõ hiện giờ bộ tai nghe đã được cắm vào hay rút ra khỏi thiết bị, cùng với đó là thành phần "name" cho loại tai nghe. Nếu bạn đã tạo ra một action SHOW_COLOR, giá trị màu (color value) sẽ được đặt trong một cặp khóa-giá trị phụ.

Đối tượng Intent có một loạt phương thức put...() để chèn nhiều loại dữ liệu phụ trợ và nhóm phương thức tương tự get... () để đọc dữ liệu. Những phương thức song song này nằm trong các đối tượng Bundle. Thực tế, các thành phần phụ có thể được cài đặt và đọc như một Bundle bằng cách sử dụng các phương thức putExtras() và getExtras().

Cờ

Có nhiều loại cờ (flag) khác nhau. Nhiều loại hướng dẫn cho hệ thống Android cách thực thi một activity (ví dụ, activity thuộc về tác vụ nào) và cách xử lý activity sau khi thực thi (ví dụ, nó nằm trong danh sách những activity gần đây không). Tất cả các cờ này được định nghĩa trong lớp Intent.

Hệ thống Android và những ứng dụng đi kèm nền tảng dùng các đối tượng Intent để vừa gửi đi các tin broadcast của hệ thống ban đầu vừa kích hoạt những thành phần đã xác định trong hệ thống. Muốn biết cách cấu trúc một intent để kích hoạt một thành phần hệ thống, mời bạn tham khảo mục "List of intents" ("Danh sách intent").

Phân tích Intent

Các Intent có thể được chia thành hai nhóm:

- *Intent tường minh (explicit intent)* quy định thành phần đích bằng tên của nó (trường tên thành phần - Component Name đề cập ở trên, đã có một giá trị thiết lập). Do các nhà phát triển của những ứng dụng khác không biết tên thành phần ứng dụng của nhau, nên các intent tường minh thường được dùng cho những thông điệp nội bộ trong ứng dụng - ví dụ, một activity khởi chạy một service cấp con hay thực thi một activity anh em của nó.
- *Intent ngầm định (implicit intent)* không đề tên thành phần đích (trường tên thành phần để trống). Intent ngầm định thường được dùng để kích hoạt các thành phần trong những ứng dụng khác.

Android truyền một intent ngầm định tới một thể hiện của lớp đích được chỉ định. Không có gì khác trong đối tượng Intent ngoài tên thành phần quan trọng đối với việc xác định đâu là thành phần nhận intent.

Cần một chiến thuật khác cho các intent ngầm định. Khi thành phần đích không được chỉ định tường minh, hệ thống Android phải tìm ra thành phần (hoặc các thành phần) tốt nhất để xử lý intent - một activity hoặc service riêng lẻ thực hiện action được yêu cầu hoặc tập các broadcast receiver phản hồi thông báo broadcast. Hệ thống Android có thể làm điều đó bằng cách so sánh nội dung của đối tượng Intent với các *bộ lọc intent*, những cấu trúc liên quan đến các thành phần có khả năng nhận intent. Bộ lọc đưa ra các khả năng của một thành phần và phân định những intent mà nó có thể xử lý được. Bộ lọc đem đến cho thành phần khả năng nhận được các intent ngầm định của loại được đưa ra. Nếu một thành phần không có bất kỳ bộ lọc intent nào, nó có thể chỉ nhận được những intent tường minh. Một thành phần có bộ lọc có thể nhận được cả intent tường minh lẫn ngầm định.

Chỉ có ba khía cạnh của một đối tượng Intent được xét đến khi đối tượng được kiểm thử dựa vào một bộ lọc intent:

action

dữ liệu (cả URI lẫn kiểu dữ liệu)

hạng mục

Các thành phần phụ và cờ không tham gia vào việc giải quyết thành phần nào sẽ nhận intent.

Bộ lọc intent

Để thông báo cho hệ thống biết chúng có thể xử lý những intent ngầm định nào, các activity, service và broadcast receiver có thể có một hay nhiều bộ lọc intent. Mỗi bộ lọc mô tả một khả năng của một thành phần, một tập intent mà thành phần sẵn sàng nhận. Kết quả là, nó lọc ra các intent mong muốn và loại bỏ những intent không mong muốn -

chỉ gồm các intent ngầm định (những intent này không đặt tên cho lớp đích). Một intent tường minh luôn được gửi tới đích của nó, bất kể nó chứa nội dung gì; bộ lọc không làm nhiệm vụ tra cứu nội dung. Tuy nhiên, một intent ngầm định chỉ được gửi tới một thành phần khi nó có thể vượt qua được các bộ lọc của thành phần đó.

Một thành phần có nhiều bộ lọc riêng biệt cho từng việc nó có thể làm, cho mỗi giao diện mà nó biểu diễn tới người dùng. Ví dụ, activity có tên NoteEditor của ứng dụng Note Pad có hai bộ lọc - một để khởi động với một ghi chú cụ thể giúp người dùng có thể xem hoặc chỉnh sửa, một bộ lọc khác để bắt đầu với một ghi chú mới, để trống hoàn toàn để người dùng có thể điền vào rồi lưu lại. (Tất cả các bộ lọc của Note Pad đều được miêu tả trong mục "Ví dụ về Note Pad" sẽ trình bày về sau).

Bộ lọc và bảo mật

Một bộ lọc intent không thể dùng cho mục đích bảo mật (security). Trong khi mở một thành phần chỉ để nhận những dạng intent ngầm định nhất định, bộ lọc không làm gì để ngăn ngừa các intent tường minh nhắm tới thành phần đó. Mặc dù một bộ lọc có thể hạn chế các intent, song một thành phần sẽ được yêu cầu xử lý một số action và nguồn dữ liệu nhất định. Tuy nhiên, ai đó vẫn có thể đặt một intent tường minh cạnh một action và nguồn dữ liệu khác rồi đặt tên thành phần giống với thành phần đích.

Bộ lọc intent là một thể hiện của lớp `IntentFilter`. Tuy nhiên, do hệ thống Android phải biết về khả năng của một thành phần trước khi thực thi thành phần đó, nên các bộ lọc intent thường không được thiết lập trong mã Java mà chỉ đóng vai trò là các phần tử (element) `<intent-filter>` trong file kê khai (`AndroidManifest.xml`) của ứng dụng. (Một ngoại lệ là những bộ lọc cho broadcast receiver được đăng ký động là gọi `Context.registerReceiver()`; chúng được tạo trực tiếp dưới dạng các đối tượng IntentFilter).

Một bộ lọc có các trường song song với các trường action, dữ liệu và hạng mục của một đối tượng Intent. Một intent ngầm định được kiểm thử bằng bộ lọc ở cả ba trường. Để được chuyển tới thành phần sở hữu bộ lọc, nó phải vượt qua cả ba kiểm thử. Nếu thất bại cho dù chỉ trong một kiểm thử, hệ thống Android sẽ không gửi nó tới thành phần - ít nhất là không dựa trên cơ sở của bộ lọc đó. Tuy nhiên, do mỗi thành phần có thể có nhiều bộ lọc intent, nên một intent không vượt qua một trong các bộ lọc của thành phần đó lại có thể vượt qua bộ lọc khác.

Mỗi kiểm thử được mô tả chi tiết dưới đây:

Kiểm thử action

Một phần tử `<intent-filter>` trong file kê khai liệt kê các action dưới dạng những phần tử con `<action>`. Ví dụ:

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

Ví dụ trên cho thấy, trong khi một đối tượng Intent chỉ đặt tên cho một action thì một bộ lọc có thể liệt kê nhiều hơn thế. Danh sách không thể để trống; một bộ lọc phải chứa ít nhất một phần tử <action>, hoặc nó sẽ chặn (block) tất cả các intent.

Để vượt qua kiểm thử này, action đã được ấn định trong đối tượng Intent phải khớp với một trong các action được liệt kê trong bộ lọc. Nếu đối tượng hoặc bộ lọc không chỉ định một action, kết quả sẽ như sau:

- Nếu bộ lọc thất bại trong việc liệt kê bất cứ action nào, intent không có gì để so khớp, nên tất cả intent sẽ thất bại trong bài kiểm thử. Không intent nào có khả năng vượt qua bộ lọc.
- Mặt khác, một đối tượng Intent không ấn định một action sẽ tự động vượt qua bài kiểm thử - miễn là bộ lọc chứa ít nhất một action.

Kiểm thử hạng mục

Một phần tử <intent-filter> liệt kê các hạng mục dưới dạng những phần tử con. Ví dụ:

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

Lưu ý, các hằng số mô tả lúc trước cho các action và hạng mục đều không được sử dụng trong file kê khai. Toàn bộ giá trị của chuỗi được sử dụng thay vào đó. Ví dụ, chuỗi "android.intent.category.BROWSABLE" trong ví dụ trên tương ứng với hằng số CATEGORY_BROWSABLE đã đề cập trước đó trong tài liệu này. Tương tự, chuỗi "android.intent.action.EDIT" tương ứng với hằng số ACTION_EDIT.

Để một intent vượt qua kiểm thử hạng mục, mỗi hạng mục trong đối tượng Intent đều phải khớp với một hạng mục trong bộ lọc. Bộ lọc có thể liệt kê những hạng mục bổ sung, nhưng không thể loại bỏ các hạng mục trong intent.

Về nguyên tắc, một đối tượng Intent không có hạng mục luôn vượt qua bài kiểm thử, bất kể trong bộ lọc chứa gì. Điều này đúng trong hầu hết trường hợp. Tuy nhiên, với một ngoại lệ, Android xử lý tất cả các intent ngầm định thông qua startActivity() như thể là chúng chứa ít nhất một hạng mục: "android.intent.category.DEFAULT" (hằng số CATEGORY_DEFAULT). Vì vậy, các activity sẵn sàng nhận những intent ngầm định đều phải bao gồm "android.intent.category.DEFAULT" trong các bộ lọc intent của chúng. (Những bộ lọc với thiết lập "android.intent.action.MAIN" và "android.intent.category.LAUNCHER" là ngoại lệ. Chúng đánh dấu các activity bắt đầu tác vụ mới và được biểu diễn trong màn hình khởi động. Chúng có thể bao gồm "android.intent.category.DEFAULT" trong danh sách các hạng mục, nhưng điều này là không cần thiết). Tham khảo mục "Sử dụng tính năng so khớp intent" để biết thêm thông tin về các bộ lọc này.

Kiểm thử dữ liệu

Tương tự action và hạng mục, đặc tả dữ liệu (data specification) của một bộ lọc intent được bao gồm trong một phần tử con. Và, với những trường hợp như vậy, phần tử con có thể xuất hiện nhiều lần, hoặc không lần nào. Ví dụ:

```
<intent-filter . . . >
   <data android:mimeType="video/mpeg" android:scheme="http" . . . />
   <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

Mỗi phần tử <data> có thể xác định một URI và một kiểu dữ liệu (kiểu MIME đa phương tiện). Có nhiều thuộc tính riêng biệt - lược đồ (scheme), host, port (cổng) và đường dẫn (path) - đối với từng phần của URI:

```
scheme://host:port/path
```

Ví dụ, trong URI dưới đây,

```
content://com.example.project:200/folder/subfolder/etc
```

lược đồ là "content", host là "com.example.project", port là "200", còn đường dẫn là "folder/subfolder/etc". Host và port cùng nhau tạo thành *chuỗi định danh (authority)* của URI; nếu không ấn định host, port sẽ bị bỏ qua.

Mỗi thuộc tính đều không bắt buộc, nhưng chúng liên quan với nhau: Để một định danh có ý nghĩa, lược đồ phải được ấn định. Để một đường dẫn có nghĩa, cả lược đồ và định danh đều phải được ấn định.

Khi URI trong đối tượng Intent được so sánh với đặc tả URI trong bộ lọc, nó chỉ được đối sánh với những bộ phận của URI thực sự được đề cập trong bộ lọc. Ví dụ, nếu một bộ lọc chỉ ấn định một lược đồ, tất cả các URI với lược đồ đó đều khớp với bộ lọc. Nếu một bộ lọc ấn định một lược đồ và một định danh nhưng không có đường dẫn thì tất cả các URI có lược đồ và định danh tương tự đều khớp, bất kể đường dẫn như thế nào. Nếu một bộ lọc ấn định một lược đồ, một định danh và một đường dẫn thì chỉ những URI có lược đồ, định danh và đường dẫn đó mới khớp. Tuy nhiên, một đặc tả đường dẫn trong bộ lọc có thể chứa các ký tự đại diện (wildcard) để yêu cầu chỉ một phần đường dẫn là khớp.

Thuộc tính type của phần tử <data> xác định kiểu MIME của dữ liệu. Nó phổ biến ở các bộ lọc hơn so với URI. Cả đối tượng Intent lẫn bộ lọc đều có thể sử dụng ký tự đại diện "*" cho trường loại con (subtype) - ví dụ, "text/*" hoặc "audio/*" - chỉ ra bất cứ loại con nào khớp.

Quá trình kiểm thử dữ liệu so sánh cả URI lẫn kiểu dữ liệu trong đối tượng Intent với một URI và kiểu dữ liệu xác định trong bộ lọc. Nguyên tắc như sau:

a. Một đối tượng Intent không chứa URI hay kiểu dữ liệu sẽ vượt qua kiểm thử chỉ khi bộ lọc tương tự không ấn định bất kỳ URI hoặc kiểu dữ liệu nào.

b. Một đối tượng Intent chứa URI nhưng không có kiểu dữ liệu (và không thể suy ra một kiểu dữ liệu từ URI) sẽ vượt qua kiểm thử chỉ khi URI của nó khớp với một URI trong bộ lọc, còn bộ lọc tương tự không ấn định kiểu nào. Đây sẽ là

trường hợp duy nhất cho các URI như `mailto:` và `tel:` mà không đề cập đến dữ liệu thực tế.

c. Một đối tượng Intent chứa một kiểu dữ liệu nhưng không có URI sẽ vượt qua kiểm thử chỉ khi bộ lọc liệt kê cùng kiểu dữ liệu và tương tự vậy, không xác định URI nào.

d. Một đối tượng Intent chứa cả URI lẫn kiểu dữ liệu (hoặc có thể suy ra kiểu dữ liệu từ URI) vượt qua phần kiểm thử kiểu dữ liệu chỉ khi kiểu của nó khớp với một kiểu được liệt kê trong bộ lọc. Nó vượt qua phần kiểm thử URI khi URI của nó khớp với một URI trong bộ lọc, hoặc nếu nó chứa `content: URI` hay `file: URI` và bộ lọc không xác định URI nào. Nói cách khác, một thành phần được coi là hỗ trợ dữ liệu `content:` và `file:`, nếu bộ lọc của nó chỉ liệt kê kiểu dữ liệu.

Nếu một intent có thể vượt qua các bộ lọc của nhiều activity hoặc service, người dùng sẽ được hỏi thành phần nào được kích hoạt. Một ngoại lệ xảy ra khi không có thành phần đích nào được tìm thấy.

Các trường hợp phổ biến

Nguyên tắc cuối cùng trình bày ở trên dành cho việc kiểm thử dữ liệu, nguyên tắc (d), phản ánh mong muốn rằng các thành phần có thể lấy dữ liệu nội bộ từ một file hoặc Content Provider. Do đó, bộ lọc của chúng có thể liệt kê chỉ một kiểu dữ liệu mà không cần xác định tường minh tên lược đồ `content:` và `file:`. Đây là một trường hợp điển hình. Ví dụ, một phần tử `<data>` như sau cho Android biết thành phần có thể lấy dữ liệu hình ảnh từ Content Provider và hiển thị nó:

```
<data android:mimeType="image/*" />
```

Do hầu hết dữ liệu đều sẵn sàng được xuất ra từ Content Provider, nên các bộ lọc chỉ ấn định kiểu dữ liệu mà không chứa URI là trường hợp phổ biến nhất.

Một cấu hình thông dụng khác là các bộ lọc với một lược đồ và một kiểu dữ liệu. Ví dụ, một phần tử `<data>` như sau cho Android biết thành phần có thể lấy dữ liệu video từ mạng (network) và hiển thị nó:

```
<data android:scheme="http" android:type="video/*" />
```

Ví dụ, hãy xem xét điều mà các ứng dụng trình duyệt (browser application) làm khi người dùng nhấn vào một liên kết trên trang Web. Trước tiên, ứng dụng trình duyệt sẽ cố gắng hiển thị dữ liệu (nó có thể thực hiện điều này nếu liên kết dẫn đến một trang HTML). Nếu không thể hiển thị dữ liệu, ứng dụng sẽ đặt intent ngầm định cùng với lược đồ và kiểu dữ liệu rồi khởi động một activity có thể làm được việc đó. Nếu không có activity nào phù hợp, ứng dụng sẽ yêu cầu trình quản lý tải về (download manager) tải dữ liệu xuống. Nó được đặt dưới quyền kiểm soát của Content Provider để có một bộ activity lớn hơn (những activity có bộ lọc chỉ ấn định kiểu dữ liệu) có thể phản hồi lại.

Hầu hết ứng dụng đều có một cách để làm mới lại mà không cần tham chiếu tới bất kỳ dữ liệu cụ thể nào. Những activity có thể khởi tạo các ứng dụng có bộ lọc với "android.intent.action.MAIN" được xác định dưới dạng action. Nếu được hiển thị trong màn hình chính (laucher) của ứng dụng, chúng cũng xác định hạng mục "android.intent.category.LAUNCHER":

```
<intent-filter . . . >
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />

</intent-filter>
```

Sử dụng tính năng so khớp intent

Các intent được so khớp với bộ lọc intent không chỉ nhằm khám phá thành phần đích để kích hoạt mà còn nhằm phát hiện điều gì đó về tập các thành phần/bộ phận trên thiết bị. Ví dụ, hệ thống Android tạo ra màn hình khởi động của ứng dụng, màn hình mức cao nhất cho thấy các ứng dụng đã sẵn sàng để người dùng khởi động, bằng cách tìm tất cả các activity với bộ lọc intent có chỉ định action "android.intent.action.MAIN" và hạng mục "android.intent.category.LAUNCHER" (như đã minh họa ở mục trước). Sau đó, nó hiển thị biểu tượng (icon) và nhãn (label) của các activity trong màn hình khởi động. Tương tự, nó khám phá màn hình chính bằng cách tìm kiếm activity có chứa "android.intent.category.HOME" trong bộ lọc của nó.

Ứng dụng của bạn có thể sử dụng cách so khớp intent tương tự. PackageManager có một tập các phương thức query...() trả về tất cả những thành phần có thể chấp nhận một intent cụ thể, và một chuỗi tương tự các phương thức resolve...() xác định thành phần phù hợp nhất để phản hồi lại intent. Ví dụ, queryIntentActivities() trả về danh sách tất cả các activity có thể thực thi với intent đã vượt qua bộ lọc làm tham số, và queryIntentServices() trả về danh sách tương tự các service. Cả hai phương thức đều không kích hoạt các thành phần; chúng chỉ liệt kê những activity có thể phản hồi. Có một phương thức tương tự, queryBroadcastReceivers(), cho các broadcast receiver.

Ví dụ về Note Pad

Ứng dụng Note Pad cho phép người dùng có thể duyệt qua một danh sách các ghi chú (note), xem chi tiết từng mục trong danh sách, chỉnh sửa các mục và thêm mục mới vào danh sách. Phần này xem xét các bộ lọc intent được khai báo (declared) trong file kê khai. (Nếu đang làm việc ngoại tuyến trong SDK (Software Developement Kit - Bộ công cụ phát triển phần mềm), bạn có thể tìm thấy toàn bộ file nguồn cho ứng dụng ví dụ này, bao gồm cả file kê khai của nó tại <sdk>/samples/NotePad/index.html. Nếu bạn đang xem tài liệu trực tuyến, các file nguồn đều có ở mục "Tutorials and Sample" ("Ví dụ mẫu và tài liệu hướng dẫn giảng dạy") tại đây).

Trong file kê khai của nó, ứng dụng Note Pad khai báo ba activity, mỗi activity ứng với ít nhất một bộ lọc intent. Nó cũng khai báo một Content Provider để quản lý dữ liệu ghi chú. Đây là nội dung file kê khai:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.android.notepad">
    <application android:icon="@drawable/app_notes"
                 android:label="@string/app_name" >

        <provider android:name="NotePadProvider"
                android:authorities="com.google.provider.NotePad" />

     <activity android:name="NotesList" android:label="@string/title_notes_list">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.EDIT" />
                <action android:name="android.intent.action.PICK" />
             <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
             </intent-filter>
             <intent-filter>
              <action android:name="android.intent.action.GET_CONTENT" />
              <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
             </intent-filter>
        </activity>

        <activity android:name="NoteEditor"
                  android:theme="@android:style/Theme.Light"
                  android:label="@string/title_note" >
            <intent-filter android:label="@string/resolve_edit">
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.EDIT" />
             <action android:name="com.android.notepad.action.EDIT_NOTE" />
             <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
             </intent-filter>
             <intent-filter>
               <action android:name="android.intent.action.INSERT" />
             <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
             </intent-filter>
        </activity>

        <activity android:name="TitleEditor"
                  android:label="@string/title_edit_title"
   android:theme="@android:style/Theme.Dialog">
<intent-filter android:label="@string/resolve_title">
            <action android:name="com.android.notepad.action.EDIT_TITLE" />
             <category android:name="android.intent.category.DEFAULT" />
```

```
            <category android:name="android.intent.category.ALTERNATIVE" />
            <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Activity đầu tiên, `NotesList`, khác biệt với những activity khác vì thực tế, nó hoạt động trên một danh mục ghi chú (danh sách ghi chú) thay vì trên một ghi chú đơn lẻ. `NotesList` thường được dùng làm giao diện người dùng đầu tiên, khi mới truy cập ứng dụng. Nó có thể làm ba việc được mô tả bằng ba bộ lọc intent:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
```

Bộ lọc này khai báo điểm bắt đầu chính (main entry point) cho ứng dụng Note Pad. Action `MAIN` chuẩn là điểm bắt đầu không đòi hỏi bất cứ thông tin nào khác trong đối tượng Intent (ví dụ như không có đặc tả dữ liệu), còn hạng mục `LAUNCHER` cho thấy điểm bắt đầu nên được liệt kê trong màn hình chính của ứng dụng.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.PICK" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
  </intent-filter>
```

Bộ lọc này khai báo những thứ mà activity có thể làm trên một danh mục các ghi chú. Nó có thể cho phép người dùng xem hoặc chỉnh sửa danh mục (thông qua action `VIEW` và `EDIT`), hoặc chọn một ghi chú đặc biệt từ danh mục (thông qua action `PICK`).

Thuộc tính `mimeType` của phần tử `<data>` quy định loại dữ liệu mà các action này có thể hoạt động cùng. Nó ngụ ý rằng activity có thể có một con trỏ (cursor) trên không hoặc nhiều mục (`vnd.android.cursor.dir`) từ Content Provider nắm giữ dữ liệu Note Pad (`vnd.google.note`). Đối tượng Intent thực thi activity sẽ bao gồm một `content: URI` xác định chính xác dữ liệu của dạng này và activity nên mở ra.

Lưu ý, hạng mục `DEFAULT` cũng được cung cấp trong bộ lọc này. Đó là do các phương thức `Context.startActivity()` và `Activity.startActivityForResult()` coi mọi intent đều chứa hạng mục `DEFAULT` - chỉ với hai ngoại lệ:

- Các intent định danh activity đích một cách tường minh.
- Các intent bao gồm action `MAIN` và hạng mục `LAUNCHER`.

Do đó, hạng mục DEFAULT là *cần thiết* cho tất cả các bộ lọc - ngoại trừ những bộ lọc với action MAIN và hạng mục LAUNCHER. (Các bộ lọc intent không cần quan tâm đến những intent tường minh).

```
<intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
```

Bộ lọc này mô tả khả năng của activity trong việc trả lại một ghi chú đã được người dùng chọn, không đòi hỏi bất cứ đặc tả của thư mục nào mà người dùng nên chọn từ đó. Action GET_CONTENT tương tự như action PICK. Trong cả hai trường hợp, activity trả về cho URI một ghi chú đã được người dùng chọn. (Trong mỗi trường hợp, nó được trả về cho activity gọi là startActivityForResult() để bắt đầu activity NoteList). Tuy nhiên, tại đây, đối tượng gọi (caller) sẽ xác định kiểu dữ liệu mong muốn thay vì thư mục dữ liệu mà người dùng sẽ chọn trong đó.

Kiểu dữ liệu, vnd.android.cursor.item/vnd.google.note, chỉ ra dạng dữ liệu của activity có thể trả về - một URI cho một ghi chú đơn lẻ. Từ URI được trả về, đối tượng gọi có thể có một con trỏ cho đúng một mục (vnd.android.cursor.item) từ Content Provider dung chứa dữ liệu Note Pad (vnd.google.note).

Nói cách khác, đối với action PICK trong bộ lọc trước đó, kiểu dữ liệu chỉ ra dạng của dữ liệu mà activity có thể hiển thị cho người dùng. Đối với bộ lọc GET_CONTENT, nó chỉ ra dạng của dữ liệu mà activity có thể trả về cho đối tượng gọi.

Nhờ được trang bị những khả năng ấy, các intent sau đây sẽ giải quyết activity NotesList:

action: android.intent.action.MAIN

Thực thi activity mà không có dữ liệu xác định.

action: android.intent.action.MAIN
hạng mục: android.intent.category.LAUNCHER

Thực thi activity mà không có dữ liệu được chọn xác định. Đây thực sự là intent được màn hình chính sử dụng để sinh ra danh sách cấp cao nhất của nó. Tất cả các activity với những bộ lọc khớp với action và hạng mục này đều được thêm vào danh sách.

action: android.intent.action.VIEW
dữ liệu: content://com.google.provider.NotePad/notes

Ra lệnh cho activity hiển thị một danh sách các ghi chú dưới content://com.google.provider.NotePad/notes. Sau đó, người dùng có thể duyệt danh sách và lấy thông tin về các mục trong đó.

action: android.intent.action.PICK
dữ liệu: content://com.google.provider.NotePad/notes

Ra lệnh cho activity hiển thị một danh sách các ghi chú dưới content://com.google.provider.NotePad/notes. Sau đó, người dùng có thể

chọn một ghi chú từ danh sách, và activity sẽ trả URI cho mục đó về activity đã khởi động activity NoteList.

action: `android.intent.action.GET_CONTENT`
kiểu dữ liệu: `vnd.android.cursor.item/vnd.google.note`

Ra lệnh cho activity cung cấp một mục đơn trong dữ liệu Note Pad.

Activity thứ hai, `NoteEditor`, cho người dùng thấy một mục ghi chú đơn, đồng thời cho phép họ chỉnh sửa nó. Nó có thể làm hai điều được mô tả bằng hai bộ lọc intent dưới đây:

```
<intent-filter android:label="@string/resolve_edit">
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="com.android.notepad.action.EDIT_NOTE" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
```

Mục đích trước tiên và chính yếu của activity này là cho phép người dùng tương tác với một ghi chú đơn lẻ - để XEM (VIEW) hoặc CHỈNH SỬA (EDIT) ghi chú. (Hạng mục `EDIT_NOTE` tương tự như `EDIT`). Intent sẽ chứa URI cho dữ liệu khớp với kiểu MIME `vnd.android.cursor.item/vnd.google.note` - đó là URI cho một ghi chú đơn, cụ thể. Thông thường, đó sẽ là một URI trả về bởi action `PICK` hoặc `GET_CONTENT` của activity `NoteList`.

Cũng như trước đây, bộ lọc này liệt kê hạng mục `DEFAULT` sao cho activity có thể được thực thi bởi các intent không xác định một cách tường minh lớp `NoteEditor`.

```
<intent-filter>
    <action android:name="android.intent.action.INSERT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
  </intent-filter>
```

Mục đích thứ hai của activity này là cho phép người dùng tạo một ghi chú mới, ghi chú này sẽ CHÈN (INSERT) vào một thư mục ghi chú đã tồn tại. Intent sẽ chứa URI cho dữ liệu khớp với kiểu MIME `vnd.android.cursor.dir/vnd.google.note` - đó là URI cho thư mục mà ghi chú sẽ được đặt vào đó.

Với những khả năng trên, các intent sau sẽ giải quyết activity `NoteEditor`:

action: `android.intent.action.VIEW`
dữ liệu: `content://com.google.provider.NotePad/notes/ID`

Ra lệnh cho activity hiển thị nội dung của ghi chú xác định bởi `ID`. (Để biết thêm chi tiết về cách các `content: URI` xác định từng thành viên của một nhóm, xem mục "Content Provider" bên dưới).

action: `android.intent.action.EDIT`
dữ liệu: `content://com.google.provider.NotePad/notes/ID`

Ra lệnh cho activity hiển thị nội dung của ghi chú xác định bởi `ID`, đồng thời cho phép người dùng chỉnh sửa nó. Nếu người dùng lưu các thay đổi, activity sẽ cập nhật dữ liệu cho ghi chú trong Content Provider.

action: `android.intent.action.INSERT`
dữ liệu: `content://com.google.provider.NotePad/notes`

Ra lệnh cho activity tạo một ghi chú mới trong danh sách ghi chú mới, rỗng tại `content://com.google.provider.notepad/notes`, đồng thời cho phép người dùng chỉnh sửa nó. Nếu người dùng lưu ghi chú, URI của nó được trả về cho đối tượng gọi.

Activity cuối cùng, `TitleEditor`, cho phép người dùng chỉnh sửa tiêu đề (title) của ghi chú. Việc này có thể được thực thi bằng cách trực tiếp kích hoạt activity (nhờ thiết lập tên thành phần của nó trong Intent một cách tường minh), mà không cần sử dụng bộ lọc intent nào. Nhưng ở đây, chúng ta cần bắt lấy cơ hội chỉ ra cách triển khai những activity thay thế trên dữ liệu đã có sẵn:

```
<intent-filter android:label="@string/resolve_title">
    <action android:name="com.android.notepad.action.EDIT_TITLE" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
```

Bộ lọc đơn cho activity này sử dụng một action tùy chỉnh gọi là "`com.android.notepad.action.EDIT_TITLE`". Nó phải được kích hoạt trên một ghi chú cụ thể (loại dữ liệu `vnd.android.cursor.item/vnd.google.note`), giống như các action `VIEW` và `EDIT` lúc trước. Tuy nhiên, ở đây, activity hiển thị tiêu đề chứa trong dữ liệu ghi chú chứ không phải bản thân nội dung của ghi chú.

Ngoài việc hỗ trợ hạng mục `DEFAULT` như thường lệ, trình chỉnh sửa tiêu đề (title editor) còn hỗ trợ hai tiêu chuẩn hạng mục khác: `ALTERNATIVE` và `SELECTED_ALTERNATIVE`. Những hạng mục này xác định các activity có thể biểu diễn cho người dùng trong một menu gồm các tùy chọn (cũng như hạng mục `LAUNCHER` xác định những activity nên được biểu diễn cho người dùng trong màn hình chính của ứng dụng). Lưu ý, bộ lọc còn hỗ rợ một nhãn tường minh (thông qua `android:label="@string/resolve_title"`) để kiểm soát tốt hơn những gì người dùng thấy khi trình bày với activity này như là một action thay thế cho dữ liệu mà họ đang xem. (Để biết thêm thông tin về các hạng mục này và việc xây dựng những menu tùy chọn, xem các phương thức `PackageManager.queryIntentActivityOptions()` và `Menu.addIntentOptions()`).

Nhờ được trang bị những khả năng trên, các intent sau sẽ giải quyết activity `TitleEditor`:

action: `com.android.notepad.action.EDIT_TITLE`
dữ liệu: `content://com.google.provider.NotePad/notes/ID`

Ra lệnh cho activity hiển thị tiêu đề liên kết với ghi chú `ID`, đồng thời cho phép người dùng chỉnh sửa tiêu đề.

3. Kiến thức cơ bản về ứng dụng

Các ứng dụng Android được viết bằng ngôn ngữ lập trình Java. Các công cụ thuộc Android SDK biên dịch (compile) mã nguồn - kèm theo bất cứ file dữ liệu và tài nguyên nào - thành một *gói Android (package)*, đó là một file lưu trữ (archive file) với hậu tố `.apk`. Toàn bộ mã nguồn trong một file `.apk` được coi là một ứng dụng, đây chính là file mà các thiết bị nền Android dùng để cài đặt ứng dụng.

Khi đã được cài đặt trên thiết bị, mỗi ứng dụng Android đều "sống" trong vùng sandbox (môi trường ảo chạy ứng dụng) bảo mật của mình:

- Hệ điều hành Android là hệ thống Linux đa người dùng (multi-user), trong mỗi ứng dụng lại là một người dùng khác.
- Theo mặc định, hệ thống gán (assign) cho mỗi ứng dụng một user ID duy nhất (ID này chỉ được hệ thống dùng và ứng dụng không biết tới nó). Hệ thống thiết lập quyền (permission) cho tất cả các file trong ứng dụng, sao cho chỉ duy nhất user ID được gán cho ứng dụng đó mới có thể truy cập (access) vào những file này.
- Mỗi tiến trình đều có máy ảo (virtual machine - VM) riêng của nó, nên mã của một ứng dụng sẽ chạy độc lập với các ứng dụng khác.
- Theo mặc định, mỗi ứng dụng sẽ chạy trong tiến trình Linux của riêng mình. Android khởi động tiến trình khi có bất cứ thành phần nào của ứng dụng cần được thực thi, sau đó tắt tiến trình khi nó không còn cần thiết hoặc khi hệ thống phải khôi phục (recover) bộ nhớ (memory) cho các ứng dụng khác.

Bằng cách này, hệ thống Android đã áp dụng *nguyên lý đặc quyền tối thiểu (principle of least privelege)*. Đó là: Theo mặc định, mỗi ứng dụng chỉ có quyền truy cập tới những thành phần mà ứng dụng yêu cầu phải làm việc cho nó và không thể hơn nữa. Điều này tạo ra một môi trường rất an toàn, trong đó một ứng dụng không thể truy cập vào những phần thuộc hệ thống mà nó không được cấp quyền.

Tuy nhiên, có nhiều cách để một ứng dụng chia sẻ dữ liệu với những ứng dụng khác và để một ứng dụng truy cập vào các dịch vụ của hệ thống:

- Có thể sắp xếp cho hai ứng dụng chia sẻ cùng user ID để chúng có thể truy cập vào các file của nhau. Để bảo toàn tài nguyên hệ thống, những ứng dụng có user ID giống nhau cũng có thể sắp xếp để chạy trong cùng một tiến trình và chia sẻ cùng một máy ảo (các ứng dụng phải được đăng ký (sign) với cùng một chứng nhận (certificate)).
- Một ứng dụng có thể yêu cầu quyền truy cập dữ liệu của thiết bị, ví dụ như danh bạ liên lạc của người dùng, tin nhắn SMS, không gian lưu trữ đã được mount (gắn kết) vào hệ thống (tức thẻ SD), camera, Bluetooth,... Mọi quyền của ứng dụng phải được người dùng cho phép trong thời gian cài đặt.

Các nội dung bên trên bao gồm những kiến thức cơ bản về cách thức một ứng dụng Android tồn tại trong hệ thống. Phần còn lại của tài liệu này giới thiệu:

- Các thành phần framework cốt lõi làm nên ứng dụng của bạn.
- File kê khai, trong đó bạn khai báo những thành phần và tính năng thiết bị cần thiết cho ứng dụng.

- Các tài nguyên được tách biệt khỏi mã ứng dụng và cho phép ứng dụng tối ưu hóa hành vi (behavior) của nó cho nhiều loại cấu hình thiết bị khác nhau.

Các thành phần của ứng dụng

Các thành phần (component) của ứng dựng đóng vai trò là những viên gạch nền tảng cần thiết để xây dựng một ứng dụng Android. Mỗi thành phần là một điểm khác nhau mà nhờ nó, hệ thống có thể truy cập vào ứng dụng. Không phải tất cả các thành phần đều là điểm bắt đầu cho người dùng và một số sẽ phụ thuộc vào những thành phần còn lại. Tuy nhiên, mỗi thành phần đều có thực thể (entity) riêng và đóng một vai trò cụ thể - mỗi thành phần là một viên gạch độc nhất quyết định hành vi tổng quan của ứng dụng.

Có bốn loại thành phần khác nhau của ứng dụng. Mỗi loại phục vụ cho một mục đích khác nhau và có vòng đời (lifecycle) khác biệt xác định cách thức thành phần được tạo ra cũng như hủy bỏ.

Sau đây là bốn loại thành phần của ứng dụng:

Activity

Activity biểu diễn một màn hình (screen) với giao diện người dùng trên đó. Ví dụ, một ứng dụng e-mail có thể có một activity liệt kê danh sách những e-mail mới, một activity khác để soạn thảo e-mail và một activity khác dùng để đọc e-mail. Mặc dù các activity làm việc cùng nhau để tạo nên trải nghiệm người dùng đồng nhất trong ứng dụng e-mail, song chúng lại độc lập với nhau. Như vậy, một ứng dụng khác có thể khởi động bất cứ một activity nào trong số chúng (nếu ứng dụng e-mail cho phép điều này). Ví dụ, một ứng dụng camera có thể khởi động activity trong ứng dụng e-mail để soạn thảo mail mới, trong đó người dùng có thể chia sẻ một bức ảnh.

Một activity được cài đặt là lớp con của `Activity` và bạn có thể tham khảo thêm thông tin về nó trong hướng dẫn cho nhà phát triển Activity.

Service

Service là thành phần chạy ngầm (background) để thực hiện một chuỗi hoạt động hoặc thực hiện công việc cho các tiến trình từ xa (remote process). Service không cung cấp giao diện người dùng. Ví dụ, một service có thể chơi nhạc, service này chạy ngầm trong khi người dùng đang sử dụng một ứng dụng khác, hoặc nó có thể lấy dữ liệu trên mạng mà không ngăn người dùng tương tác với activity khác. Thành phần khác, chẳng hạn như một activity, có thể khởi động một service và để nó chạy hoặc liên kết (bind) với nó để tương tác.

Một service được cài đặt là lớp con của `Service` và bạn có thể tham khảo thêm thông tin về nó trong hướng dẫn cho nhà phát triển Service.

Content Provider

Content provider quản lý một tập dữ liệu của ứng dụng được chia sẻ. Bạn có thể lưu trữ dữ liệu trong hệ thống file, trong cơ sở dữ liệu SQLite, trên Web hay bất kỳ nơi lưu trữ lâu dài nào mà ứng dụng có thể truy cập. Thông qua Content Provider, các ứng dụng khác có thể truy vấn (query) hoặc thậm chí sửa đổi dữ liệu (nếu Content Provider cho phép). Ví dụ, hệ thống Android cung cấp một Content Provider

để quản lý thông tin liên lạc của người dùng. Như vậy, một ứng dụng bất kỳ với quyền phù hợp có thể truy vấn một phần của Content Provider (chẳng hạn như `ContactsContract.Data`) để đọc và ghi thông tin về một người cụ thể nào đó).

Các Content Provider cũng rất hữu ích trong việc đọc, ghi dữ liệu riêng tư đối với ứng dụng của bạn và không chia sẻ đi. Ví dụ, ứng dụng mẫu `Note Pad` dùng một Content Provider để lưu các ghi chú.

Content Provider được cài đặt là một lớp con của `ContentProvider` và phải cài đặt một tập các API chuẩn cho phép những ứng dụng khác thực hiện các giao dịch (transaction). Để biết thêm thông tin, xem hướng dẫn cho nhà phát triển "Content Providers" ("Trình cung cấp nội dung")..

Broadcast receiver

Broadcast receiver là thành phần phản hồi lại các thông báo quảng bá của toàn hệ thống. Nhiều broadcast xuất phát từ hệ thống - ví dụ, một broadcast thông báo rằng màn hình đã tắt, pin sắp hết hoặc một bức ảnh đang được chụp. Các ứng dụng cũng có thể khởi tạo broadcast - ví dụ, để cho những ứng dụng khác biết rằng một số dữ liệu đã được tải về thiết bị và sẵn sàng cho người dùng sử dụng. Mặc dù các broadcast receiver không hiển thị giao diện người dùng, song chúng có thể tạo một thông báo qua thanh trạng thái (create a status bar notification) để báo cho người dùng khi xảy ra một sự kiện broadcast. Mặc dù thường thì một broadcast receiver chỉ là một "cổng ra" ("gateway") tới các thành phần khác và dùng để thực hiện một khối lượng công việc rất nhỏ. Ví dụ, broadcast receiver có thể khởi tạo một service để thực hiện một số việc dựa trên sự kiện.

Broadcast receiver được cài đặt là một lớp con của `BroadcastReceiver` và mỗi broadcast được gửi dưới dạng một đối tượng `Intent`. Để biết thêm thông tin, xem lớp `BroadcastReceiver`.

Một khía cạnh độc đáo trong thiết kế hệ thống Android là bất cứ ứng dụng nào cũng có thể khởi động thành phần của ứng dụng khác. Ví dụ, nếu bạn muốn chụp một bức ảnh bằng camera của thiết bị, có thể một ứng dụng khác sẽ làm điều đó và ứng dụng của bạn sẽ sử dụng nó, thay vì bạn phải tự phát triển một activity để chụp ảnh. Bạn không phải kết hợp chặt chẽ hoặc thậm chí liên kết tới mã nguồn từ ứng dụng camera. Thay vào đó, bạn chỉ cần làm một việc đơn giản là khởi động activity có chức năng chụp ảnh trong ứng dụng camera. Khi đã hoàn thành, bức ảnh được trả về cho ứng dụng của bạn để bạn có thể sử dụng nó. Đối với người dùng, có vẻ như camera thực sự là một phần trong ứng dụng của bạn.

Khi hệ thống khởi động một thành phần, nó khởi động tiến trình cho ứng dụng đó (nếu nó chưa chạy sẵn) và khởi tạo các lớp cần thiết cho thành phần. Ví dụ, nếu ứng dụng của bạn khởi động activity trong ứng dụng camera để chụp một bức ảnh, activity sẽ chạy trong tiến trình thuộc về ứng dụng camera chứ không phải trong tiến trình thuộc về ứng dụng của bạn. Do đó, không giống như các ứng dụng trên hầu hết những hệ thống khác, các ứng dụng Android không có điểm bắt đầu đơn lẻ (ví dụ như không có hàm (function) `main()`).

Do hệ thống chạy từng ứng dụng trong một tiến trình riêng, với quyền file hạn chế truy cập vào các ứng dụng khác, nên ứng dụng của bạn không thể trực tiếp kích hoạt một thành phần từ ứng dụng khác. Tuy nhiên, hệ thống Android lại có khả năng thực hiện

điều đó. Vì vậy, để kích hoạt một thành phần trong ứng dụng khác, bạn phải gửi một thông điệp tới hệ thống xác định *intent* của mình để khởi động một thành phần cụ thể. Sau đó, hệ thống sẽ kích hoạt thành phần cho bạn.

Kích hoạt các thành phần

Ba trong bốn loại thành phần - activity, service và broadcast receiver - được kích hoạt bằng một thông điệp không đồng bộ (asynchronous message) gọi là một *intent*. Các intent kết nối những thành phần riêng rẽ lại với nhau trong lúc chạy chương trình (bạn có thể hình dung chúng như những người truyền thư yêu cầu một action từ các thành phần khác), bất kể thành phần thuộc về ứng dụng của bạn hay ứng dụng khác.

Mỗi intent được tạo ra với một đối tượng Intent, đối tượng này định nghĩa một thông điệp (message) để kích hoạt một thành phần cụ thể hoặc một *loại (type)* thành phần cụ thể - một intent có thể là intent tường minh hoặc không tường minh.

Đối với các activity và service, một `intent` định nghĩa một action để thực thi (ví dụ, để "xem" ("view") hoặc "gửi" ("send") cái gì đó), đồng thời có thể xác định URI của dữ liệu để thao tác với (đây là một trong số những thông tin mà thành phần đang được khởi động cần biết). Ví dụ, một intent có thể truyền đạt một yêu cầu (request) tới một activity để mở một tấm ảnh hoặc một trang Web. Trong vài trường hợp, bạn có thể khởi động một activity để nhận kết quả, khi đó activity cũng trả về kết quả dưới dạng một `Intent` (ví dụ, bạn có thể tạo một intent cho phép người dùng chọn một thông tin liên lạc cá nhân và trả về intent này cho bạn - intent được trả về sẽ bao gồm cả URI chỉ tới số liên lạc trong danh bạ đã chọn).

Đối với các broadcast receiver, intent chỉ đơn thuần định nghĩa thông báo đang được phát đi (ví dụ, một broadcast cho thấy rằng pin sắp hết chỉ bao gồm một chuỗi nội dung là "battery is low" - có nghĩa là "pin yếu").

Một thành phần khác không được các intent kích hoạt là Content Provider. Thay vào đó, Content Provider được kích hoạt khi là thành phần đích của một yêu cầu từ một `ContentResolver`. Trình xử lý nội dung (content resolver) xử lý tất cả các giao dịch trực tiếp với Content Provider, nên thành phần đang thực hiện giao dịch với content provider không phải làm công việc trên mà thay vào đó sẽ gọi các phương thức của đối tượng `ContentResolver`. Điều này sinh ra một tầng (layer) trừu tượng giữa Content Provider và thành phần yêu cầu thông tin (cho mục đích bảo mật).

Một số phương thức dùng để kích hoạt từng loại thành phần:

- Bạn có thể khởi động một activity (hoặc cung cấp cho nó một action để thực thi) bằng cách truyền một `Intent` tới phương thức `startActivity()` hoặc `startActivityForResult()` (khi bạn muốn activity trả về kết quả).
- Bạn có thể khởi động một service (hoặc đưa ra chỉ thị mới tới một service đang chạy) bằng cách truyền một `Intent` tới phương thức `startService()`. Hoặc, bạn có thể gắn vào một service bằng cách truyền một `Intent` tới phương thức `bindService()`.
- Bạn có thể khởi tạo một broadcast bằng cách truyền một `Intent` tới các phương thức như `sendBroadcast()`, `sendOrderedBroadcast()` hoặc `sendStickyBroadcast()`.

- Bạn có thể thực hiện một truy vấn tới Content Provider bằng cách gọi phương thức `query()` trên một `ContentResolver`.

Để biết thêm thông tin về cách sử dụng intent, xem tài liệu "Intents and Intent Filters" ("Intent và bộ lọc intent"). Thông tin tham khảo về việc kích hoạt các thành phần cụ thể cũng được cung cấp trong những tài liệu sau: Activity, Service, Broadcast Receiver và Content Providers.

File kê khai

Trước khi hệ thống Android có thể khởi động một thành phần của ứng dụng, hệ thống phải biết thành phần đó đã tồn tại hay chưa bằng cách đọc file `AndroidManifest.xml` của ứng dụng (file "kê khai"). Ứng dụng của bạn phải khai báo tất cả các thành phần của nó trong file kê khai và file này phải lưu ở thư mục gốc (root) của project (dự án) ứng dụng.

Ngoài việc khai báo các thành phần của ứng dụng, file kê khai có thể thực hiện một số việc khác như:

- Xác định quyền người dùng mà ứng dụng đòi hỏi, như quyền truy cập Internet hoặc quyền đọc thông tin liên lạc trong danh bạ của người dùng.
- Khai báo API Level (Cấp API) tối thiểu cần thiết cho ứng dụng, dựa vào việc ứng dụng sử dụng những API nào.
- Khai báo các tính năng phần cứng và phần mềm được ứng dụng yêu cầu hoặc sử dụng, chẳng hạn như camera, dịch vụ bluetooth hay màn hình cảm ứng đa điểm (multitouch screen).
- Các thư viện API mà ứng dụng cần liên kết tới (ngoài các API trong framework Android), chẳng hạn như Google Maps library (thư viện Google Maps).
- Và còn nhiều việc khác nữa.

Khai báo các thành phần

Nhiệm vụ chính của file kê khai là thông báo cho hệ thống về các thành phần của ứng dụng. Ví dụ, một file kê khai có thể khai báo một activity như sau:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

Trong phần tử `<application>`, thuộc tính `android:icon` chỉ tới những tài nguyên của biểu tượng nhận diện ứng dụng.

Trong phần tử <activity>, thuộc tính android:name xác định tên lớp đầy đủ của lớp con Activity, còn các thuộc tính android:label chỉ định một chuỗi để sử dụng như là nhãn mà người dùng có thể thấy được (user-visible label) cho activity.

Bạn cần khai báo tất cả các thành phần của ứng dụng theo cách sau:

- Các phần tử <activity> cho activity.
- Các phần tử <service> cho service.
- Các phần tử <receiver> cho broadcast receiver.
- Các phần tử <provider> cho content provider.

Các activity, service và Content Provider mà bạn bao gồm trong mã nguồn nhưng không khai báo trong file kê khai sẽ không được hệ thống nhìn thấy, nên có thể sẽ không bao giờ chạy. Tuy nhiên, các broadcast receiver có thể được khai báo trong file kê khai hoặc được tạo động trong mã (như các đối tượng BroadcastReceiver) và được đăng ký với hệ thống bằng cách gọi phương thức registerReceiver().

Để biết thêm thông tin về cấu trúc của file kê khai, xem tài liệu "File AndroidManifest.xml".

Khai báo khả năng của thành phần

Như đã thảo luận từ trước, trong mục "Kích hoạt các thành phần", bạn có thể sử dụng một Intent để khởi động activity, service và broadcast receiver. Bạn có thể làm điều này bằng cách đặt tên một cách tường minh cho thành phần đích (sử dụng tên lớp thành phần) trong intent. Tuy nhiên, sức mạnh thực sự của các intent nằm bên trong khái niệm về những action của chúng. Bạn chỉ việc mô tả loại của action mình muốn thực hiện (và tùy chọn dữ liệu khi bạn muốn thực hiện), đồng thời cho phép hệ thống tìm một thành phần trên thiết bị có thể thực thi action rồi khởi động nó. Nếu có nhiều thành phần có khả năng thực hiện action do intent mô tả, người dùng sẽ chọn một thành phần cần sử dụng.

Cách thức mà hệ thống nhận diện những thành phần có thể phản hồi lại một intent là so sánh *intent* nhận được với các *bộ lọc intent* cung cấp trong file kê khai của những ứng dụng khác trên thiết bị.

Khi khai báo một thành phần trong file kê khai của ứng dụng, bạn có thể tùy chọn bao gồm các bộ lọc intent nhằm khai báo những khả năng của thành phần để nó có thể phản hồi lại các intent từ những ứng dụng khác. Bạn có thể khai báo một bộ lọc intent cho thành phần bằng cách thêm vào một phần tử <intent-filter> đóng vai trò làm con của phần tử khai báo thành phần đó.

Ví dụ, một ứng dụng e-mail chứa một activity dùng để soạn thảo e-mail mới có thể khai báo một bộ lọc intent trong file kê khai của nó để phản hồi các intent "send" (gửi) (để gửi e-mail đi). Sau đó, một activity trong ứng dụng của bạn có thể tạo ra một intent với action "gửi" (ACTION_SEND), sao cho hệ thống khớp với activity "gửi" của ứng dụng e-mail và khởi động nó khi bạn kích hoạt intent thông qua phương thức startActivity().

Để biết thêm thông tin về cách tạo các bộ lọc intent, xem tài liệu "Intents and Intent Filters" ("Intent và bộ lọc intent").

Khai báo các yêu cầu của ứng dụng

Có rất nhiều thiết bị vận hành bằng Android và không phải tất cả trong số chúng đều cung cấp các tính năng và khả năng như nhau. Để tránh cho ứng dụng của bạn khỏi bị cài đặt trên các thiết bị thiếu tính năng cần thiết để vận hành, một điều quan trọng là bạn phải định nghĩa rõ ràng một profile cho những loại thiết bị mà ứng dụng của mình hỗ trợ, bằng cách khai báo thiết bị cùng các yêu cầu phần mềm trong file kê khai. Hầu hết những khai báo này đều chỉ mang tính thông tin và hệ thống không đọc chúng, song các service ngoài như Google Play lại đọc chúng để cung cấp tính năng lọc tìm kiếm ứng dụng từ thiết bị của họ.

Ví dụ, nếu ứng dụng của bạn yêu cầu một camera và sử dụng những API được giới thiệu trong Android 2.1 (`API Cấp 7`), bạn nên khai báo những yêu cầu này trong file kê khai. Theo cách này, những thiết bị *không* có camera và có phiên bản Android *thấp hơn* bản 2.1 sẽ không thể cài đặt ứng dụng của bạn từ Google Play.

Tuy nhiên, bạn cũng có thể khai báo rằng ứng dụng của mình sử dụng camera, nhưng điều này là không *bắt buộc*. Với trường hợp này, ứng dụng của bạn phải thực hiện một thao tác kiểm tra trong thời gian chạy để xác định xem thiết bị có camera không và tắt các tính năng sử dụng camera đi nếu camera không sẵn sàng.

Sau đây là một số đặc tính quan trọng của thiết bị mà bạn nên xem xét khi thiết kế và phát triển ứng dụng của mình.

Kích thước màn hình (screen size) và mật độ (density)

> Để phân loại thiết bị theo kích thước màn hình, Android định nghĩa hai đặc tính cho mỗi thiết bị: Kích thước màn hình (kích thước vật lý của màn hình) và mật độ màn hình (mật độ các pixel - điểm ảnh trên màn hình, hay còn gọi là dpi - dots per inch). Để đơn giản hóa tất cả các dạng khác nhau của cấu hình màn hình, hệ thống Android phân chúng thành các nhóm để dễ dàng lựa chọn hơn.
>
> Các kích thước màn hình là: Nhỏ (small), vừa (normal), lớn (large) và rất lớn (extra large).
>
> Các mật độ màn hình là: Thấp (low), vừa (medium), cao (high) và rất cao (extra high).
>
> Theo mặc định, ứng dụng của bạn sẽ tương thích với tất cả các kích thước và mật độ màn hình, vì hệ thống Android sẽ thực hiện tinh chỉnh phù hợp với layout của giao diện người dùng và tài nguyên ảnh. Tuy nhiên, bạn nên tạo ra các layout (bố cục) chuyên biệt cho một số kích thước màn hình nhất định và cung cấp các ảnh chuyên dụng cho những mật độ nhất định, sử dụng các tài nguyên layout thay thế, và bằng cách khai báo chính xác trong file kê khai về kích thước màn hình được ứng dụng của bạn hỗ trợ với phần tử `<supports-screens>`.
>
> Để tham khảo thêm thông tin, xem tài liệu "Supporting Multiple Screens" ("Hỗ trợ đa màn hình").

Cấu hình đầu vào

> Nhiều thiết bị cung cấp một loại cơ chế nhập đầu vào khác biệt, chẳng hạn như bàn phím phần cứng (hardware keyboard), một bi lăn (trackball), hoặc bàn di chuột 5 chiều (five-way navigation pad). Nếu ứng dụng của bạn đòi hỏi một phần cứng cụ thể dùng để nhập đầu vào, bạn nên khai báo nó trong file kê khai với phần tử

`<uses-configuration>`. Tuy nhiên, rất hiếm khi một ứng dụng yêu cầu cấu hình nhập như vậy.

Các đặc điểm của thiết bị

Có nhiều đặc điểm phần cứng và phần mềm có thể hoặc không thể tồn tại trên một thiết bị chạy Android, chẳng hạn camera, cảm biến ánh sáng (light sensor), bluetooth, một phiên bản nhất định của OpenGL, hoặc màn hình cảm ứng trung thực. Bạn không nên cho rằng một đặc điểm nhất định là có sẵn trên mọi thiết bị chạy Android (ngoài thư viện chuẩn của Android). Do đó, bạn nên khai báo bất kỳ đặc điểm nào được ứng dụng của mình sử dụng với phần tử `<uses-feature>`.

Phiên bản nền tảng

Các thiết bị chạy Android thường chạy những phiên bản khác nhau của nền tảng Android, chẳng hạn như Android 1.6 hay Android 2.3. Mỗi phiên bản thành công thường bao gồm thêm các API mà chưa tồn tại trong những phiên bản trước đó. Để chỉ ra rằng tập các API đã sẵn sàng, mỗi phiên bản nền tảng xác định một Cấp API (ví dụ như Android 1.0 là API Cấp 1 (API Level 1) và Android 2.3 là API Cấp 9). Nếu sử dụng bất cứ API nào được thêm vào nền tảng sau phiên bản 1.0, bạn nên khai báo Cấp API tối thiểu có chứa những API đó bằng cách sử dụng phần tử `<uses-sdk>`.

Điều quan trọng là bạn cần khai báo tất cả các yêu cầu đó cho ứng dụng của mình, bởi vì khi bạn phân phối ứng dụng trên Google Play, cửa hàng sẽ sử dụng những khai báo này để lọc các ứng dụng phù hợp cho mỗi thiết bị. Như vậy, ứng dụng của bạn chỉ nên hiển thị sẵn sàng đối với những thiết bị đáp ứng được mọi yêu cầu mà nó đưa ra.

Để biết thêm thông tin về cách Google Play lọc các ứng dụng dựa trên những yêu cầu trên (hoặc các yêu cầu khác), xem tài liệu "Lọc trên Google Play" ("Filters on Google Play").

Tài nguyên của ứng dụng

Một ứng dụng Android không chỉ bao gồm mã nguồn - nó đòi hỏi những tài nguyên khác nằm tách biệt với mã nguồn (source code), chẳng hạn như hình ảnh, file âm thanh và bất cứ thứ gì liên quan tới hình thức trình bày trực quan của ứng dụng. Ví dụ, bạn nên định nghĩa các hoạt cảnh (animation), menu, style, màu sắc và layout của các giao diện người dùng activity với file XML. Sử dụng tài nguyên của ứng dụng làm cho việc cập nhật nhiều đặc điểm khác nhau trong ứng dụng của bạn trở nên dễ dàng mà không phải chỉnh sửa mã nguồn. Bên cạnh đó, bằng cách cung cấp tập các tài nguyên thay thế, bạn có thể tối ưu hóa ứng dụng của mình cho nhiều loại cấu hình thiết bị (chẳng hạn như những ngôn ngữ và kích thước màn hình khác nhau).

Đối với mọi tài nguyên bao gồm trong dự án Android, các công cụ SDK (SDK build toos) xác định một ID duy nhất có dạng số nguyên (integer), và bạn có thể dùng ID này để tham chiếu tới tài nguyên từ mã nguồn của ứng dụng hoặc từ các tài nguyên khác được định nghĩa trong XML. Ví dụ, nếu ứng dụng của bạn chứa một file ảnh tên là `logo.png` (lưu trong thư mục `res/drawable/directory`), các công cụ SDK sẽ tạo một ID tài nguyên có tên là `R.drawable.logo`, để bạn có thể dùng cho việc tham chiếu tới file ảnh và chèn nó vào giao diện người dùng của mình.

Một trong những khía cạnh quan trọng nhất của việc cung cấp tài nguyên tách biệt khỏi mã nguồn là bạn có thể cung cấp các tài nguyên thay thế cho những cấu hình thiết bị khác. Ví dụ, bằng việc định nghĩa các chuỗi (từ ,cụm từ, hoặc câu) xuất hiện trên giao diện người dùng (UI) trong XML, bạn có thể dịch (translate) những chuỗi này sang loại ngôn ngữ khác và lưu chúng trong các file riêng biệt. Sau đó, dựa vào một *bộ ánh xạ* ngôn ngữ (language *qualifier*) mà bạn thêm vào ở tên thư mục tài nguyên (chẳng hạn như `res/values-fr/` đối với các giá trị chuỗi tiếng Pháp) và thiết lập ngôn ngữ người dùng, hệ thống Android sẽ áp dụng những chuỗi ngôn ngữ phù hợp cho giao diện người dùng của bạn.

Android hỗ trợ nhiều *bộ ánh xạ* khác nhau cho các tài nguyên thay thế của bạn. *Bộ ánh xạ* là một chuỗi ngắn được bao gồm trong tên của thư mục tài nguyên, xác định loại cấu hình thiết bị mà tại đó, những tài nguyên trên nên được sử dụng. Một ví dụ khác, bạn nên thường xuyên tạo các layout khác nhau cho những activity của mình, dựa vào vào chiều xoay và kích thước màn hình của thiết bị. Ví dụ, khi màn hình thiết bị đang ở dạng đứng (portrait orientation, còn gọi là tall), bạn muốn một layout với những button nằm dọc. Tuy nhiên, khi màn hình ở dạng nằm ngang (landscapte orientation, còn gọi là wide), các button nên được căn chỉnh theo chiều ngang. Để thay đổi layout dựa trên chiều xoay màn hình, bạn có thể định nghĩa hai layout khác nhau và áp dụng bộ ánh xạ thích hợp đối với từng tên thư mục của layout. Sau đó, hệ thống sẽ tự động áp dụng layout thích hợp dựa vào chiều xoay hiện tại của thiết bị.

Để biết thêm thông tin về những loại tài nguyên khác nhau mà bạn có thể bao gồm trong ứng dụng của mình và cách tạo các tài nguyên thay thế cho nhiều loại cấu hình thiết bị, xem hướng dẫn dành cho các nhà phát triển tại mục “Tài nguyên của ứng dụng”.

4. File AndroidManifest.xml

Mọi ứng dụng đều phải có một file `AndroidManifest.xml` (với chính xác tên gọi đó) trong thư mục gốc của nó. File kê khai này trình bày những thông tin cần thiết về ứng dụng cho hệ thống Android, đó là thông tin mà hệ thống bắt buộc phải có trước khi nó có thể chạy bất cứ dòng mã nguồn nào của ứng dụng. Một số nhiệm vụ mà file kê khai có thể thực hiện bao gồm:

- Xác định tên gói Java cho ứng dụng. Tên gói đóng vai trò là như một định danh (identifier) duy nhất của ứng dụng.
- Mô tả các thành phần của ứng dụng - activity, service, broadcast receiver và Content Provider cấu thành nên ứng dụng. File `AndroidManifest.xml` chỉ ra tên các lớp thực thi mỗi thành phần và đưa ra những khả năng của chúng (ví dụ, các loại thông điệp `Intent` mà chúng có thể xử lý). Những khai báo như vậy cho hệ thống Android biết các thành phần nào được khởi động và chúng được thực thi dưới những điều kiện nào.
- Quyết định những tiến trình nào sẽ xử lý các thành phần của ứng dụng.
- Khai báo các quyền mà ứng dụng bắt buộc phải có để có thể truy cập vào các phần được bảo vệ của API và tương tác với những ứng dụng khác.
- Khai báo những quyền mà các thành phần khác cần có để có thể tương tác với những thành phần của ứng dụng.

- Liệt kê các lớp `Instrumentation` cung cấp profile cùng các thông tin khác khi ứng dụng đang chạy. Những khai báo này chỉ tồn tại trong file kê khai trong khi ứng dụng đang được phát triển và kiểm thử; chúng bị loại bỏ trước khi ứng dụng được phát hành.
- Khai báo cấp API tối thiểu mà ứng dụng yêu cầu.
- Liệt kê những thư viện mà ứng dụng phải liên kết tới.

Cấu trúc của file kê khai

Hình bên dưới thể hiện cấu trúc chung của file kê khai và mọi phần tử nó có thể chứa. Mỗi phần tử, cùng với mọi thuộc tính của nó, được ghi lại đầy đủ trong một file riêng biệt. Để xem thông tin chi tiết về bất cứ phần tử nào, hãy nhấn chuột vào tên phần tử trong biểu đồ, trong danh sách các phần tử theo trình tự ABC trên sơ đồ hoặc trên bất cứ chỗ nào khác đề cập đến tên phần tử.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
```

```
        <service>
           <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Tất cả các phần tử có thể xuất hiện trong file kê khai đều được liệt kê trong danh sách bên dưới theo trình tự ABC. Đây chỉ là những phần tử hợp lệ; bạn không thể tự thêm các phần tử hoặc thuộc tính của riêng mình được.

```
<action>
<activity>
<activity-alias>
<application>
<category>
<data>
<grant-uri-permission>
<instrumentation>
<intent-filter>
<manifest>
<meta-data>
<permission>
<permission-group>
<permission-tree>
<provider>
<receiver>
<service>
```

```
<supports-screens>
<uses-configuration>
<uses-feature>
<uses-library>
<uses-permission>
<uses-sdk>
```

Các quy ước về file

Một số quy ước và quy tắc áp dụng chung cho tất cả các phần tử cũng như thuộc tính trong file kê khai:

Các phần tử

Chỉ có các phần tử `<manifest>` và `<application>` mới là phần tử bắt buộc, chúng phải tồn tại và có thể chỉ duy nhất một lần. Hầu hết các phần tử khác đều có thể xuất hiện nhiều lần hoặc không - mặc dù ít nhất một vài trong số chúng phải có mặt trong file kê khai để có nghĩa.

Nếu một phần tử chứa bất kỳ thứ gì, nó sẽ chứa cả những phần tử khác. Mọi giá trị đều được thiết lập thông qua thuộc tính, chứ không phải dưới dạng dữ liệu trong một phần tử.

Các phần tử đồng cấp thường không có cùng thứ tự. Ví dụ, các phần tử `<activity>`, `<provider>` và `<service>` có thể được trộn lẫn theo bất cứ trật tự nào. (Phần tử `<activity-alias>` là ngoại lệ của nguyên tắc này: Nó phải theo sau `<activity>` mà nó là bí danh (alias)).

Các thuộc tính

Theo nghĩa chính thức thì tất cả các thuộc tính là tùy chọn (không bắt buộc). Tuy nhiên, có một số thuộc tính phải được chỉ định cho một phần tử để phục vụ mục đích nào đó. Hãy sử dụng tài liệu này như một hướng dẫn. Đối với những thuộc tính thực sự không bắt buộc, tài liệu này đề cập tới một giá trị mặc định hoặc trạng thái xảy ra khi không có sự ấn định giá trị nào.

Ngoại trừ một số thuộc tính của phần tử gốc `<manifest>`, còn lại mọi tên thuộc tính đều bắt đầu với tiền tố `android:` - ví dụ như `android:alwaysRetainTaskState`. Do tiền tố mang tính toàn cục, nên các tài liệu thường bỏ qua nó khi tham chiếu đến các thuộc tính qua tên của chúng.

Khai báo tên lớp

Nhiều phần tử tương ứng với các đối tượng Java, bao gồm các phần tử cho bản thân ứng dụng (phần tử `<application>`) và những thành phần chính - activitiy (`<activity>`), service (`<service>`), broadcast receiver (`<receiver>`), Content Provider (`<provider>`).

Nếu bạn định nghĩa một lớp con, hầu hết là cho các lớp thành phần (`Activity`, `Service`, `BroadcastReceiver` và `ContentProvider`), lớp con sẽ được

khai báo thông qua thuộc tính `name`. Tên phải bao gồm tên đầy đủ của gói. Ví dụ, một lớp con `Service` có thể được khai báo như sau:

```
<manifest . . . >
    <application . . . >
       <service android:name="com.example.project.SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

Tuy nhiên, có một cách viết giản lược, nếu ký tự đầu tiên của chuỗi là một dấu chấm, chuỗi sẽ được thêm vào tên gói của ứng dụng (được chỉ định trong thuộc tính `package` của phần tử `<manifest>`). Đoạn mã dưới đây tương tự như bên trên:

```
<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

Khi khởi động một thành phần, Android tạo một thể hiện của lớp con đã được xác định tên. Nếu không lớp con nào được chỉ định, Android sẽ tạo một thể hiện của lớp nền tảng (base class).

Nhiều giá trị

Nếu có nhiều giá trị được ấn định, xu hướng phổ biến là phần tử sẽ được lặp lại thay vì liệt kê nhiều giá trị đó trong một phần tử đơn. Ví dụ, một bộ lọc intent có thể liệt kê vài action:

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

Giá trị tài nguyên

Một số thuộc tính có giá trị có thể được hiển thị đối với người dùng - ví dụ, label (nhãn) và icon (biểu tượng) cho một activity. Các giá trị của những thuộc tính này

nên được định vị từ một tài nguyên hoặc theme. Các giá trị tài nguyên được thể hiện dưới định dạng sau:

```
@[package:]type:name
```

trong đó, tên package (gói) có thể bị bỏ qua nếu tài nguyên nằm cùng gói với ứng dụng, `type` là loại tài nguyên - chẳng hạn như "`string`" hay "`drawable`" - và `name` là tên định danh tài nguyên cụ thể. Ví dụ:

```
<activity android:icon="@drawable/smallPic" . . . >
```

Giá trị lấy từ theme được biểu diễn tương tự, nhưng bắt đầu với '`?`' chứ không phải '`@`':

```
?[package:]type:name
```

Giá trị chuỗi

Trong khi một giá trị thuộc tính là một chuỗi, hai dấu chéo ngược ('`\\`') phải được dùng cho các ký tự thoát (escape character) - ví dụ, '`\\n`' cho một dòng mới hoặc '`\\uxxxx`' cho một ký tự Unicode.

Các tính năng của file

Mục dưới đây sẽ mô tả cách một số tính năng của Android được phản ánh trong file kê khai.

Bộ lọc intent

Các thành phần cốt lõi của một ứng dụng (activity, service và broadcast receiver) đều được intent kích hoạt. Intent là một bundle thông tin (đối tượng `Intent`) mô tả một action mong muốn - bao gồm dữ liệu cần giải quyết, hạng mục của thành phần nên thực hiện action cùng những chỉ dẫn khác. Android định vị một thành phần thích hợp để phản hồi intent, khởi động một thể hiện mới của thành phần nếu cần thiết và truyền cho nó một đối tượng Intent.

Các thành phần chỉ ra khả năng của chúng - những loại intent mà chúng có thể phản hồi - thông qua các *bộ lọc intent*. Do hệ thống Android phải biết những intent nào mà một thành phần có thể xử lý trước khi khởi động một thành phần, nên các bộ lọc intent sẽ được xác định trong file kê khai dưới dạng các phần tử `<intent-filter>`. Một thành phần có thể có số lượng bộ lọc tùy ý, nhưng mỗi bộ lọc lại mô tả một khả năng khác nhau.

Một intent chỉ định rõ ràng tên của thành phần đích sẽ kích hoạt thành phần đó; bộ lọc không làm việc này. Tuy nhiên, một intent không chỉ định cụ thể tên thành phần đích lại có thể kích hoạt một thành phần chỉ khi intent có thể vượt qua một trong những bộ lọc của thành phần ấy.

Để biết thêm thông tin về cách các đối tượng Intent được kiểm thử qua bộ lọc intent, xem tài liệu "Intents and Intent Filters" ("Intent và bộ lọc intent") .

Nhãn và biểu tượng

Một số phần tử có thuộc tính icon và label để hiển thị một biểu tượng nhỏ và một nhãn chữ cho người dùng. Một số khác còn có thuộc tính description để cung cấp dòng mô tả dài hơn và cũng có thể được hiển thị trên màn hình. Ví dụ, phần tử <permission> có tất cả ba thuộc tính, nên khi người dùng được yêu cầu cấp quyền cho một ứng dụng thì một biểu tượng đại diện cho quyền, tên của quyền và một mô tả về được hiển thị cho người dùng.

Trong mọi trường hợp, biểu tượng và nhãn thiết lập trong phần tử chứa nó trở thành thiết lập biểu tượng và nhãn mặc định cho tất cả các phần tử con. Do vậy, biểu tượng và nhãn đặt trong phần tử <application> là biểu tượng và nhãn mặc định cho mỗi thành phần của ứng dụng. Tương tự, biểu tượng và nhãn thiết lập cho một thành phần - ví dụ, phần tử <activity> - là thiết lập mặc định cho từng phần tử <intent-filter> của thành phần đó. Nếu phần tử <application> đặt một nhãn nhưng một activity và bộ lọc intent của nó thì không, nhãn của ứng dụng sẽ được coi là nhãn chung cho cả activity lẫn bộ lọc intent.

Biểu tượng và nhãn thiết lập cho một bộ lọc intent được dùng để đại diện cho một thành phần khi thành phần đó thể hiện tới người dùng nhằm mục đích hoàn thành chức năng được khai báo của bộ lọc. Ví dụ, một bộ lọc với thiết lập "`android.intent.action.MAIN`" và "`android.intent.category.LAUNCHER`" nhận một activity như là nhân tố khởi tạo ứng dụng - tức là, nhân tố nên được hiển thị trong màn hình chính của ứng dụng. Do đó, biểu tượng và nhãn thiết lập trong bộ lọc sẽ được hiển thị trong màn hình chính.

Quyền

Quyền (permission) là khả năng hạn chế truy cập vào một phần mã nguồn hoặc dữ liệu trên thiết bị. Sự hạn chế được áp đặt để bảo vệ dữ liệu và mã nguồn quan trọng khỏi việc bị lạm dụng để bóp méo hoặc gây hậu quả tới trải nghiệm người dùng.

Mỗi quyền được xác định bằng một nhãn duy nhất. Thường thì nhãn chỉ ra action bị hạn chế. Ví dụ, dưới đây là một số quyền định nghĩa bởi Android:

```
android.permission.CALL_EMERGENCY_NUMBERS
android.permission.READ_OWNER_DATA
android.permission.SET_WALLPAPER
android.permission.DEVICE_POWER
```

Một tính năng có thể được nhiều nhất một quyền bảo vệ.

Nếu một ứng dụng cần truy cập tới tính năng được một quyền bảo vệ, ứng dụng cần phải khai báo rằng nó cần quyền đó bằng cách sử dụng phần tử <uses-permission> trong file kê khai. Sau đó, khi ứng dụng được cài lên thiết bị, trình cài đặt (installer) sẽ quyết định xem có cấp quyền mà ứng dụng yêu cầu hay không bằng cách kiểm tra cơ quan cấp chứng thực cho ứng dụng, trong một số trường hợp sẽ hỏi người dùng. Nếu quyền được cấp, ứng dụng có thể sử dụng các tính năng được bảo vệ. Nếu không, việc cố gắng truy cập vào những tính năng này sẽ thất bại mà không có một thông báo nào cho người dùng.

Một ứng dụng cũng có thể bảo vệ các thành phần của nó (activity, service, broadcast receiver và Content Provider) với quyền. Ứng dụng có thể sử dụng bất cứ quyền nào do Android định nghĩa (được liệt kê trong `android.Manifest.permission`), hoặc do những ứng dụng khác khai báo, hoặc tự định nghĩa. Một quyền mới được khai báo với phần tử `<permission>`. Ví dụ, một activity có thể được bảo vệ như sau:

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />
     . . .
     <application . . .>
        <activity android:name="com.example.project.FreneticActivity"
                  android:permission="com.example.project.DEBIT_ACCT"
                   . . . >
             . . .
         </activity>
     </application>
</manifest>
```

Lưu ý, trong ví dụ này, quyền `DEBIT_ACCT` không chỉ được khai báo với phần tử `<permission>` mà còn được yêu cầu với phần tử `<uses-permission>`. Lý do là để các thành phần khác của ứng dụng có thể khởi động activity được bảo vệ, cho dù sự bảo vệ được áp dụng bởi chính bản thân ứng dụng.

Trong cùng ví dụ trên, nếu thuộc tính permission được đặt cho một quyền đã được khai báo ở đâu đó (chẳng hạn như `android.permission.CALL_EMERGENCY_NUMBERS`), thì không nhất thiết phải khai báo lại với phần tử `<permission>`. Tuy nhiên, vẫn cần phải khai báo quyền đó với `<uses-permission>`.

Phần tử `<permission-tree>` khai báo một namespace cho một nhóm quyền sẽ được định nghĩa trong mã nguồn. Và `<permission-group>` xác định một nhãn cho tập các quyền (bao gồm cả những quyền được khai báo trong file kê khai với các phần tử `<permission>` và những quyền được khai báo ở đâu đó khác). Phần tử này chỉ ảnh hưởng tới cách các quyền được nhóm lại khi hiển thị tới người dùng. Phần tử `<permission-group>` không xác định quyền nào thuộc về nhóm nào mà chỉ cấp cho nhóm một cái tên. Một quyền được đặt trong nhóm bằng cách gán tên nhóm vào thuộc tính `permissionGroup` của phần tử `<permission>`.

Các thư viện

Mỗi ứng dụng được liên kết tới thư viện mặc định của Android, bao gồm các gói cơ bản để xây dựng ứng dụng (với những lớp thông dụng như `Activity`, `Service`, `Intent`, `View`, `Button`, `Application`, `ContentProvider`,...).

Tuy nhiên, một số gói lại nằm trong thư viện riêng của chúng. Nếu ứng dụng của bạn dùng mã từ bất kỳ gói nào trong số những gói này, ứng dụng phải được liên kết rõ ràng tới chúng. File kê khai phải chứa một phần tử `<uses-library>` riêng biệt để đặt tên cho mỗi thư viện. (Có thể tìm thấy tên thư viện trong tài liệu của gói đó).

5. Giao diện người dùng trên mobile

Giao diện người dùng (User Interface - UI) của ứng dụng là tất cả những gì người dùng có thể nhìn thấy và tương tác được. Android cung cấp nhiều thành phần giao diện người dùng được dựng sẵn (pre-build UI), chẳng hạn như các đối tượng layout có cấu trúc và những điều khiển (control) cho phép bạn xây dựng giao diện đồ họa người dùng (Graphical User Interface - GUI) cho ứng dụng của mình. Ngoài ra, Android còn cung cấp những module UI khác cho các giao diện đặc biệt như hộp thoại (dialog), thông báo (notification) và menu.

5.1 Tổng quan về giao diện người dùng

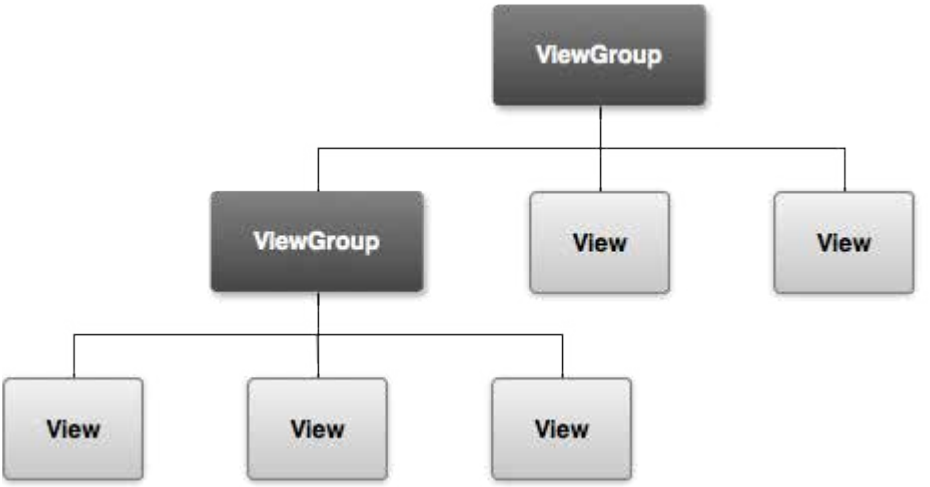
Mọi phần tử giao diện người dùng trong ứng dụng Android đều được xây dựng bằng cách sử dụng các đối tượng `View` và `ViewGroup`. `View` là đối tượng vẽ ra thứ gì đó trên màn hình sao cho người dùng có thể tương tác được. `ViewGroup` là đối tượng chứa các đối tượng `View` (và `ViewGroup`) khác để định nghĩa layout của giao diện.

Android cung cấp một tập hợp (collection) gồm cả các lớp con của `View` lẫn `ViewGroup`, nhằm cung cấp cho người dùng những điều khiển đầu vào thông dụng (chẳng hạn như các button và trường văn bản cùng những dạng layout khác nhau (ví dụ như layout tuyến tính hay layout tương đối).

Layout của giao diện người dùng

Giao diện người dùng của mỗi thành phần trong ứng dụng được định nghĩa bằng cách sử dụng một cây phân cấp các đối tượng `View` và `ViewGroup`, như minh họa ở Hình 1. Mỗi nhóm view là một đối tượng chứa vô hình (invisible container) bố trí các

view con, trong khi view con có thể là những điều khiển đầu vào hoặc các widget[2] khác tạo thành một số phần của giao diện người dùng. Cây phân cấp có thể ở dạng đơn giản hoặc phức tạp, tuy theo nhu cầu (nhưng sự đơn giản vẫn là cách tốt nhất để gia tăng hiệu năng).



Hình 1. Minh họa một cây phân cấp view, xác định một layout giao diện người dùng.

Để khai báo layout, bạn có thể khởi tạo các đối tượng `View` trong mã nguồn và bắt đầu xây dựng một cây phân cấp, nhưng cách dễ dàng cũng như hiệu quả nhất là định nghĩa layout với một file XML. XML cung cấp cho layout một cấu trúc mà con người có khả năng đọc hiểu được, tương tự như HTML.

Tên một phần tử XML cho một view sẽ tương ứng với lớp Android mà view này đại diện. Do đó, một phần tử `<TextView>` sẽ tạo widget `TextView` trong giao diện người dùng, còn một phần tử `<LinearLayout>` tạo một view group `LinearLayout`.

Ví dụ, một layout theo chiều dọc đơn giản với text view (view dạng văn bản) và một button trông sẽ như dưới đây:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Đây là một TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Đây là một Button" />
</LinearLayout>
```

[2] Là các tiện ích nhỏ, thường dùng để cung cấp thông tin hoặc tương tác với user dạng đơn giản khi user không muốn mất thời gian vào phần mềm chính.

Khi bạn nạp một tài nguyên layout trong ứng dụng, Android sẽ khởi tạo mỗi nút (node) của layout đó thành một đối tượng đang trong thời gian chạy, bạn có thể dùng đối tượng để định nghĩa thêm các hành vi, truy vấn trạng thái đối tượng hoặc chỉnh sửa layout.

Để xem hướng dẫn hoàn chỉnh về cách tạo một layout giao diện người dùng, mời bạn tham khảo XML Layouts.

Các thành phần của giao diện người dùng

Bạn không nhất thiết phải xây dựng tất cả giao diện bằng các đối tượng `View` và `ViewGroup`. Android cung cấp vài thành phần ứng dụng đưa ra một chuẩn layout giao diện người dùng mà bạn chỉ cần làm một việc đơn giản là định nghĩa nội dung. Mỗi thành phần giao diện người dùng như vậy đều có một tập các API riêng được mô tả trong tài liệu tương ứng, chẳng hạn như `Action Bar`, `Dialogs` và `Status Notifications`.

5.2 Layout

Layout định nghĩa cấu trúc trực quan cho giao diện người dùng, chẳng hạn như giao diện người dùng cho một activity hoặc widget của ứng dụng (app widget). Bạn có thể khai báo layout theo hai cách:

- **Khai báo các phần tử giao diện người dùng trong XML**. Android cung cấp một bộ từ vựng XML đơn giản tương ứng với các lớp View và lớp con, ví dụ như cho widget và layout.
- **Khởi tạo các phần tử layout trong thời điểm chạy**. Ứng dụng của bạn có thể tạo các đối tượng View cũng như ViewGroup (và thao tác với các thuộc tính của chúng) hoàn toàn bằng việc lập trình.

Framework Android cho bạn khả năng sử dụng linh hoạt một hoặc cả hai phương thức này để khai báo và quản lý giao diện người dùng của ứng dụng. Ví dụ, bạn có thể khai báo layout mặc định cho ứng dụng trong XML, bao gồm những phần tử màn hình sẽ xuất hiện trong đó và các thuộc tính của chúng. Sau đó, bạn có thể thêm mã trong ứng dụng để thay đổi trạng thái của các đối tượng màn hình, bao gồm những đối tượng được khai báo trong XML tại thời điểm chạy ứng dụng.

Ưu điểm của việc khai báo giao diện người dùng trong XML là cho phép bạn tách biệt tốt hơn mã trình bày (giao diện) của ứng dụng khỏi phần mã điều khiển hành vi của nó. Các mô tả giao diện người dùng được đặt bên ngoài mã nguồn, có nghĩa là bạn có thể sửa đổi hoặc tùy chỉnh giao diện mà không phải sửa mã nguồn và biên dịch lại (recompile). Ví dụ, bạn có thể tạo layout XML cho các chiều xoay màn hình khác nhau, kích thước màn hình khác nhau và ngôn ngữ khác nhau. Ngoài ra, việc khai báo layout trong XML giúp cho việc hình dung cấu trúc của giao diện người dùng trở nên dễ dàng hơn, dẫn tới việc gỡ lỗi (debug) cũng dễ hơn. Như vậy, tài liệu này tập trung vào việc hướng dẫn bạn cách khai báo layout trong XML. Nếu bạn quan tâm đến cách khởi tạo các đối tượng View vào thời điểm chạy, hãy tham khảo lớp `ViewGroup` và `View`.

Thông thường, bộ từ vựng XML để khai báo các phần tử cho giao diện người dùng sẽ tuân theo cấu trúc cũng như cách đặt tên của lớp và phương thức, trong đó tên phần tử tương ứng với tên lớp và tên thuộc tính tương ứng với phương thức. Thực tế, sự tương ứng xảy ra thường xuyên, nên bạn có thể đoán thuộc tính XML nào sẽ tương ứng với phương thức lớp nào, hoặc biết được lớp nào sẽ ứng với một phần tử XML đã cho. Tuy nhiên, lưu ý rằng không phải tất cả bộ từ vựng đều giống nhau. Trong vài trường hợp, những khác biệt nhỏ trong cách đặt tên vẫn tồn tại. Ví dụ, phần tử EditText có một thuộc tính `text` tương ứng với `EditText.setText()`.

> **Mách nhỏ**: Tìm hiểu thêm về các loại layout khác nhau trong mục "Common Layout Objects" ("Các đối tượng layout thông thường"). Ngoài ra còn có một bộ hướng dẫn về việc xây dựng các layout khác nhau trong tài liệu hướng dẫn Hello Views.

- Plugin ADT cho Eclipse hỗ trợ bạn xem trước layout của file XML - sau khi mở file XML, chọn tab **Layout**.
- Bạn nên thử dùng công cụ Hierarchy Viewer để gỡ lỗi layout - nó hiển thị những giá trị thuộc tính của layout, vẽ những wireframe (bản phác thảo và bố trí các thành phần, chỉ tập trung vào layout, không tập trung vào giao diện đồ họa như hình ảnh, font chữ, màu sắc) có dấu hiện chỉ rõ phần canh đệm và canh lề, và hiển thị toàn bộ view trong khi bạn gỡ lỗi trên phần mềm giả lập hoặc thiết bị thật.
- Công cụ layoutopt cho phép bạn phân tích nhanh layout và cây phân cấp để tìm ra các vấn đề về hiệu suất hoặc những vấn đề khác.

Viết file XML

Sử dụng bộ từ vựng XML của Android, bạn có thể thiết kế nhanh những layout và phần tử màn hình mà chúng bao hàm, theo cách giống như khi bạn tạo trang Web trong HTML - với một chuỗi phần tử lồng nhau (nested element).

Mỗi file layout phải chứa chính xác một phần tử gốc (root element) và phần tử đó phải là một đối tượng `View` hoặc `ViewGroup`. Sau khi đã định nghĩa phần tử gốc, bạn có thể thêm các đối tượng layout hoặc widget dưới dạng phần tử con (child element) để từng bước xây dựng một cây phân cấp View định nghĩa layout của bạn. Ví dụ, sau đây là một layout XML sử dụng đối tượng `LinearLayout` dạng dọc để chứa `TextView` và `Button`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Xin chào, Đây là một TextView" />
    <Button android:id="@+id/button"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Xin chào, Đây là một Button" />
</LinearLayout>
```

Sau khi khai báo layout của mình trong XML, bạn lưu file lại với đuôi `.xml` trong thư mục `res/layout/` của dự án Android, để layout được biên dịch.

Để biết thêm thông tin về cú pháp cho file layout XML, xem tài liệu "Layout Resources" ("Các tài nguyên layout").

Nạp tài nguyên XML

Khi bạn biên dịch ứng dụng, mỗi file layout XML được biên dịch thành một tài nguyên `View`. Bạn nên nạp (load) tài nguyên layout từ mã nguồn của ứng dụng, trong phần thực thi callback (gọi trở lại) `Activity.onCreate()`. Để thực hiện điều đó, bạn gọi `setContentView()`, sau đó truyền cho phương thức này một tham chiếu tới tài nguyên layout của bạn theo mẫu: `R.layout.`*`ten_file_layout`*. Ví dụ, nếu layout XML được lưu thành `main_layout.xml`, bạn có thể nạp layout này cho Activity như sau:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Framework của Android sẽ gọi phương thức callback `onCreate()` trong Activity của bạn khi Activity được khởi động (xem phần thảo luận về vòng đời (lifecycle) trong tài liệu Activities.

Các thuộc tính

Mỗi đối tượng `View` và `ViewGroup` hỗ trợ các thuộc tính XML khác nhau của chúng. Một số thuộc tính là chuyên biệt đối với một đối tượng `View` (ví dụ, `TextView` hỗ trợ thuộc tính `textSize`), nhưng những thuộc tính này còn được bất cứ đối tượng `View` nào có thể mở rộng từ lớp này kế thừa. Một số thuộc tính là thuộc tính chung đối với tất cả các đối tượng `View`, bởi vì chúng được kế thừa từ lớp `View` gốc (như thuộc tính `id` chẳng hạn). Và, các thuộc tính khác được coi như là "tham số của layout" ("layout parameter"), đó là những thuộc tính mô tả hướng layout của đối tượng `View`, như đã được đối tượng cha là `ViewGroup` định nghĩa.

ID

Bất kỳ đối tượng `View` nào cũng có thể sở hữu một ID dạng số nguyên (integer ID) liên kết với nó để chỉ xác định View trong cây phân cấp. Khi ứng dụng được biên dịch, ID

này được tham chiếu dưới dạng số nguyên, nhưng trong file XML của layout, ID được gán dưới dạng chuỗi, trong thuộc tính `id`. Đây là một thuộc tính XML thường xuất hiện ở tất cả các đối tượng `View` (định nghĩa bởi lớp `View`) và bạn sẽ thường xuyên sử dụng nó. Cú pháp của ID bên trong thẻ `XML` là:

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

Biểu tượng `(@)` ở phần đầu chuỗi chỉ ra rằng bộ phân tích XML (XML parser) nên phân tích, mở rộng phần còn lại của chuỗi ID và nhận biết nó như là một tài nguyên ID. Biểu tượng dấu cộng (+) có nghĩa đây là một tên tài nguyên mới được tạo và thêm vào kho tài nguyên (trong file `R.java`). Có một số tài nguyên ID khác được cung cấp bởi framework Android. Khi tham chiếu một ID của tài nguyên Android, bạn không cần biểu tượng dấu cộng, nhưng phải thêm vào namespace của gói `android` như sau:

```
android:id="@android:id/empty"
```

Với namespace gói `android`, hiện chúng ta đang tham chiếu một ID từ lớp tài nguyên `android.R` chứ không phải lớp tài nguyên nội bộ.

Để tạo các view và tham chiếu tới chúng từ ứng dụng, cách thông thường là:

1. Định nghĩa một view/widget trong file layout và gán cho nó một ID duy nhất:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

2. Sau đó, bạn tạo một thể hiện của đối tượng view và kiểm soát nó từ layout (thường là trong phương thức `onCreate()`):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Việc xác định ID cho các đối tượng view đóng vai trò quan trọng khi tạo một `RelativeLayout`. Trong một layout tương đối (relative layout), những view đồng cấp (sibling view) có thể định nghĩa layout của chúng liên quan tới view đồng cấp khác bằng cách tham chiếu tới ID duy nhất của view đó.

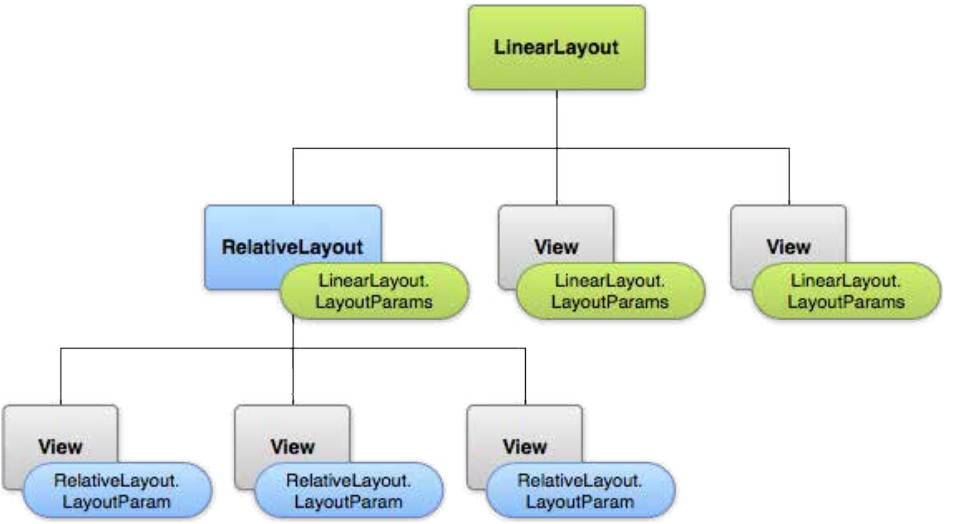
Một ID không cần phải là duy nhất trong toàn bộ cây phân cấp, nhưng nó nên là duy nhất trong một phần của cây mà bạn đang tìm kiếm (cũng có thể đó là toàn bộ cây, do vậy tốt nhất nó nên là duy nhất trên toàn bộ cây khi có thể).

Các thông số layout

Các thuộc tính layout XML được đặt tên dưới dạng layout_*<ten thuoc tinh>* định nghĩa các tham số layout cho View phù hợp với `ViewGroup` chứa nó.

Mỗi lớp ViewGroup cài đặt dưới dạng một lớp lồng, mở rộng từ lớp `ViewGroup.LayoutParams`. Lớp con này chứa các loại thuộc tính xác định kích cỡ và vị trí cho

từng view con (child view) sao cho phù hợp với nhóm view. Như bạn có thể thấy ở Hình 1, nhóm view cha (parent view) xác định các thông số layout cho mỗi view con (bao gồm cả nhóm view con).



Hình 1. Minh họa một cây phân cấp view có các tham số layout liên kết với từng view.

Lưu ý, mọi lớp con `LayoutParams` đều có cú pháp riêng để thiết lập các giá trị. Mỗi phần tử con phải xác định `LayoutParams` thích hợp với phần tử cha của nó, mặc dù phần tử con có thể xác định `LayoutParams` khác cho con của mình.

Mọi nhóm view đều chứa một thông số chiều rộng (width) và chiều cao (height) (`layout_width` và `layout_height`), trong đó mỗi view đều phải xác định hai thông số này. Nhiều `LayoutParams` còn chứa cả các tùy chọn lề (margin) và viền (border).

Bạn có thể xác định chiều rộng và chiều cao với các kích thước chính xác, mặc dù có thể bạn không muốn làm điều này thường xuyên. Thường thì bạn sẽ hay sử dụng một trong những hằng số dưới đây hơn để thiết lập chiều rộng và chiều cao:

- *wrap_content* cho biết view này có kích thước thay đổi phù hợp với nội dung.
- *fill_parent* (hay *match_parent* đã đổi tên trong API Cấp 8) cho biết view này có kích thước lớn bằng kích thước mà view cha cho phép.

Nhìn chung, việc xác định một chiều rộng và chiều cao layout sử dụng các đơn vị tuyệt đối như pixel (điểm ảnh) không được khuyến khích. Thay vào đó, sử dụng các kích thước tương đối như là những đơn vị pixel độc lập với mật độ (density-independent pixel - *dp*), *wrap_content* hoặc *fill_parent*, là phương pháp tốt hơn, vì nó đảm bảo rằng ứng dụng của bạn sẽ hiển thị đúng những kích thước màn hình thiết bị khác nhau. Các loại kích thước được chấp nhận được trình bày trong tài liệu "Available Resources" ("Các tài nguyên có sẵn").

Vị trí layout

Dạng hình học của view là hình chữ nhật. Mỗi view có một vị trí, được thể hiện bằng một cặp tọa độ *left (bên trái)*, *top (bên trên)* cùng hai thông số kích thước là chiều rộng và chiều cao. Đơn vị cho vị trí và kích thước là pixel.

Chúng ta có thể lấy thông số vị trí của một view bằng cách kích hoạt các phương thức `getLeft()` và `getTop()`. Phương thức `getLeft()` trả về thông số left, hay tọa độ X của hình chữ nhật đại diện cho view. Phương thức `getTop()` trả về thông số top, hay tọa độ Y của hình chữ nhật đại diện cho view. Các phương thức này đều trả về vị trí tương đối của view so với cha của nó. Ví dụ, khi `getLeft()` trả về 20, có nghĩa là view có vị trí cách cạnh trái của view cha của nó 20 pixel.

Ngoài ra, có một vài phương thức tiện lợi được cung cấp để tránh các phép tính không cần thiết, đó là `getRight()` và `getBottom()`. Những phương thức này trả về tọa độ của cạnh phải và đáy của hình chữ nhật đại diện cho view. Chẳng hạn, gọi `getRight()` sẽ tương đương với phép tính sau: `getLeft() + getWidth()`.

Kích thước, vùng đệm và căn lề

Kích thước của view được thể hiện bằng một thông số chiều rộng và chiều cao. Thực tế, mỗi view luôn sở hữu hai cặp giá trị chiều rộng và chiều cao.

Cặp thứ nhất được biết đến với tên *chiều rộng đo lường (measured width)* và *chiều cao đo lường (measured height)*. Những độ dài này xác định độ lớn của một view so với phần tử cha. Có thể thu được các độ dài đo lường bằng cách gọi `getMeasuredWidth()` và `getMeasuredHeight()`.

Cặp thứ hai được biết đến đơn giản với tên *chiều rộng (width)* và *chiều cao (height)*, hoặc đôi khi là *chiều rộng hình học (drawing width)* và *chiều cao hình học (drawing height)*. Những độ dài này xác định kích thước thực sự của view trên màn hình, trong thời gian vẽ hình và sau layout. Những giá trị này có thể (không nhất thiết phải) khác biệt với chiều cao đo lường và chiều rộng đo lường. Có thể thu được chiều rộng và chiều cao bằng cách gọi `getWidth()` và `getHeight()`.

Để đo các kích thước, một view tính cả vùng đệm (padding) của nó. Vùng đệm được thể hiện bằng pixel cho phần bên trái, bên trên, bên phải và bên dưới của view. Có thể dùng vùng đệm để thiết lập khoảng cách cho phần nội dung của view với cạnh của view bằng một lượng pixel cụ thể. Ví dụ, vùng đệm bên trái là 2 sẽ đẩy nội dung của view đi 2 pixel về bên phải của cạnh trái. Có thể thiết lập vùng đệm bằng cách dùng phương thức `setPadding(int, int, int, int)`, đồng thời truy vấn vùng này bằng cách gọi `getPaddingLeft()`, `getPaddingTop()`, `getPaddingRight()` và `getPaddingBottom()`.

Mặc dù có thể xác định vùng đệm, song view lại không hỗ trợ lề (margin). Tuy nhiên, các nhóm view lại hỗ trợ. Xem `ViewGroup` và `ViewGroup.MarginLayoutParams` để biết thêm thông tin.

Để tham khảo thông tin về các độ dài, xem tài liệu "Dimension Values" ("Các giá trị độ dài")

Các layout thông dụng

Mỗi lớp con của lớp `ViewGroup` cung cấp một cách thức duy nhất để hiển thị các view lồng trong nó. Dưới đây là một số loại layout thông dụng hơn được xây dựng cho nền tảng Android.

> **Ghi chú:** Mặc dù bạn có thể lồng một hay nhiều layout vào trong layout khác để hoàn thiện thiết kế giao diện của mình, song bạn vẫn nên cố gắng giữ cho cây phân cấp layout có ít cấp nhất. Layout của bạn sẽ vẽ nhanh hơn nếu có ít layout lồng trong đó hơn (một cây phân cấp thiên về chiều rộng sẽ tốt hơn một cây phân cấp thiên về chiều sâu).

Layout tuyến tính

Layout tổ chức các con của nó thành một hàng ngang hoặc dọc. Nó tạo một thanh cuộn dọc (scrollbar) nếu chiều dài cửa sổ vượt quá chiều dài màn hình.

Layout tương đối

Cho phép bạn xác định vị trí tương đối của các đối tượng con (con A ở bên trái con B) hoặc tương đối với đối tượng cha (được căn chỉnh theo cạnh trên cùng của đối tượng cha).

Web View

Hiển thị các trang Web.

Xây dựng layout với một adapter

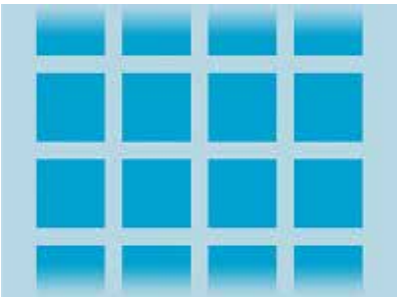
Khi nội dung của layout là động (dynamic) hoặc không được xác định trước, bạn có thể sử dụng một layout là lớp con của AdapterView để tạo ra một layout bao gồm các view tại thời điểm thực thi chương trình. Lớp con của lớp `AdapterView` dùng một `Adapter` để liên kết (bind) dữ liệu cho layout của nó. `Adapter` đóng vai trò là thành phần trung chuyển ở giữa nguồn dữ liệu và layout `AdapterView` - `Adapter` lấy dữ liệu (từ một nguồn, chẳng hạn như một mảng hoặc truy vấn cơ sở dữ liệu) và chuyển đổi mỗi mục thành một view có thể thêm vào layout `AdapterView`.

Các layout thông dụng được hỗ trợ bởi một adapter bao gồm:

List View

Hiển thị một danh sách cuộn gồm một cột đơn.

Grid View



Hiển thị một ô lưới cuộn gồm các cột và dòng.

Điền dữ liệu vào một adapter view

Bạn có thể tạo một `AdapterView`, chẳng hạn như `ListView` hoặc `GridView` bằng cách liên kết thể hiện `AdapterView` với một `Adapter`, `Adapter` lấy dữ liệu từ một nguồn bên ngoài và tạo một `View` biểu diễn mỗi mục dữ liệu.

Android cung cấp một số lớp con của `Adapter` rất hữu ích trong việc truy xuất những loại dữ liệu khác nhau và xây dựng các view cho một `AdapterView`. Hai adapter thông dụng nhất là:

`ArrayAdapter`

Sử dụng adapter này khi nguồn dữ liệu của bạn là một mảng. Mặc định, `ArrayAdapter` tạo một view cho từng phần tử mảng bằng cách gọi phương thức `toString()` trên mỗi thành viên và đặt nội dung vào một `TextView`.

Ví dụ, nếu bạn có một mảng các chuỗi muốn hiển thị trong một `ListView`, hãy khởi tạo một `ArrayAdapter` mới bằng cách sử dụng một phương thức khởi tạo (constructor) để xác định layout cho từng chuỗi và mảng chuỗi:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, myStringArray);
```

Các tham số cho phương thức khởi tạo trên là:

- Ứng dụng `Context` của bạn.
- Layout chứa một `TextView` cho từng chuỗi trong mảng.
- Mảng các chuỗi.

Sau đó, bạn chỉ cần gọi `setAdapter()` trong `ListView`:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

Để tùy chỉnh hình thức hiển thị của mỗi mục, bạn có thể ghi đè (override) phương thức `toString()` cho các đối tượng trong mảng. Hoặc, để tạo một view cho mỗi mục khác `TextView` (ví dụ nếu bạn muốn một `ImageView` cho từng mục), hãy mở rộng lớp `ArrayAdapter` và ghi đè phương thức `getView()` để trả về loại của view bạn muốn cho mỗi mục.

`SimpleCursorAdapter`

Sử dụng adapter này khi dữ liệu của bạn tới từ một đối tượng `Cursor` (con trỏ). Khi sử dụng `SimpleCursorAdapter`, bạn phải chỉ định một layout để sử dụng cho mỗi dòng trong `Cursor` và các cột trong `Cursor` nên được chèn vào những view tương ứng của layout. Ví dụ, nếu muốn tạo một danh sách tên mọi người kèm theo số điện thoại, bạn có thể thực thi một truy vấn trả về một `Cursor` chứa một dòng cho từng người và các cột cho tên cùng số điện thoại. Sau đó, bạn tạo một mảng chuỗi xác định các cột từ `Cursor` mà bạn muốn trong layout cho từng kết quả và một mảng số nguyên xác định những view tương ứng mà mỗi cột nên được đặt vào đó:

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

Khi bạn khởi tạo `SimpleCursorAdapter`, hãy truyền layout để sử dụng cho từng kết quả, `Cursor` sẽ chứa các kết quả và hai mảng sau:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
   R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView();
listView.setAdapter(adapter);
```

Sau đó, adapter `SimpleCursorAdapter` sẽ tạo một view cho từng dòng trong `Cursor`, sử dụng layout thu được từ việc chèn mỗi mục `fromColumns` vào view `toViews` tương ứng.

Nếu trong suốt vòng đời của ứng dụng, bạn thay đổi dữ liệu cơ bản được adapter của mình đọc, bạn nên gọi `notifyDataSetChanged()`. Thao tác này sẽ thông báo cho view đính kèm rằng dữ liệu đã bị thay đổi và nó nên tự cập nhật.

Xử lý các sự kiện click

Bạn có thể phản hồi các sự kiện Click trên từng mục trong một `AdapterView` bằng cách thực thi giao diện (interface) `AdapterView.OnItemClickListener`. Ví dụ:

```
// Tạo một thông điệp xử lý đối tượng dưới dạng một lớp ẩn danh
// (anonymous class).
private OnItemClickListener mMessageClickedHandler =
new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position,
long id) {
        // Làm gì đó để phản hồi lại sự kiện Click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

5.2.1 Layout tuyến tính

LinearLayout là một nhóm view có chức năng căn chỉnh toàn bộ con của nó theo một hướng duy nhất, có thể là dọc hoặc ngang. Bạn có thể xác định hướng của layout nhờ thuộc tính `android:orientation`.

Tất cả view con của một LinearLayout đều được xếp chồng (stack) lên nhau, nên một danh sách dọc chỉ có duy nhất một con trên từng dòng, bất kể chúng rộng bao nhiêu đi nữa; trong khi đó, một danh sách ngang sẽ chỉ cao một dòng (chiều cao của con cao nhất, cộng thêm vùng đệm). Một LinearLayout bị ảnh hưởng bởi các lề (margin) giữa những view con và kiểu căn chỉnh (gravity) (căn phải, căn giữa hoặc căn trái) của từng view con đó.

Trọng số của layout

LinearLayout cũng hỗ trợ việc gán một *trọng số (weight)* cho riêng từng *view* con với thuộc tính `android:layout_weight`. Thuộc tính này gán một giá trị "tầm quan

trọng" ("importance") cho một view theo nghĩa là nó sẽ chiếm bao nhiêu không gian trên màn hình. Một giá trị trọng số lớn hơn sẽ cho phép nó mở rộng để lấp đầy bất kỳ không gian còn lại nào trong view cha. Các view con có thể xác định một giá trị trọng số, sau đó bất cứ không gian còn lại nào trong nhóm view cũng được gán cho các con theo tỷ lệ đã khai báo. Trọng số mặc định là 0.

Ví dụ, nếu có ba trường văn bản và hai trong số đó khai báo một trọng số là 1, trong khi trường khác không có trọng số, trường văn bản thứ ba không có trọng số đó sẽ không tăng độ dài và sẽ chỉ chiếm vùng không gian nhất định theo nội dung mà nó yêu cầu. Hai trường kia sẽ cùng mở rộng để lấp đầy không gian còn lại sau khi cả ba trường được đo lường. Nếu, trường thứ ba được thiết lập trọng số là 2 (thay vì 0), tức hiện tại trường thứ ba được khai báo là quan trọng hơn hai trường kia, trường này sẽ lấy một nửa trong tổng không gian còn lại, trong khi hai trường kia chia đều khoảng không gian thừa ra cho nhau.

Các view con cùng trọng số

Để tạo một layout tuyến tính, trong đó mỗi view con sử dụng cùng một khoảng không gian trên màn hình, hãy thiết lập giá trị của `android:layout_height` của từng view thành `"0dp"` (cho layout dọc) hoặc `android:layout_width` của mỗi view thành `"0dp"` (cho layout ngang). Sau đó, bạn đặt giá trị cho `android:layout_weight` của từng view bằng "`1`".

Ví dụ

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
```

```
        android:hint="@string/message" />
<Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

Để biết thêm chi tiết về các thuộc tính cho từng view con của một `LinearLayout`, xem `LinearLayout.LayoutParams`.

5.2.2 Layout tương đối

`RelativeLayout` là một nhóm view có tính năng hiển thị các view con theo những vị trí tương đối. Vị trí của mỗi view có thể được xác định là tương đối với các phần tử ngang cấp (chẳng hạn như so với cạnh trái và cạnh dưới của view khác), hoặc theo vị trí tương đối so với cha `RelativeLayout` (chẳng hạn như so với bên dưới, bên trái của phần giữa).

`RelativeLayout` là một tiện ích rất hiệu quả trong thiết kế giao diện người dùng, vì nó có thể xóa bỏ các nhóm view lồng nhau và giữ cho cây phân cấp layout có ít cấp nhằm gia tăng hiệu suất. Nếu tự thấy mình sử dụng vài nhóm `LinearLayout` lồng nhau, bạn có thể thay thế chúng bằng một `RelativeLayout`.

Định vị các view

`RelativeLayout` cho phép các view con ấn định vị trí tương đối của chúng so với view cha hoặc với một view khác (được xác định bằng ID). Do vậy, bạn có thể căn chỉnh hai phần tử bằng viền phải, hoặc làm cho view nọ bên dưới view kia, nằm ở giữa màn hình, nằm ở giữa theo chiều dọc và bên trái theo chiều ngang,... Theo mặc định, tất cả view con đều được vẽ ở góc trên bên trái của layout, nên bạn phải định nghĩa vị trí của từng view, sử dụng nhiều thuộc tính của layout có sẵn trong `RelativeLayout.LayoutParams`.

Một số thuộc tính layout dùng cho các view trong một `RelativeLayout` bao gồm:

`android:layout_alignParentTop`

Nếu `"true"`: Làm cho góc trên của view này khớp với góc trên của view cha.

`android:layout_centerVertical`

Nếu `"true"`: Căn giữa view con này theo chiều dọc trong phạm vi view cha.

`android:layout_below`

Các vị trí cạnh trên của view này nằm dưới view có ID được chỉ định.

`android:layout_toRightOf`

Các vị trí cạnh trái của view này nằm về bên phải của view có ID được chỉ định.

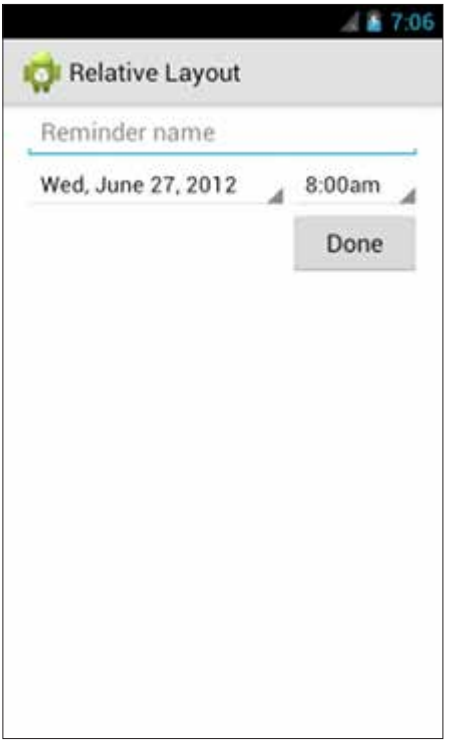
Trên đây chỉ là một số ví dụ. Tất cả thuộc tính của layout được ghi tại `RelativeLayout.LayoutParams`.

Giá trị cho mỗi thuộc tính layout có thể là boolean (true/false) để cho phép một vị trí layout tương đối với cha `RelativeLayout`, hoặc một ID tham chiếu tới một view khác trong layout, nơi view này được định vị.

Trong layout XML, các đối tượng phụ thuộc vào những view khác có thể khai báo theo thứ tự bất kỳ. Ví dụ, bạn có thể khai báo rằng "view1" nằm bên dưới "view2" ngay cả nếu khi "view2" là view cuối cùng được khai báo trong cây phân cấp. Ví dụ dưới đây minh họa một trường hợp như vậy.

Ví dụ

Trong đoạn mã dưới đây, mỗi thuộc tính kiểm soát vị trí tương đối của từng view được in đậm.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

Để biết thêm chi tiết về tất cả các thuộc tính của layout cho từng view con của một RelativeLayout, xem RelativeLayout.LayoutParams.

5.2.3 List View

ListView là nhóm view có tính năng hiển thị danh sách những mục có thể cuộn được (scrollable item). Các mục của danh sách được chèn tự động vào danh sách bằng cách sử dụng một Adapter để lấy nội dung từ một nguồn, như một mảng hoặc truy vấn cơ sở dữ liệu, đồng thời chuyển đổi từng kết quả thành một view đặt bên trong danh sách.

Để xem hướng dẫn cách chèn động các view bằng cách sử dụng adapter, mời bạn tham khảo mục "Xây dựng layout với một adapter".

Sử dụng Loader

Sử dụng `CursorLoader` là phương pháp chuẩn để truy vấn một `Cursor` (con trỏ) dưới dạng một tác vụ không đồng bộ (asynchronous task) để tránh chặn luồng (thread) chính của ứng dụng khi truy vấn. Khi `CursorLoader` nhận kết quả `Cursor`, `LoaderCallbacks` nhận một phương thức callback tới `onLoadFinished()`, đó là nơi bạn cập nhật `Adapter` với `Cursor` mới và list view để hiển thị kết quả về sau.

Mặc dù các API `CursorLoader` được giới thiệu lần đầu trong Android 3.0 (API Cấp 11), song chúng đều có mặt trong Support Library (thư viện hỗ trợ) nên ứng dụng của bạn có thể sử dụng chúng nếu nó hỗ trợ các thiết bị chạy Android 1.6 hoặc cao hơn.

Muốn biết thêm thông tin về cách sử dụng `Loader` để nạp dữ liệu theo cách không đồng bộ, hãy xem hướng dẫn về Loaders (Các trình Loader).

Ví dụ

Ví dụ sau đây sử dụng `ListActivity`, đó là activity chứa một `ListView` với tư cách là phần tử layout duy nhất theo mặc định. Nó thực thi một truy vấn tới Contacts Provider để yêu cầu danh sách các tên và số điện thoại.

Activity thi hành giao diện `LoaderCallbacks` để sử dụng `CursorLoader` nhằm nạp động dữ liệu cho list view.

```
public class ListViewLoader extends ListActivity
       implements LoaderManager.LoaderCallbacks<Cursor> {

    // Đây là Adapter được sử dụng để hiển thị dữ liệu của danh sách
     SimpleCursorAdapter mAdapter;

    // Đây là các dòng danh bạ mà chúng ta sẽ nhận được
    static final String[] PROJECTION = new String[] {ContactsContract.Data._ID,
           ContactsContract.Data.DISPLAY_NAME};

    // Đây là điều kiện chọn
    static final String SELECTION = "((" +
           ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND (" +
           ContactsContract.Data.DISPLAY_NAME + " != '' ))";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Tạo một thanh tiến trình (progress bar) để hiển thị khi danh sách nạp
        ProgressBar progressBar = new ProgressBar(this);
        progressBar.setLayoutParams(new
LayoutParams(LayoutParams.WRAP_CONTENT,
              LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        progressBar.setIndeterminate(true);
        getListView().setEmptyView(progressBar);

        // Phải thêm thanh tiến trình vào gốc của layout
        ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
        root.addView(progressBar);

        // Đối với adapter con trỏ, hãy xác định những cột nào sẽ đi
        // với view nào
        String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME};
        int[] toViews = {android.R.id.text1}; // TextView trong
        //simple_list_item_1

        // Tạo một adapter trống, chúng ta sẽ dùng nó để hiển thị dữ
        // liệu đã nạp.
        // Chúng ta truyền null vào con trỏ, sau đó cập nhật nó trong
        // onLoadFinished()
        mAdapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1, null,
                fromColumns, toViews, 0);
            setListAdapter(mAdapter);

        // Chuẩn bị loader. Hoặc kết nối lại với một loader đã có sẵn,
        // hoặc khởi động một loader mới.
```

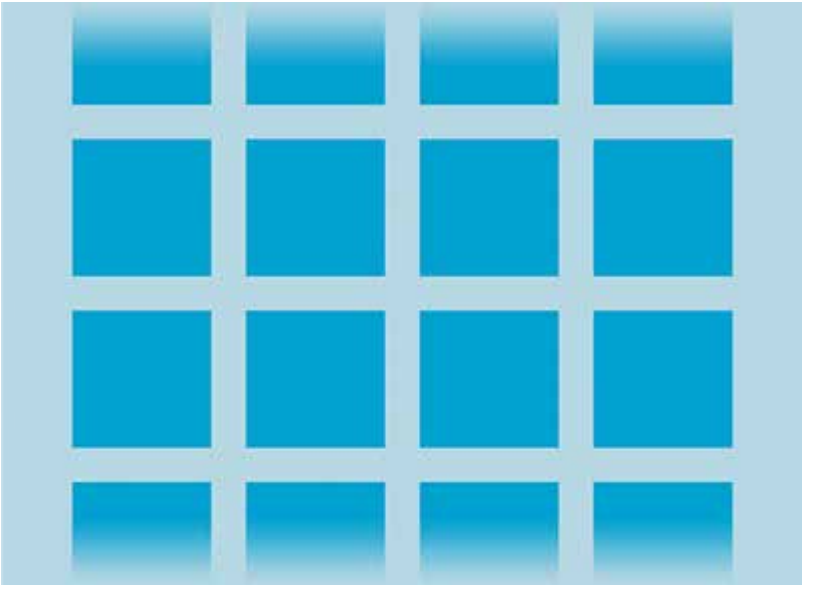
```
        getLoaderManager().initLoader(0, null, this);
    }

    // Được gọi khi một loader mới phải được tạo ra
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // Lúc này, hãy tạo và trả về một CursorLoader, CursorLoader
        // này sẽ đảm nhiệm việc tạo một Con trỏ cho dữ liệu được hiển thị.
        return new CursorLoader(this, ContactsContract.Data.CONTENT_URI,
            PROJECTION, SELECTION, null, null);
    }

    // Được gọi khi một loader đã tạo từ trước được nạp xong
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Tráo đổi (swap) con trỏ mới vào. (Framework sẽ tiến hành
        // việc đóng con trỏ cũ khi chúng ta quay về).
        mAdapter.swapCursor(data);
    }

    // Được gọi khi một trình loader đã tạo từ trước được phục hồi về
    // trạng thái ban đầu (reset), làm cho dữ liệu không khả dụng
    public void onLoaderReset(Loader<Cursor> loader) {
        // Được gọi khi Con trỏ cuối cùng được gửi tới onLoadFinished()
        // ở bên trên bị đóng. Chúng ta cần đảm bảo là sẽ không sử dụng
        // con trỏ này nữa.
        mAdapter.swapCursor(null);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Làm gì đó khi một mục danh sách được nhấn
    }
}
```

> **Ghi chú**: Do ví dụ này thực thi một truy vấn trên Contacts Provider nên nếu bạn muốn thử đoạn mã này, ứng dụng của bạn phải yêu cầu quyền READ_CONTACTS trong file kê khai (manifest file):
>
> `<uses-permission android:name="android.permission.READ_CONTACTS" />.`

5.2.4 Grid View

`GridView` là một `ViewGroup` có tính năng hiển thị các mục trong một lưới hai chiều có thể cuộn được. Các mục của lưới được tự động chèn vào layout bằng cách sử dụng một `ListAdapter`.

Để xem hướng dẫn cách chèn động các view bằng cách sử dụng một adapter, mời bạn tham khảo mục "Xây dựng layout với một adapter" trong cuốn sách này.



Ví dụ

Trong hướng dẫn này, bạn sẽ tạo một lưới gồm các mẫu ảnh (image thumbnail - dạng hình ảnh thu nhỏ của một bức ảnh). Khi một mục được chọn, một thông điệp tức thời (toast message) sẽ hiển thị vị trí của ảnh.

1. Bắt đầu một dự án mới có tên *HelloGridView*.
2. Tìm một số tấm ảnh bạn muốn sử dụng, hoặc tải về các ảnh mẫu này. Lưu các file ảnh vào thư mục `res/drawable/` của dự án.
3. Mở file `res/layout/main.xml` và thêm đoạn mã sau vào:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

GridView này sẽ lấp đầy toàn bộ màn hình. Hầu hết các thuộc tính đều khá dễ hiểu. Để biết thêm thông tin về những thuộc tính hợp lệ, mời bạn xem phần tham khảo GridView.

4. Mở HelloGridView.java và chèn đoạn mã sau cho phương thức onCreate():

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int
        position, long id) {
    Toast.makeText(HelloGridView.this, "" + position, Toast.LENGTH_
  SHORT).show();
        }
    });
}
```

Sau khi layout main.xml được thiết lập cho view nội dung, GridView được lấy về từ layout thông qua phương thức findViewById(int). Sau đó, phương thức setAdapter() thiết lập một adapter tùy chọn (ImageAdapter) với tư cách là nguồn cho tất cả các mục cần hiển thị trong lưới. ImageAdapter được tạo ở bước tiếp theo.

Để thực hiện một tác vụ nào đó khi nhấn vào một mục trong lưới, phương thức setOnItemClickListener() được truyền một AdapterView.OnItemClickListener mới. Thể hiện vô danh vô danh (anonymous instance) này xác định phương thức callback onItemClick() để thực thi một Toast hiển thị vị trí chỉ số (index) (bắt đầu từ số 0) của mục được chọn (thực tế, vị trí có thể được dùng để lấy ảnh ở kích thước gốc cho những tác vụ khác).

5. Tạo một lớp mới có tên ImageAdapter mở rộng BaseAdapter:

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
     }

    public Object getItem(int position) {
         return null;
```

```
}
    public long getItemId(int position) {
         return 0;
    }
    // tạo một ImageView mới cho mỗi mục tham chiếu bởi Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {  // nếu convertView chưa bị hủy, hãy khởi
                                    // tạo một số thuộc tính
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }
        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
    // tham chiếu tới các ảnh
        private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            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,
            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
        };
}
```

Trước hết, lớp này thực thi một số phương thức cần thiết thừa kế từ `BaseAdapter`. Phương thức khởi tạo và `getCount()` là hai phương thức đơn giản, không cần giải thích thêm. Thông thường, `getItem(int)` nên trả về đối tượng thực sự ở vị trí xác định trong adapter nhưng trong ví dụ này, nó đã bị bỏ qua. Tương tự, `getItemId(int)` nên trả về id hàng của mục, song ở đây, việc này là không cần thiết.

Phương thức cần thiết đầu tiên là `getView()`. Phương thức này tạo một `View` mới cho mỗi ảnh được thêm vào `ImageAdapter`. Khi `getView()` được gọi, một `View` được truyền vào, đó thường là một đối tượng có khả năng bị thu hồi bộ nhớ (ít nhất là sau khi phương thức này được gọi một lần), nên ở đây sẽ có một bước kiểm tra để xem đối tượng này có bằng null không. Nếu đối tượng là null, một đối tượng `ImageView` sẽ được khởi tạo và cấu hình với các thuộc tính mong muốn cho việc trình bày ảnh:

- `setLayoutParams(ViewGroup.LayoutParams)` thiết lập chiều cao và chiều rộng cho View - điều này đảm bảo rằng, cho dù kích cỡ của drawable là bao nhiêu, mỗi ảnh sẽ được thay đổi kích thước và cắt xén (crop) để phù hợp với những độ dài này.
- `setScaleType(ImageView.ScaleType)` khai báo rằng các bức ảnh nên được cắt xén quanh trung tâm (nếu cần).
- `setPadding(int, int, int, int)` xác định vùng đệm (padding) cho tất cả các cạnh bên. (Lưu ý, nếu các ảnh có tỷ lệ khung hình khác nhau, ảnh nào có ít vùng đệm hơn sẽ bị cắt xén nhiều hơn nếu nó không khớp với những độ dài được đưa ra cho ImageView).

  Nếu `View` truyền tới `getView()` *khác* null, `ImageView` cục bộ được khởi tạo với đối tượng có khả năng được tái sử dụng là `View`.

  Ở cuối phương thức `getView()`, số nguyên `position` truyền tới phương thức được sử dụng để chọn một ảnh từ mảng `mThumbIds`, mảng này được thiết lập là tài nguyên ảnh cho `ImageView`.

  Đoạn mã còn lại định nghĩa `mThumbIds` như là một mảng gồm các tài nguyên drawable.

6. Chạy ứng dụng.

   Thử nghiệm với những hành vi của `GridView` và các phần tử `ImageView` bằng cách điều chỉnh thuộc tính của chúng. Ví dụ, thay vì dùng `setLayoutParams(ViewGroup.LayoutParams)`, hãy sử dụng `setAdjustViewBounds(boolean)`.

5.3 Các sự kiện đầu vào

Trên Android, có nhiều cách để chặn (intercept) các sự kiện từ một action tương tác của người dùng với ứng dụng của bạn. Khi xem xét các sự kiện trong giao diện người dùng, bạn nên áp dụng phương pháp bắt các sự kiện từ đối tượng View cụ thể mà người dùng tương tác cùng. Lớp View cung cấp phương tiện để thực hiện điều này.

Trong những lớp View khác nhau dùng để tạo layout, hãy lưu ý rằng có vài phương thức callback dạng public hữu ích cho các sự kiện giao diện người dùng. Những phương thức này được framework Android gọi khi action tương ứng xảy ra trên đối tượng đó. Ví dụ, khi nhấn vào một View (chẳng hạn như một Button), phương thức `onTouchEvent()` sẽ được gọi trên đối tượng đó. Tuy nhiên, để ngăn chặn điều này, bạn phải mở rộng lớp và ghi đè lên phương thức đó. Nhưng việc mở rộng mọi đối tượng View để xử lý sự kiện như vậy sẽ không thiết thực. Đó là lý do tại sao lớp View còn chứa một tập các giao diện lồng nhau với những callback mà bạn có thể dễ định

nghĩa hơn nhiều. Các giao diện như vậy gọi là trình lắng nghe sự kiện (event listener), đó là chìa khóa để bắt những tương tác của người dùng với giao diện.

Trong khi bạn sử dụng trình lắng nghe sự kiện để lắng nghe tương tác của người dùng, có thể sẽ đến lúc nào đó bạn muốn mở rộng một lớp View để xây dựng một thành phần tùy chỉnh. Có thể bạn muốn mở rộng lớp Button để tạo ra cái gì đó hay hơn chẳng hạn. Trong trường hợp này, bạn có thể định nghĩa các hành vi sự kiện mặc định cho lớp bằng cách sử dụng trình xử lý sự kiện (event handler) cho lớp.

Trình lắng nghe sự kiện

Trình lắng nghe sự kiện là một giao diện trong lớp `View` chứa một phương thức callback. Những phương thức này sẽ được framework Android gọi khi trình lắng nghe sự kiện đã đăng ký được kích hoạt trên view do người dùng tương tác với một mục trên giao diện.

Những phương thức callback sau được bao hàm trong trình lắng nghe sự kiện:

`onClick()`

Từ `View.OnClickListener`. Phương thức này được gọi khi người dùng chạm vào mục (ở chế độ cảm ứng), hoặc chuyển focus[3] vào mục đó bằng cách sử dụng các phím điều hướng (nevigation key) hay bi lăn và nhấn phím "enter" hoặc nhấn bi lăn xuống.

`onLongClick()`

Từ `View.OnLongClickListener`. Phương thức này được gọi khi người dùng chạm rồi giữ vào mục (ở chế độ cảm ứng), hoặc focus vào mục đó với các phím điều hướng hay bi lăn và nhấn giữ phím "enter" phù hợp hoặc nhấn giữ bi lăn (trong một giây).

`onFocusChange()`

Từ `View.OnFocusChangeListener`. Phương thức này được gọi khi người dùng di chuyển đến hoặc rời khỏi một mục bằng cách sử dụng các phím điều hướng hoặc bi lăn.

`onKey()`

Từ `View.OnKeyListener`. Phương thức này được gọi khi người dùng focus vào một mục và nhấn hoặc thả một phím cứng trên thiết bị.

`onTouch()`

Từ `View.OnTouchListener`. Phương thức này được gọi khi người dùng thực hiện một action, chẳng hạn như một sự kiện chạm (touch event), bao gồm nhấn, thả hay bất cứ động tác di chuyển nào trên màn hinh (trong phạm vi của mục).

`onCreateContextMenu()`

Từ `View.OnCreateContextMenuListener`. Phương thức này được gọi khi một menu ngữ cảnh (context menu) đang được dựng (kết quả của việc "nhấn lâu" ("long click"). Xem thảo luận về menu ngữ cảnh trong phần hướng dẫn cho nhà phát triển Menu của sách này.

[3]Trạng thái của điều khiển đang được kích hoạt hay được chọn để người dùng có thể thao tác hoặc nhập liệu với nó.

Mỗi phương thức này là thành phần duy nhất của giao diện tương ứng. Để định nghĩa một trong những phương thức này và xử lý các sự kiện, hãy thực thi giao diện lồng (nested interface) trong Activity hoặc định nghĩa nó như là một lớp ẩn danh. Sau đó, hãy truyền một thể hiện của lớp này tới phương thức `View.set...Listener()` tương ứng. (Ví dụ, hãy gọi `setOnClickListener()` và truyền cho nó thể hiện của `OnClickListener`).

Ví dụ bên dưới chỉ rõ cách đăng ký một trình lắng nghe sự kiện on-click cho một Button.

```
// Tạo một lớp ẩn danh của OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // làm gì đó khi button được nhấn
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Lấy về button từ layout
    Button button = (Button)findViewById(R.id.corky);
    // Đăng ký trình lắng nghe onClick với lớp ẩn danh bên trên
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Bạn có thể thấy thuận tiện hơn khi thực thi `OnClickListener` như là một phần của Activity. Điều này sẽ ngăn việc nạp thêm lớp và cấp phát đối tượng không xảy ra. Ví dụ:

```
public class ExampleActivity extends Activity implements OnClickListener
{
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Triển khai callback OnClickListener
    public void onClick(View v) {
        // làm gì đó khi button được nhấn
    }
     ...
}
```

Lưu ý, callback `onClick()` trong ví dụ bên trên không có giá trị trả về, nhưng một số phương thức lắng nghe sự kiện khác phải trả về một kết quả dạng boolean. Lý do tùy thuộc vào sự kiện. Đối với một số phương thức khác đó, lý do chính là:

- `onLongClick()` - trả về một boolean cho biết liệu bạn đã sử dụng xong sự kiện hay chưa và bạn không cần xử lý nó nữa. Tức là, trả về *true (đúng)* để chỉ ra rằng bạn đã xử lý sự kiện và nó nên dừng tại đây; trả về *false (sai)* nếu bạn chưa xử lý xong sự kiện và/hoặc nó nên tiếp tục làm việc với bất cứ trình lắng nghe sự kiện on-click nào khác.
- `onKey()` - trả về một boolean cho biết liệu bạn đã sử dụng xong sự kiện hay chưa và bạn không cần xử lý nó nữa. Tức là, trả về *true* để chỉ ra rằng bạn đã xử lý xong sự kiện và nó nên dừng tại đây; trả về *false* nếu bạn chưa xử lý xong sự kiện và/hoặc nó nên tiếp tục làm việc với bất cứ trình lắng nghe sự kiện on-key nào khác.
- `onTouch()` - trả về một boolean cho biết liệu trình lắng nghe sự kiện xử lý xong sự kiện này. Vấn đề quan trọng là sự kiện này có thể có nhiều action theo sau nhau. Vì vậy, nếu trả về false khi sự kiện action nhấn chạm vào màn hình được nhận, có nghĩa là bạn **chưa** dùng xong sự kiện và cũng không quan tâm tới các action theo sau sự kiện này. Do vậy, bạn sẽ không được gọi cho bất cứ action nào khác trong sự kiện, chẳng hạn như một cử động ngón tay hoặc sự kiện action cuối cùng.

Cần nhớ rằng, các sự kiện phím cứng luôn được gửi tới View hiện tại mà bạn đang focus vào. Chúng được gửi bắt đầu từ đỉnh của cây phân cấp View, sau đó xuống dần tới khi đến được đích thích hợp. Nếu View của bạn (hoặc một con của View) đang được focus, bạn có thể thấy sự kiện truyền qua phương thức `dispatchKeyEvent()`. Như một giải pháp thay thế cho việc bắt các sự kiện phím qua View, bạn có thể nhận tất cả sự kiện trong Activity với `onKeyDown()` và `onKeyUp()`.

Đồng thời, khi nghĩ về đầu vào văn bản cho ứng dụng của mình, bạn cần nhớ rằng nhiều thiết bị chỉ có các phương tiện nhập liệu (input method) mềm. Các phương án này không đòi hỏi phải dựa trên phím cứng; một số có thể sử dụng cách nhập liệu bằng giọng nói (voice input), nhập liệu bằng chữ viết (handwriting input) và nhiều loại khác. Thậm chí, nếu một phương tiện nhập liệu hiển thị một giao diện giống bàn phím, thường thì nó sẽ **không** kích hoạt họ các sự kiện `onKeyDown()`. Bạn không nên xây dựng một giao diện người dùng đòi hỏi bấm vào phím cụ thể để kiểm soát, trừ phi bạn muốn hạn chế ứng dụng của mình chỉ chạy với những thiết bị có bàn phím cứng. Cụ thể, bạn nên thiết kế sao cho không phụ thuộc vào các phương thức để kiểm tra đầu vào khi người dùng nhấn phím return, thay vào đó sử dụng những action như `IME_ACTION_DONE` để báo cho phương tiện nhập liệu biết ứng dụng trông đợi cách phản ứng như thế nào, từ đó có thể thay đổi giao diện người dùng theo hướng tốt hơn. Hãy loại bỏ những giả định không cần thiết về cách hoạt động của phương tiện nhập liệu mềm và chỉ việc tin tưởng nó cung cấp văn bản đã định dạng cho ứng dụng của bạn.

> **Ghi chú**: Android sẽ gọi các phương thức xử lý sự kiện trước, sau đó là những phương thức xử lý mặc định phù hợp với định nghĩa lớp ở bước thứ hai. Như vậy, việc trả về *true* từ những trình lắng nghe sự kiện này sẽ ngăn sự kiện không truyền tới các trình lắng nghe sự kiện khác, đồng thời chặn phương thức callback đối với phương thức xử lý sự kiện mặc định trong View. Vì vậy, hãy chắc chắn rằng bạn muốn chấm dứt sự kiện khi trả về *true*.

Các phương thức xử lý sự kiện

Nếu đang xây dựng một thành phần tùy chỉnh từ View, bạn có thể định nghĩa một vài phương thức callback như những phương thức xử lý sự kiện mặc định. Trong tài liệu "Thành phần tùy chỉnh", bạn sẽ biết được một số callback thông dụng dùng cho việc xử lý sự kiện, bao gồm:

- `onKeyDown(int, KeyEvent)` - Được gọi khi xảy ra một sự kiện nhấn phím.
- `onKeyUp(int, KeyEvent)` - Được gọi khi xảy ra một sự kiện thả phím.
- `onTrackballEvent(MotionEvent)` - Được gọi khi xảy ra một sự kiện di chuyển bi lăn.
- `onTouchEvent(MotionEvent)` - Được gọi khi xảy ra một sự kiện di chuyển trên màn hình cảm ứng.
- `onFocusChanged(boolean, int, Rect)` - Được gọi khi một view được focus hoặc mất focus.

Có một số phương thức khác không phải là một phần của lớp View, nhưng có thể trực tiếp ảnh hưởng tới cách bạn xử lý các sự kiện. Do vậy, khi quản lý nhiều sự kiện phức tạp hơn trong một layout, hãy xem xét các phương thức sau:

- `Activity.dispatchTouchEvent(MotionEvent)` - Cho phép `Activity` của bạn chặn tất cả các sự kiện chạm màn hình trước khi chúng được gửi đến window (cửa sổ).
- `ViewGroup.onInterceptTouchEvent(MotionEvent)` - Cho phép `ViewGroup` theo dõi các sự kiện khi chúng được gửi tới những View con.
- `ViewParent.requestDisallowInterceptTouchEvent(boolean)` - Gọi phương thức này trên View cha để chỉ ra rằng view cha không chặn các sự kiện chạm màn hình với `onInterceptTouchEvent(MotionEvent)`.

Chế độ cảm ứng

Khi người dùng thực hiện điều hướng trên giao diện với các phím điều hướng (directional key) hoặc bi lăn, điều cần thiết là phải focus vào những mục có khả năng tương tác (như button), giúp người dùng biết được cái gì sẽ tiếp nhận đầu vào. Tuy nhiên, nếu thiết bị có tính năng cảm ứng và người dùng bắt đầu tương tác với giao diện bằng cách chạm lên màn hình thì không nhất th iết phải đánh dấu các mục hoặc focus vào một View cụ thể. Do đó, có một chế độ tương tác gọi là "chế độ cảm ứng" ("touch mode").

Đối với một thiết bị có khả năng cảm ứng, một khi người dùng chạm vào màn hình, thiết bị sẽ chuyển sang chế độ cảm ứng. Từ lúc này trở đi, chỉ những View mà tại đó, `isFocusableInTouchMode()` có giá trị true sẽ được focus vào, chẳng hạn như các widget chỉnh sửa văn bản. Những View khác cũng có khả năng cảm ứng, như các button, sẽ không được focus khi người dùng chạm vào; thay vì vậy chúng là kích hoạt các trình lắng nghe sự kiện on-click khi bạn nhấn vào.

Bất cứ thời điểm nào người dùng nhấn phím điều hướng hoặc lăn bi lăn, thiết bị sẽ thoát khỏi chế độ cảm ứng và tìm một view để focus vào. Lúc này, người dùng có thể tiếp tục tương tác với giao diện người dùng mà không phải chạm vào màn hình.

Chế độ cảm ứng được duy trì trên toàn hệ thống (tất cả window và activity). Để truy vấn trạng thái hiện thời, bạn có thể gọi `isInTouchMode()` để xem thiết bị có đang ở chế độ cảm ứng hay không.

Xử lý sự kiện focus

Framework sẽ định kỳ xử lý thay đổi về focus để đáp ứng thao tác nhập liệu từ người dùng. Điều này bao gồm việc thay đổi focus khi một số View bị xóa hoặc ẩn đi, hay khi View mới tạo ra. Các view cho biết chúng đã sẵn sàng để nhận focus thông qua phương thức `isFocusable()`. Để di chuyển focus vào một View, hãy gọi `setFocusable()`. Khi ở chế độ cảm ứng, bạn có thể truy vấn xem một View có cho phép focus hay không với `isFocusableInTouchMode()`. Bạn có thể thay đổi điều này với `setFocusableInTouchMode()`.

Việc di chuyển focus được dựa vào thuật toán tìm hàng xóm gần nhất theo hướng đã cho. Trong một số trường hợp hiếm gặp, thuật toán mặc định có thể không khớp với hành vi dự định của lập trình viên. Trong những tình huống đó, bạn có thể cung cấp các thuộc tính ghi đè tường minh lên những thuộc tính XML trong file layout là *nextFocusDown, nextFocusLeft,nextFocusRight và nextFocusUp*. Hãy thêm một trong các thuộc tính này vào View sẽ mất focus, đồng thời định nghĩa giá trị thuộc tính bằng giá trị id của View sẽ nhận focus. Ví dụ:

```
<LinearLayout
        android:orientation=”vertical”
        ... >
    <Button android:id=”@+id/top”
        android:nextFocusUp=”@+id/bottom”
        ... />
    <Button android:id=”@+id/bottom”
        android:nextFocusDown=”@+id/top”
        ... />
</LinearLayout>
```

Trong layout dọc này, điều hướng lên trên từ Button đầu tiên sẽ không đi đâu cả, và cũng không điều hướng trở xuống từ Button thứ hai. Lúc này, Button trên cùng đã xác định nút dưới cùng là *nextFocusUp* (và ngược lại), màn hình sẽ focus vào các đối tượng lần lượt và tuần hoàn từ trên xuống dưới và từ dưới lên trên.

Nếu bạn muốn khai báo một View có thể focus được trong giao diện người dùng (dù nó có theo truyền thống hay không), hãy thêm thuộc tính XML `android:focusable` vào View đó, trong phần khai báo layout và thiết lập giá trị true. Bạn cũng có thể khai báo một View có thể focus được trong lúc ở chế độ cảm ứng với `android:focusableInTouchMode`.

Để yêu cầu một View nhận focus, gọi `requestFocus()`.

Để lắng nghe các sự kiện focus (khi một View được focus hoặc mất focus), sử dụng `onFocusChange()`, như đã thảo luận trong mục Trình lắng nghe sự kiện ở trên.

5.4 Menu

Menu là thành phần giao diện người dùng thông dụng trong nhiều loại ứng dụng. Để cung cấp trải nghiệm người dùng quen thuộc và nhất quán, bạn nên sử dụng các API `Menu` để biểu diễn action của người dùng và những tùy chọn khác trong các activity của mình.

Kể từ Android 3.0 (API Cấp 11), các thiết bị nền Android không còn bắt buộc phải có một button *Menu* nữa. Với sự thay đổi này, các ứng dụng Android nên thoát khỏi sự phụ thuộc vào bảng menu 6 mục mà thay vào đó cung cấp một action bar để thể hiện những action thông thường của người dùng.

Mặc dù thiết kế và trải nghiệm người dùng cho các mục của menu đã thay đổi, nhưng những yếu tố căn bản để định nghĩa một tập action và tùy chọn vẫn dựa trên các API `Menu`. Hướng dẫn này chỉ ra cách tạo ba loại menu cơ bản hoặc biểu diễn các action trên mọi phiên bản của Android:

Menu tùy chọn và action bar

Menu tùy chọn (Options menu) là tập các mục menu chính cho một activity. Đây là nơi bạn nên đặt các action có ảnh hưởng trên toàn ứng dụng, chẳng hạn như "Search" (tìm kiếm), "Compose email" (soạn thảo e-mail) và "Settings" (chỉnh sửa các thiết lập).

Nếu bạn đang phát triển cho Android 2.3 hay cấp thấp hơn, người dùng có thể hiển thị bảng menu tùy chọn tùy chọn bằng cách nhấn button *Menu*.

Từ phiên bản 3.0 trở đi, các mục trên menu tùy chọn được hiển thị trên action bar bao gồm các action trên màn hình và các hành động ẩn (overflow option - người dùng chạm vào một biểu tượng ở cuối action bar để hiển thị những action ẩn này). Kể từ Android 3.0, button *Menu* dần bị loại bỏ (một số thiết bị không có button này). Do vậy, bạn nên hướng tới sử dụng action bar để cung cấp cách thức truy cập tới những action và tùy chọn khác.

Xem mục "Tạo một menu tùy chọn".

Menu ngữ cảnh và chế độ contextual action

Menu ngữ cảnh (context menu) là một menu động (floating menu) xuất hiện khi người dùng nhấn lâu trên trên một phần tử. Menu này cung cấp các action ảnh hưởng tới nội dung hoặc khung ngữ cảnh được chọn.

Khi phát triển ứng dụng cho Android 3.0 và cấp cao hơn, bạn nên sử dụng chế độ contextual action (contextual action mode) để kích hoạt các action trên phần nội dung được chọn. Chế độ này hiển thị những mục action ảnh hưởng tới nội dung được chọn trong một thanh ở phía trên màn hình và cho phép người dùng chọn cùng lúc nhiều mục.

Xem mục "Tạo menu ngữ cảnh".

Menu popup

Menu popup hiển thị một danh sách các mục theo dạng dọc ngay trên view kích hoạt menu đó. Menu cung cấp dùng để cung cấp nhiều action liên quan tới nội dung cụ thể hoặc để cung cấp thêm tùy chọn cho một lệnh. Các action trong một menu popup **không** ảnh hưởng trực tiếp tới nội dung tương ứng - giống như các action trên menu ngữ cảnh. Thay vào đó, menu popup cung cấp những action mở rộng liên quan tới các vùng nội dung trong activity của bạn.

Xem mục "Tạo menu popup".

Định nghĩa một menu trong XML

Android cung cấp một định dạng XML chuẩn để định nghĩa các mục menu cho tất cả các loại menu. Thay vì xây dựng một menu trong mã nguồn của activity, bạn nên định nghĩa một menu và tất cả các mục của nó trong một file tài nguyên menu XML. Sau đó, bạn có thể điền nạp file XML này dưới dạng một đối tượng `Menu` trong activity.

Bạn nên sử dụng tài nguyên menu vì những lý do sau:

- Tài nguyên menu giúp bạn dễ hình dung cấu trúc menu trong XML hơn.
- Tài nguyên menu giúp tách biệt phần nội dung của menu ra khỏi mã xử lý của ứng dụng.
- Tài nguyên menu cho phép bạn tạo các cấu hình menu thay đổi cho những phiên bản nền tảng, các kích thước màn hình và cấu hình khác nhau bằng cách sử dụng framework tài nguyên ứng dụng (app resources).

Để định nghĩa menu, hãy tạo một file XML bên trong thư mục `res/menu/` của dự án và xây dựng menu với các phần tử dưới đây:

`<menu>`

Định nghĩa một `Menu` là đối tượng chứa cho các mục menu. Một phần tử `<menu>` nên là nút gốc cho file, có thể chứa một hoặc nhiều phần tử `<item>` và `<group>`.

`<item>`

Tạo một `MenuItem` đại diện cho một mục đơn trong menu. Phần tử này có thể chứa một phần tử `<menu>` lồng để tạo một menu con (submenu).

`<group>`

Là phần tử không bắt buộc, định nghĩa một vật chứa ẩn (vô hình) cho các phần tử `<item>`. Nó cho phép bạn phân loại các mục menu để chúng có thể chia sẻ thuộc tính, chẳng hạn như trạng thái active và trạng thái hiển thị (visibility). Để tham khảo thêm thông tin, xem mục "Tạo nhóm menu".

Đây là một menu ví dụ có tên `game_menu.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

Phần tử `<item>` hỗ trợ vài thuộc tính mà bạn có thể dùng để định nghĩa giao diện và hành vi cho một mục. Các mục trong menu trên bao gồm những thuộc tính sau:

`android:id`

ID tài nguyên duy nhất đối với một mục, cho phép ứng dụng nhận diện mục khi người dùng chọn nó.

`android:icon`

Tham chiếu tới một drawable để sử dụng biểu tượng cho mục.

`android:title`

Tham chiếu tới một chuỗi để sử dụng làm tiêu đề cho mục.

`android:showAsAction`

Xác định thời điểm và cách thức mà mục này xuất hiện dưới dạng một mục action trong action bar.

Đây là những thuộc tính quan trọng nhất mà bạn nên sử dụng, nhưng ngoài ra còn có sẵn nhiều thuộc tính khác. Để biết thêm thông tin về tất cả các thuộc tính được hỗ trợ, xem tài liệu "Menu Resource" ("Tài nguyên menu").

Bạn có thể thêm một menu con vào một mục trong bất cứ menu nào (ngoại trừ trong một menu con) bằng cách thêm một phần tử `<menu>` dưới dạng là con của `<item>`. Các menu con rất hữu ích khi ứng dụng có nhiều chức năng có thể được tổ chức theo chủ đề như các mục trong thanh menu của ứng dụng máy tính (File, Edit, View,...). Ví dụ:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
       <!-- menu con của file -->
       <menu>
          <item android:id="@+id/create_new"
                android:title="@string/create_new" />
```

```
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Để sử dụng menu trong activity, bạn phải chuyển đổi tài nguyên XML thành một đối tượng có thể lập trình được bằng cách sử dụng `MenuInflater.inflate()`. Trong các mục tiếp theo, bạn sẽ thấy cách tạo một đối tượng menu dựa trên file XML cho từng loại menu.

Tạo một menu tùy chọn

Hình 1. Menu tùy chọn trong Trình duyệt, trên Android 2.3.

Menu tùy chọn là nơi chứa các action và tùy chọn khác liên quan tới ngữ cảnh activity hiện tại, chẳng hạn như "Search", "Compose email" và "Settings".

Vị trí để các mục trong menu tùy chọn xuất hiện trên màn hình tùy chọn sẽ phụ thuộc vào phiên bản mà bạn đang phát triển ứng dụng:

- Nếu bạn phát triển ứng dụng cho **Android 2.3.x (API Cấp 10) hay cấp thấp hơn**, nội dung của menu tùy chọn sẽ xuất hiện ở phần dưới màn hình khi người dùng nhấn button Menu, như minh họa ở Hình 1. Khi menu tùy chọn được mở ra, đầu tiên menu chứa biểu tượng của sáu mục menu. Nếu menu chứa quá sáu mục,

Android sẽ đặt mục thứ sau và những mục menu còn lại vào menu tràn (overflow menu) mà tại đó, người dùng có thể mở bằng cách chọn More.

- Nếu bạn phát triển ứng dụng cho **Android 3.0 (API Cấp 11) và cấp cao hơn**, các mục từ menu tùy chọn đều sẵn có trong action bar. Theo mặc định, hệ thống đặt tất cả các action đều bị ẩn trong một biểu tượng ở bên phải của action bar và người dùng có thể hiển thị chúng bằng cách bấm vào biểu tượng này (hoặc bằng cách nhấn button *Menu* của thiết bị, nếu có). Để cho phép truy cập nhanh vào những action quan trọng, bạn có thể tăng cấp vài mục xuất hiện trong action bar bằng cách thêm `android:showAsAction="ifRoom"` vào các phần tử `<item>` tương ứng (xem Hình 2).

  Để biết thêm thông tin về các mục action và những hành vi khác của action bar, xem hướng dẫn Action Bar.

> **Ghi chú**: Ngay cả khi *không* phát triển ứng dụng cho Android 3.0 và cấp cao hơn, bạn cũng có thể xây dựng layout cho action bar của mình với hiệu ứng tương tự. Để tìm hiểu ví dụ về cách bạn có thể hỗ trợ các phiên bản cũ hơn của Android với action bar, xem ví dụ mẫu trong tài liệu mẫu "Action Bar Compatibility" ("Tương thích Action Bar").

Hình 2. Action bar từ ứng dụng Honeycomb Gallery hiển thị các tab điều hướng và một mục action camera (cùng với đó là biểu tượng chứa các hành động ẩn).

Bạn có thể khai báo các mục cho menu tùy chọn từ lớp con `Activity` hoặc lớp con `Fragment`. Nếu cả activity lẫn những phân vùng (được tạo bởi lớp Fragment) đều khai báo các mục cho menu tùy chọn, chúng sẽ được kết hợp trong giao diện người dùng. Các mục của activity xuất hiện trước tiên, theo sau là những mục thuộc từng phân vùng theo thứ tự mà mỗi phân vùng được thêm vào activity. Nếu cần, bạn có thể sắp xếp lại các mục menu với thuộc tính `android:orderInCategory` trong từng `<item>` bạn cần phải di chuyển thứ tự.

Để ấn định menu tùy chọn cho một activity, hãy ghi đè phương thức `onCreateOptionsMenu()` (các phân vùng tự cung cấp phương thức callback `onCreateOptionsMenu()` của chúng). Trong phương thức này, bạn có thể sử dụng tài nguyên menu (được định nghĩa trong file XML) để tạo Menu. Ví dụ:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Bạn cũng có thể thêm các mục menu bằng cách sử dụng `add()` và truy xuất các mục menu với `findItem()` để cập nhật thuộc tính của chúng, sử dụng lớp `MenuItem`.

Nếu bạn phát triển ứng dụng cho Android 2.3.x và cấp thấp hơn, hệ thống sẽ gọi phương thức `onCreateOptionsMenu()` để tạo menu tùy chọn khi người dùng mở menu lần đầu. Còn nếu bạn phát triển ứng dụng cho Android 3.0 và cấp cao hơn, hệ thống sẽ gọi `onCreateOptionsMenu()` khi khởi động activity để hiển thị các mục lên action bar.

Xử lý các sự kiện click

Khi người dùng chọn một mục từ menu tùy chọn (gồm các mục action trong action bar), hệ thống sẽ gọi phương thức `onOptionsItemSelected()` của activity. Phương thức này truyền `MenuItem` đã chọn. Bạn có thể nhận diện mục bằng cách gọi phương thức `getItemId()` để trả về ID duy nhất cho mục menu (xác định bởi thuộc tính `android:id` trong tài nguyên menu hoặc với một số nguyên được truyền cho phương thức `add()`). Bạn có thể khớp ID này với các mục menu đã có để thực hiện action thích hợp. Ví dụ:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Xử lý việc lựa chọn mục menu
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Khi bạn xử lý thành công một mục menu, hãy trả về `true`. Nếu không xử lý mục menu, bạn nên gọi phương thức `onOptionsItemSelected()` của lớp cha (mặc định phương thức này trả về false).

Nếu activity của bạn bao gồm các phân vùng thì trước tiên, hệ thống sẽ gọi `onOptionsItemSelected()` của activity rồi của từng phân vùng (theo thứ tự mà

mỗi phân vùng được thêm vào) tới khi có một giá trị `true` được trả về hoặc tất cả các phân vùng đều đã được gọi.

> **Mách nhỏ:** Android 3.0 bổ sung khả năng cho phép bạn định nghĩa hành vi on-click cho một mục menu trong XML, sử dụng thuộc tính `android:onClick`. Giá trị cho thuộc tính phải là tên phương thức do activity sử dụng menu này định nghĩa. Phương thức phải là public và chấp nhận một tham số `MenuItem` đơn - khi hệ thống gọi phương thức này, nó sẽ truyền mục menu đã chọn. Để tham khảo thêm thông tin và ví dụ, xem tài liệu "Menu Resource" ("Tài nguyên menu").
>
> **Mách nhỏ:** Nếu ứng dụng chứa nhiều activity và một vài trong số chúng cung cấp menu tùy chọn giống nhau, hãy cân nhắc việc tạo một activity không thực thi gì, ngoại trừ các phương thức `onCreateOptionsMenu()` và `onOptionsItemSelected()`. Sau đó, hãy mở rộng lớp này cho mỗi activity mà có menu tùy chọn giống nhau đó. Bằng cách này, bạn có thể quản lý một tập mã nguồn cho việc xử lý những action menu và mỗi lớp con cháu kế thừa các hành vi menu. Nếu bạn muốn thêm các mục menu vào một trong những activity con cháu, hãy ghi đè `onCreateOptionsMenu()` trong activity đó. Gọi `super.onCreateOptionsMenu(menu)` để các mục menu gốc (của lớp cha) được tạo, rồi thêm những mục menu mới với `menu.add()`. Bạn cũng có thể ghi đè hành vi của lớp cha cho từng mục menu riêng lẻ.

Thay đổi mục menu vào thời điểm thực thi

Sau khi hệ thống gọi `onCreateOptionsMenu()`, hệ thống sẽ duy trì một thể hiện của `Menu` bạn tạo và không gọi lại `onCreateOptionsMenu()` nữa, trừ phi menu bị mất hiệu lực vì một số lý do. Tuy nhiên, bạn chỉ sử dụng phương thức `onCreateOptionsMenu()` để tạo trạng thái menu ban đầu và không thay đổi trong suốt vòng đời của activity đó.

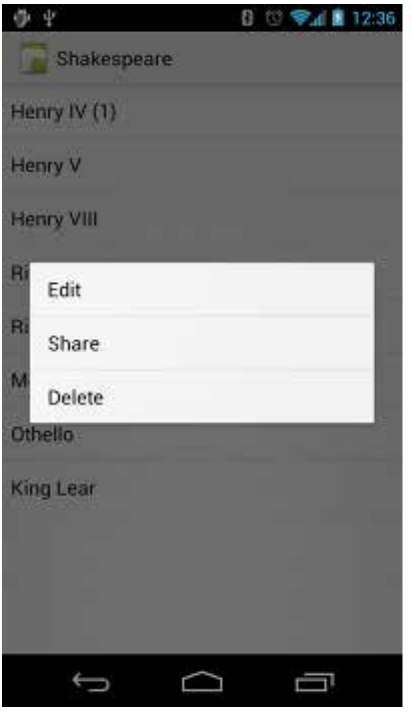
Nếu muốn chỉnh sửa menu tùy chọn dựa trên các sự kiện xảy ra trong suốt vòng đời của activity, bạn có thể thực hiện điều này bên trong phương thức `onPrepareOptionsMenu()`. Phương thức này truyền đối tượng `Menu` đang tồn tại để bạn có thể chỉnh sửa, chẳng hạn như thêm (add), loại bỏ (remove) hay vô hiệu hóa (disable) các mục. (Các phân vùng cũng cung cấp một phương thức callback `onPrepareOptionsMenu()`).

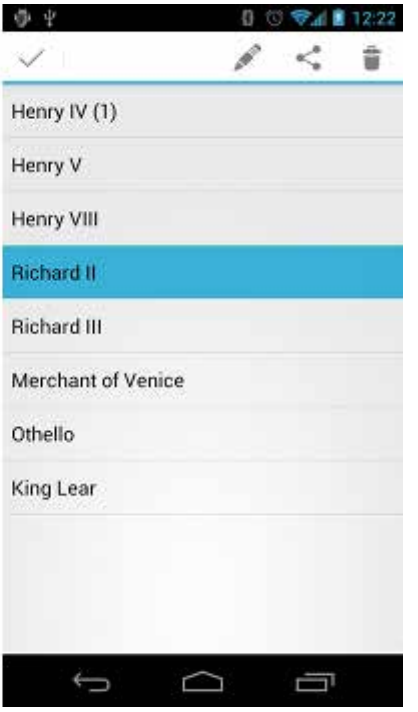
Trên Android 2.3.x và cấp thấp hơn, hệ thống gọi `onPrepareOptionsMenu()` mỗi lần người dùng mở menu tùy chọn (nhấn button *Menu*).

Trên Android 3.0 và cấp cao hơn, menu tùy chọn được xem như luôn mở khi các mục menu được hiển thị trong action bar. Khi một sự kiện xảy ra và bạn muốn cập nhật menu, bạn phải gọi `invalidateOptionsMenu()` để yêu cầu hệ thống gọi `onPrepareOptionsMenu()`.

> **Ghi chú:** Bạn không nên thay đổi các mục trong menu tùy chọn dựa trên `View` hiện đang focus. Tại chế độ chế độ cảm ứng (khi người dùng đang không sử dụng bi lăn hoặc bàn điều hướng D-pad), các view không thể được focus, do đó bạn không nên dùng focus làm cơ sở để chỉnh sửa các mục trong menu tùy chọn. Nếu bạn muốn cung cấp các mục menu ngữ cảnh cho một `View`, hãy sử dụng menu ngữ cảnh (Context Menu).

Tạo menu ngữ cảnh





Hình 3. Ảnh chụp màn hình của một menu ngữ cảnh động (trái) và một action bar ngữ cảnh (phải).

Một menu ngữ cảnh cung cấp các action ảnh hưởng tới một mục hoặc khung ngữ cảnh cụ thể trong giao diện người dùng. Bạn có thể cung cấp một menu ngữ cảnh cho bất cứ view nào, nhưng chúng thường được sử dụng cho các mục trong một `ListView`, `GridView` hoặc những tập view khác mà tại đó, người dùng có thể thực hiện action trực tiếp lên từng mục.

Hai cách cung cấp các action theo ngữ cảnh:

- Trong một menu ngữ cảnh động (floating context menu). Một menu xuất hiện dưới dạng danh sách động bao gồm các mục menu (tương tự như hộp thoại) khi người dùng thực hiện một sự kiện long-click (nhấn và giữ) trên view khai báo hỗ trợ cho menu ngữ cảnh. Người dùng có thể thực hiện một action ngữ cảnh trên một mục tại một thời điểm.
- Trong chế độ action ngữ cảnh (contextual action mode). Chế độ này là thực thi hệ thống của `ActionMode` hiển thị một *action bar ngữ cảnh* ở phía trên màn hình với các mục action ảnh hưởng tới những mục được chọn. Khi chế độ này hoạt động, người dùng có thể thực hiện một action trên nhiều mục cùng lúc (nếu ứng dụng của bạn cho phép điều này).

> **Ghi chú:** Chế độ action ngữ cảnh có trên Android 3.0 (API Cấp 11) và cấp cao hơn. Kỹ thuật này thường được dùng để hiển thị các action ngữ cảnh khi đã sẵn sàng. Nếu ứng dụng của bạn hỗ trợ các phiên bản cấp thấp hơn 3.0, bạn nên quay lại với menu ngữ cảnh động trên những thiết bị đó.

Tạo menu ngữ cảnh động

Để tạo một menu ngữ cảnh động:

1. Đăng ký `View` với đối tượng mà menu ngữ cảnh nên được liên kết bằng cách gọi `registerForContextMenu()` và truyền `View` cho phương thức.

   Nếu activity của bạn sử dụng một `ListView` hoặc `GridView` và bạn muốn mỗi mục cung cấp menu ngữ cảnh giống nhau, hãy đăng ký tất cả các mục cho một menu ngữ cảnh bằng cách truyền `ListView` hoặc `GridView` cho `registerForContextMenu()`.

2. Thực thi phương thức `onCreateContextMenu()` trong `Activity` hoặc trong `Fragment`.

   Khi view mà bạn đăng ký nhận một sự kiện long-click, hệ thống sẽ gọi phương thức `onCreateContextMenu()`. Đây là nơi bạn định nghĩa các mục menu, thường là bằng cách sử dụng một tài nguyên menu. Ví dụ:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

   `MenuInflater` cho phép bạn tạo menu ngữ cảnh từ một tài nguyên menu (menu resource). Các tham số của phương thức callback chứa View mà người dùng đã chọn và một đối tượng `ContextMenu.ContextMenuInfo` cung cấp thêm thông tin về mục được chọn. Nếu activity của bạn có vài `view` mà mỗi view lại cung cấp một menu ngữ cảnh khác nhau, bạn có thể sử dụng những tham số đó để quyết định nên tạo menu ngữ cảnh nào từ tài nguyên menu nào.

3. Thực thi `onContextItemSelected()`.

   Khi người dùng chọn một mục menu, hệ thống gọi phương thức này để bạn có thể thực hiện action thích hợp. Ví dụ:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info =
        (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
```

```
            editNote(info.id);
return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Phương thức `getItemId()` truy vấn ID cho mục menu được chọn, đây là thứ bạn nên gán cho từng mục menu trong XML bằng cách sử dụng thuộc tính `android:id`, như đã trình bày trong mục "Định nghĩa một menu trong XML" ("Defining a Menu in XML").

Khi bạn xử lý thành công một mục menu, hãy trả về `true`. Nếu không xử lý mục menu, bạn nên truyền mục menu cho phương thức xử lý của lớp cha. Nếu activity của bạn chứa các phân vùng, activity sẽ nhận callback này trước tiên. Bằng cách gọi lớp cha khi không xử lý, hệ thống truyền sự kiện cho phương thức callback tương ứng trong từng phân vùng, mỗi thời điểm một sự kiện (theo thứ tự mà từng phân vùng được thêm vào) cho tới khi trả về `true` hoặc `false`. (Phương thức mặc định của `Activity` và `android.app.Fragment` trả về `false`, do vậy lời khuyên là bạn nên gọi lớp cha thường xuyên khi không xử lý).

Sử dụng chế độ action ngữ cảnh

Chế độ action ngữ cảnh là thực thi hệ thống của `ActionMode` tập trung vào tương tác của người dùng để thực hiện các action ngữ cảnh. Khi người dùng bật chế độ này bằng cách chọn một mục, một *action bar ngữ cảnh* xuất hiện ở phần trên của màn hình để hiển thị những action mà người dùng có thể thực hiện trên mục đang được chọn. Trong khi chế độ này được bật, người dùng có thể chọn nhiều mục (nếu bạn cho phép), bỏ chọn các mục, đồng thời tiếp tục điều hướng trong phạm vi activity (bạn có thể cho phép số lượng mục tùy ý). Chế độ action bị vô hiệu hóa (disabled) và action bar ngữ cảnh biến mất khi người dùng không chọn một mục nào, nhấn button BACK, hoặc chọn action *Done* ở bên trái action bar ngữ cảnh.

> **Ghi chú:** Action bar ngữ cảnh không nhất thiết phải liên kết với action bar. Chúng hoạt động độc lập, bất kể action bar ngữ cảnh chiếm vị trí hiển thị của action bar.

Nếu phát triển ứng dụng cho Android 3.0 (API Cấp 11) hoặc cấp cao hơn, bạn nên thường xuyên sử dụng chế độ action ngữ cảnh để biểu diễn các action theo ngữ cảnh thay vì sử dụng menu ngữ cảnh động (floating context menu).

Đối với các view cung cấp action ngữ cảnh, bạn nên thường xuyên kích hoạt chế độ action ngữ cảnh theo một trong hai sự kiện (hoặc cả hai):

- Người dùng thực hiện một long-click trên view.
- Người dùng chọn một checkbox hoặc thành phần giao diện tương tự bên trong view.

Cách ứng dụng kích hoạt chế độ action ngữ cảnh và xác định hành vi cho mỗi action tùy thuộc vào thiết kế của bạn. Có hai thiết kế cơ bản:

- Thiết kế cho những action theo ngữ cảnh trong các view độc lập, tùy ý.
- Thiết kế cho tập các action ngữ cảnh trên những nhóm mục của một `ListView` hoặc `GridView` (cho phép người dùng chọn nhiều mục và thực hiện một action trên tất cả các mục).

Các mục nội dung tiếp theo sẽ mô tả thiết lập cần thiết cho từng kịch bản.

Bật chế độ action ngữ cảnh cho từng view riêng biệt

Nếu muốn kích hoạt chế độ action ngữ cảnh chỉ khi người dùng chọn các view cụ thể, bạn nên:

1. Thực thi giao diện `ActionMode.Callback`. Trong các phương thức callback, bạn có thể xác định action cho action bar ngữ cảnh, phản hồi sự kiện click trên các mục action, đồng thời xử lý những sự kiện liên quan đến vòng đời của chế độ action(Action Mode).
2. Gọi `startActionMode()` nếu bạn muốn hiển thị action bar (chẳng hạn như khi người dùng nhấn lâu vào view).

Ví dụ:

1. Thực thi giao diện `ActionMode.Callback`:

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback()
{
    // Được gọi khi tạo chế độ action; startActionMode() được gọi
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Sử dụng tài nguyên menu để tạo các mục của menu ngữ cảnh
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Được gọi mỗi khi hiển thị chế độ action. Luôn luôn được gọi
    // sau onCreateActionMode nhưng có thể được gọi nhiều lần nếu
    // chế độ không còn hiệu lực.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Trả về false nếu không làm gì
    }

    // Được gọi khi người dùng chọn một mục menu ngữ cảnh
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item)
```

```
{
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish(); // Action đã được chọn, sau đó đóng CAB
                return true;
            default:
                return false;
        }
    }

    // Được gọi khi người dùng thoát khỏi chế độ action
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};
```

Lưu ý, những callback của sự kiện này gần như giống hệt các callback cho menu tùy chọn (options menu), ngoại trừ mỗi callback trong đó lại truyền đối tượng `ActionMode` liên kết với sự kiện. Bạn có thể sử dụng các API `ActionMode` để tạo nhiều thay đổi cho action bar ngữ cảnh (CAB), chẳng hạn như xem lại tiêu đề (title) và phụ đề (subtitle) với `setTitle()` và `setSubtitle()` (rất hữu ích trong việc chỉ ra số lượng mục được chọn).

Cũng cần lưu ý rằng ví dụ trên đã đặt biến (variable) `mActionMode` là null khi chế độ action bị hủy. Ở bước tiếp theo, bạn sẽ thấy việc biến này được khởi tạo và đang lưu biến thành viên trong activity hoặc phân vùng đem lại lợi ích như thế nào.

2. Gọi `startActionMode()` để bật chế độ action ngữ cảnh lúc thích hợp, chẳng hạn như để phản hồi một long-click trên một `View`:

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Được gọi khi người dùng nhấn lâu trên một số View
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Khởi động CAB bằng cách sử dụng ActionMode. Callback đã được
        // xác định ở trên
        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

Khi bạn gọi phương thức `startActionMode()`, hệ thống trả về `ActionMode` được tạo. Bằng cách lưu nó trong một biến thành viên (member variable), bạn có thể thay đổi action bar ngữ cảnh để phản hồi lại những sự kiện khác. Trong ví dụ trên, `ActionMode` được sử dụng để đảm bảo rằng thể hiện `ActionMode` không được tái tạo nếu nó đã hoạt động bằng cách kiểm tra xem biến thành viên có null không trước khi bắt đầu chế độ action .

Bật một tập action ngữ cảnh trong một ListView hoặc GridView

Nếu có một tập các mục trong `ListView` hay `GridView` (hoặc mở rộng khác của `AbsListView`) và muốn cho phép người dùng thực hiện tập action, bạn nên:

- Thực thi giao diện `AbsListView.MultiChoiceModeListener` và thiết lập nó cho nhóm view với `setMultiChoiceModeListener()`. Trong các phương thức callback của trình lắng nghe, bạn có thể xác định các action cho action bar ngữ cảnh, phản hồi những sự kiện click trên các mục action, đồng thời xử lý các callback khác thừa kế từ giao diện `ActionMode.Callback`.
- Gọi `setChoiceMode()` với đối số (argument) `CHOICE_MODE_MULTIPLE_MODAL`.

Ví dụ:

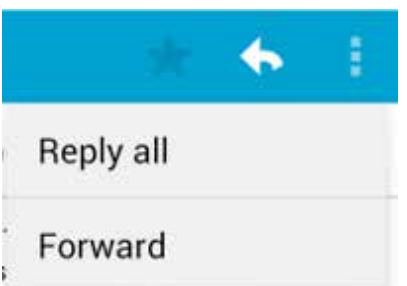
```
ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
                                          long id, boolean checked) {
        // Tại đây, bạn có thể làm gì đó khi các mục được chọn/bỏ chọn,
        // chẳng hạn như cập nhật tiêu đề trong CAB (Action bar ngữ cảnh)
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item)
{
        // Phản hồi sự kiện click trên các action trong CAB
        switch (item.getItemId()) {
            case R.id.menu_delete:
                deleteSelectedItems();
                mode.finish(); // Action đã được chọn, sau đó đóng CAB
                return true;
            default:
                return false;
        }
    }

    @Override
```

```
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Tạo menu cho CAB
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context, menu);
        return true;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        // Tại đây, bạn có thể thực hiện các cập nhật cần thiết cho
        // activity khi CAB bị loại bỏ. Theo mặc định, các mục được
        // chọn đều bị bỏ chọn chưa được chọn (unchecked).
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        // Tại đây, bạn có thể thực hiện cập nhật cho CAB nhờ một yêu
        // cầu (request) invalidate()
        return false;
    }
});
```

Như vậy đấy! Lúc này, khi người dùng nhấn lâu trên một mục, hệ thống sẽ gọi phương thức `onCreateActionMode()` và hiển thị action bar ngữ cảnh với những action cụ thể. Trong lúc action bar ngữ cảnh còn hiển thị, người dùng có thể chọn thêm các mục khác.

Trong một số trường hợp mà tại đó, các action ngữ cảnh cung cấp những mục action thông dụng, có thể bạn sẽ muốn thêm một checkbox hoặc một phần tử giao diện tương tự cho phép người dùng lựa chọn các mục, bởi vì có thể họ không biết đến việc sử dụng thao tác nhấn lâu. Khi người dùng chọn checkbox, bạn có thể kích hoạt chế độ action ngữ cảnh bằng cách thiết lập mục danh sách tương ứng với trạng thái đã chọn (checked) bằng `setItemChecked()`.

Tạo menu popup



Hình 4. Một menu popup trong ứng dụng Gmail, được gắn vào một biểu tượng ở phía bên phải màn hình (nhấn vào để hiển thị các action ẩn).

PopupMenu là menu được neo vào một View. Loại menu này xuất hiện bên dưới view neo nếu còn chỗ, hoặc bên trên view đó nếu ngược lại. PopupMenu rất hữu ích trong việc:

- Cung cấp một menu có mục menu ẩn cho các action *liên quan tới* nội dung cụ thể (chẳng hạn như các header e-mail của Gmail, như minh họa ở Hình 4).

> **Ghi chú:** Popup menu không giống như một menu ngữ cảnh thường dùng cho các action *ảnh hưởng* tới nội dung được chọn. Đối với những action ảnh hưởng tới nội dung được chọn, sử dụng chế độ action ngữ cảnh hoặc menu ngữ cảnh động.

- Cung cấp phần thứ hai của câu lệnh (chẳng hạn như một button đánh dấu "Add" sẽ tạo một menu popup với các tùy chọn "Add" khác nhau).
- Cung cấp một danh sách xổ xuống (drop-down list) tương tự với Spinner không lưu giữ lâu lựa chọn của nó.

Ghi chú: PopupMenu luôn sẵn sàng đối với API Cấp 11 và cấp cao hơn.

Nếu bạn định nghĩa menu trong file XML, đây là cách bạn có thể hiển thị menu popup:

1. Tạo một thể hiện của PopupMenu phương thức khởi tạo của nó, phương thức này sẽ lấy Context (ngữ cảnh) của ứng dụng hiện tại và View mà menu nên neo vào.
2. Sử dụng MenuInflater để điền tài nguyên menu cho đối tượng Menu (đối tượng này được trả về bởi phương thức PopupMenu.getMenu(). Trên API Cấp 14 và cấp cao hơn, bạn có thể dùng PopupMenu.inflate() để thay thế.
3. Gọi phương thức PopupMenu.show().

Ví dụ, đây là một button với thuộc tính android:onClick mà sẽ hiển thị một menu popup:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

Sau đó, activity có thể hiển thị menu popup như sau:

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

Trong API Cấp 14 và cấp cao hơn, bạn có thể kết hợp hai dòng điền menu với PopupMenu.inflate().

Menu bị hủy (dismiss) khi người dùng chọn một mục hoặc chạm vào bên ngoài khu vực menu. Bạn có thể lắng nghe sự kiện hủy menu bằng cách sử dụng `PopupMenu.OnDismissListener`.

Xử lý các sự kiện click

Để thực hiện một action khi người dùng chọn một mục menu, bạn phải thực thi giao diện `PopupMenu.OnMenuItemClickListener` và đăng ký nó với `PopupMenu` bằng cách gọi `setOnMenuItemclickListener()`. Khi người dùng chọn một mục, hệ thống gọi phương thức callback `onMenuItemClick()` trong giao diện của bạn.

Ví dụ:

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // Activity này thực thi OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

Tạo nhóm menu

Nhóm menu là tập các mục menu có chung một số đặc điểm nhất định. Với một nhóm menu, bạn có thể:

- Hiển thị hoặc ẩn tất cả các mục với `setGroupVisible()`.
- Bật hoặc vô hiệu hóa tất cả các mục với `setGroupEnabled()`.
- Cho phép/không cho phép người dùng chọn các mục với `setGroupCheckable()`.

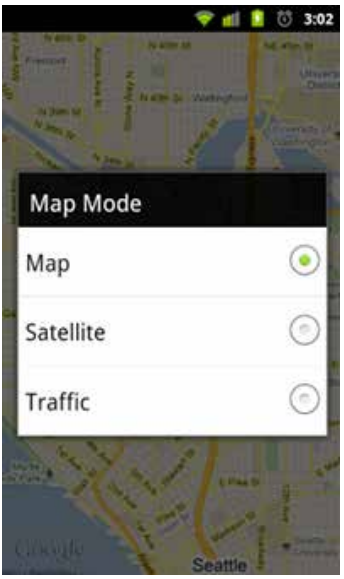
Bạn có thể tạo một nhóm bằng cách lồng các phần tử `<item>` bên trong một phần tử `<group>` trong tài nguyên menu, hoặc bằng cách ấn định một ID nhóm với phương thức `add()`.

Dưới đây là ví dụ về tài nguyên menu có chứa một nhóm:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/menu_save"
          android:title="@string/menu_save" />
    <!-- nhóm menu -->
    <group android:id="@+id/group_delete">
        <item android:id="@+id/menu_archive"
              android:title="@string/menu_archive" />
        <item android:id="@+id/menu_delete"
              android:title="@string/menu_delete" />
    </group>
</menu>
```

Các mục trong nhóm xuất hiện ở cùng cấp với mục đầu tiên - cả ba mục trong menu đều là anh em (sibling) của nhau. Tuy nhiên, bạn có thể chỉnh sửa các đặc trưng của hai mục trong nhóm bằng cách tham chiếu ID nhóm và sử dụng những phương thức liệt kê ở trên. Hệ thống cũng không bao giờ chia tách các mục đã được nhóm. Ví dụ, nếu bạn khai báo `android:showAsAction="ifRoom"` cho mỗi mục, chúng sẽ xuất hiện trong action bar hoặc trong biểu tượng chứa các action ẩn.

Sử dụng các mục menu có thể đánh dấu



Hình 5. Ảnh chụp màn hình của một menu con với các mục có thể đánh dấu.

Đôi khi, việc một menu có giao diện hỗ trợ khả năng lựa chọn mục menu bằng cách bật hoặc tắt sẽ rất hữu ích, bạn hãy sử dụng checkbox khi mỗi mục được lựa chọn độc lập với nhau hoặc dùng các radio button khi các mục thuộc một nhóm và chỉ cho phép lựa chọn một mục. Hình 5 minh họa một menu con với những mục có thể đánh dấu được với các radio button.

> **Ghi chú:** Các mục menu trong Icon Menu (từ menu tùy chọn) không thể hiển thị một checkbox hoặc radio button. Nếu chọn cách tạo các mục trong Icon Menu là có thể đánh dấu, bạn phải tự biểu thị trạng thái đã chọn bằng cách hoán đổi biểu tượng và/hoặc văn bản mỗi lần trạng thái thay đổi.

Bạn có thể định nghĩa khả năng có thể đánh dấu cho từng mục menu riêng bằng cách sử dụng thuộc tính `android:checkable` trong phần tử `<item>`, hoặc cho cả nhóm với thuộc tính `android:checkableBehavior` trong phần tử `<group>`. Ví dụ, tất cả các mục trong nhóm menu của đoạn mã dưới đây đều có thể đánh dấu được với một radio button:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
              android:title="@string/red" />
        <item android:id="@+id/blue"
              android:title="@string/blue" />
    </group>
</menu>
```

Thuộc tính `android:checkableBehavior` chấp nhận cả hai giá trị:

`single`

Chỉ một mục của nhóm có thể được chọn (button radio).

`all`

Tất cả các mục có thể được chọn (checkbox).

`none`

Không thể chọn bất cứ mục nào.

Bạn có thể áp dụng trạng thái đã chọn mặc định cho một mục bằng cách sử dụng thuộc tính `android:checked` trong phần tử `<item>`, đồng thời thay đổi trạng thái này trong mã nguồn với phương thức `setChecked()`.

Khi một mục có thể đánh dấu được chọn, hệ thống sẽ gọi phương thức callback tương ứng với mục đó (chẳng hạn như `onOptionsItemSelected()`). Phương thức này là nơi bạn phải thiết lập trạng thái của checkbox, do checkbox hoặc radio button không tự động thay đổi trạng thái của nó được. Bạn có thể truy vấn trạng thái hiện tại của mục (như trước khi người dùng chọn nó) với `isChecked()` và thiết lập trạng thái đã chọn với `setChecked()`. Ví dụ:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.vibrate:
        case R.id.dont_vibrate:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Nếu bạn không thiết lập trạng thái đã chọn bằng cách này thì, trạng thái hiển thị của mục (checkbox hoặc button radio) sẽ không đổi khi người dùng chọn nó. Khi bạn thiết lập trạng thái, activity sẽ bảo toàn trạng thái đã chọn của mục; do đó, khi người dùng mở menu lần sau, trạng thái đã chọn mà bạn thiết lập sẽ hiển thị.

> **Ghi chú:** Các mục menu có thể đánh dấu thường được sử dụng chỉ dựa trên cơ sở theo phiên (session) và không được lưu sau khi tắt ứng dụng. Nếu có các thiết lập ứng dụng muốn lưu cho người dùng, bạn nên lưu trữ dữ liệu bằng cách dùng Các thiết lập chia sẻ (Shared Preferences).

Thêm mục menu dựa trên một Intent

Đôi khi, bạn muốn có một mục menu để khởi động một activity dùng `Intent` (bất kể đó là một activity trong ứng dụng của bạn hay ứng dụng khác). Khi bạn biết intent muốn dùng và mục menu cụ thể sẽ kích hoạt intenet đó, bạn có thể khởi động intent với `startActivity()` trong phương thức callback trên mục được chọn thích hợp (chẳng hạn như callback `onOptionsItemSelected()`).

Tuy nhiên, nếu bạn không chắc chắn rằng thiết bị của người dùng chứa một ứng dụng có thể xử lý intent, hãy thêm một mục menu để kích hoạt nó. Điều này sẽ dẫn tới một mục menu không hoạt động, bởi intent có thể không xử lý activity đó. Để giải quyết vấn đề này, Android cho phép bạn thêm động các mục menu khi Android tìm kiếm những activity xử lý intent trên thiết bị.

Để thêm các mục menu dựa trên những activity đang sẵn sàng chấp nhận một intent:

1. Định nghĩa một intent với hạng mục `CATEGORY_ALTERNATIVE` và/hoặc `CATEGORY_SELECTED_ALTERNATIVE`, cùng với bất cứ yêu cầu nào khác.
2. Gọi `Menu.addIntentOptions()`. Sau đó, Android sẽ tìm kiếm bất kỳ ứng dụng nào có thể thực thi intent và thêm chúng vào menu.

Nếu không ứng dụng đã cài đặt nào đáp ứng intent thì không có mục menu nào được thêm vào.

> **Ghi chú:** CATEGORY_SELECTED_ALTERNATIVE được dùng để xử lý phần tử đang được chọn trên màn hình. Vì vậy, chỉ nên sử dụng hạng mục này khi tạo một Menu trong onCreateContextMenu().

Ví dụ:

```
@Override
public boolean onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);

    // Tạo một Intent mô tả những yêu cầu cần thực hiện để được xuất
    // hiện trong menu của chúng ta. Ứng dụng cung cấp phải bao gồm
    // một giá trị hạng mục của Intent.CATEGORY_ALTERNATIVE.
    Intent intent = new Intent(null, dataUri);
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);

    // Tìm kiếm và tạo menu với các ứng dụng cung cấp được chấp nhận.
    menu.addIntentOptions(
        R.id.intent_group,  // Nhóm menu mà các mục mới sẽ được thêm vào
        0,      // ID mục menu duy nhất (none)
        0,      // Sắp xếp các mục (none)
        this.getComponentName(),   // Tên activity hiện tại null,
          // Các mục cụ thể để đưa vào trước (none) intent,
         // Intent đã được tạo ở trên, mô tả các yêu cầu của chúng ta 0,
          // Thêm cờ (flag) cho các mục điều khiển (none) null);
          // Mảng các MenuItem liên quan tới những mục cụ thể (none)

    return true;
}
```

Đối với mỗi activity được tìm thấy cung cấp một bộ lọc intent khớp với intent đã định nghĩa, một mục menu sẽ được thêm vào, sử dụng giá trị trong `android:label` của bộ lọc intent làm tiêu đề cho mục menu, đồng thời dùng biểu tượng của ứng dụng làm biểu tượng cho mục menu. Phương thức addIntentOptions() trả về số lượng mục menu đã thêm vào.

> **Ghi chú:** Khi bạn gọi addIntentOptions(), phương thức này ghi đè lên bất kỳ và tất cả các mục menu thuộc nhóm menu đã được chỉ định trong đối số thứ nhất.

Cho phép activity của bạn được thêm vào các menu khác

Bạn cũng có thể cung cấp các service của activity cho những ứng dụng khác, vậy nên ứng dụng của bạn có thể được bao gồm trong các menu khác (đảo ngược vai trò đã mô tả ở trên).

Để được bao gồm trong các menu của ứng dụng khác, bạn cần định nghĩa một bộ lọc intent như thông lệ, nhưng lưu ý phải bao gồm CATEGORY_ALTERNATIVE và/hoặc các giá trị CATEGORY_SELECTED_ALTERNATIVE cho hạng mục bộ lọc intent. Ví dụ:

```
<intent-filter label="@string/resize_image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```

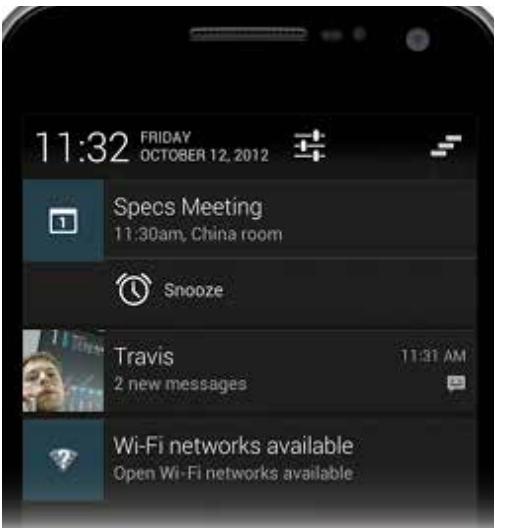
Để tham khảo thông tin về cách tạo các bộ lọc intent, xem mục "Intents and Intent Filters" ("Intent và bộ lọc intent").

Để xem một ứng dụng mẫu sử dụng kỹ thuật này, hãy tìm trong ví dụ mẫu Note Pad.

5.5 Thông báo

Thông báo (notification) là một thông điệp mà bạn có thể hiển thị cho người dùng ở ngoài giao diện người dùng thông thường của ứng dụng. Khi bạn ra lệnh cho hệ thống đưa ra một thông báo, trước hết thông báo sẽ xuất hiện dưới dạng một biểu tượng trong vùng **thông báo (notification area)**. Để xem chi tiết về thông báo, người dùng cần mở **ngăn thông báo (notification drawer)**. Cả vùng thông báo lẫn ngăn thông báo đều là các khu vực được hệ thống kiểm soát mà người dùng có thể xem vào mọi thời điểm.

Hình 1. Các thông báo trong vùng thông báo.



Hình 2. Các thông báo trong ngăn thông báo.

Thiết kế của thông báo

Là một bộ phận quan trọng của giao diện người dùng Android, thông báo có những quy tắc thiết kế riêng của chúng. Để biết cách thiết kế thông báo và tương tác với chúng, hãy đọc chủ đề Hướng dẫn Thiết kế Notifications của Android.

> **Ghi chú:** Ngoại trừ nơi được ghi chú, hướng dẫn này đề cập tới lớp `NotificationCompat.Builder` trong Support Library (thư viện hỗ trợ) phiên bản 4. Lớp `Notification.Builder` được thêm vào trong Android 3.0.

Các phần tử hiển thị thông báo

Thông báo trong ngăn thông báo có thể xuất hiện ở một trong hai kiểu trực quan, phụ thuộc vào phiên bản và trạng thái của ngăn thông báo:

View thông thường

View chuẩn của thông báo trong ngăn thông báo.

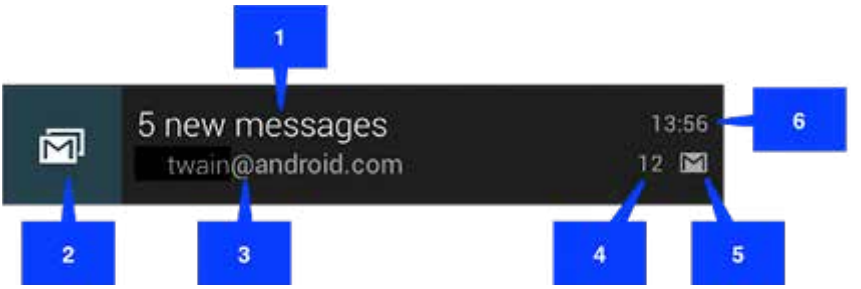
View lớn (big view)

View hiển thị khi thông báo được mở rộng. View lớn là một phần của tính năng thông báo mở rộng có sẵn trong Android 4.1.

Các kiểu này được mô tả ở những mục sau.

View thông thường

Một thông báo trong view thông thường sẽ xuất hiện trong vùng thông báo với chiều cao lên tới 64 dp. Ngay cả khi bạn tạo một thông báo với kiểu view lớn, thông báo vẫn sẽ xuất hiện trong view thông thường cho tới khi được mở rộng. Đây là một ví dụ về view thông thường:



Hình 3. Thông báo trong view thông thường.

Các chú thích trong hình minh họa trên đề cập những nội dung sau:

1. Tiêu đề nội dung (content title).
2. Biểu tượng lớn (large icon).
3. Văn bản nội dung (content text).
4. Thông tin nội dung (content info).

5. Biểu tượng nhỏ (small icon).
6. Thời gian thông báo xuất hiện. Bạn có thể thiết lập một giá trị cụ thể với `setWhen()`; nếu bạn không làm như vậy thì giá trị mặc định sẽ là thời gian hệ thống nhận được thông báo.

View lớn

Một thông báo của view lớn chỉ xuất hiện khi thông báo được mở rộng, điều này xảy ra khi thông báo nằm ở đầu ngăn thông báo, hoặc khi người dùng thao tác mở rộng ngăn thông báo. Các thông báo mở rộng là tính năng sẵn có với Android 4.1.

Ảnh chụp màn hình dưới đây hiển thị một thông báo dạng hộp thư đến (inbox):

Hình 4. Thông báo view lớn.

Lưu ý, view lớn có hầu hết các phần tử như view thông thường. Điểm khác biệt duy nhất là chú thích số 7, vùng chi tiết. Mỗi kiểu view lớn đều thiết lập vùng này theo một cách khác nhau. Các kiểu hiện có là:

Kiểu bức tranh lớn (big picture)

Vùng chi tiết chứa một ảnh bitmap cao tới 256 dp trong phần chi tiết của nó.

Kiểu văn bản lớn (big text)

Hiển thị một khối văn bản lớn trong phần chi tiết.

Kiểu hộp thư đến

Hiển thị các dòng văn bản trong phần chi tiết.

Mọi kiểu view lớn đều có các tùy chọn nội dung mà view thông thường không có, bao gồm:

Tiêu đề nội dung lớn

Cho phép bạn ghi đè tiêu đề nội dung của view thông thường với một tiêu đề chỉ xuất hiện trong view mở rộng.

Văn bản tóm lược (summary text)

Cho phép bạn thêm một dòng văn bản dưới vùng chi tiết.

Việc áp dụng một kiểu view lớn cho một thông báo được mô tả trong mục "Áp dụng kiểu view lớn cho một thông báo".

5.5.1 Tạo thông báo

Bạn có thể xác định thông tin giao diện người dùng và các action cho một thông báo trong một đối tượng `NotificationCompat.Builder`. Để tự tạo một thông báo, bạn gọi `NotificationCompat.Builder.build()`, nó trả về một đối tượng `Notification` chứa các thông số kỹ thuật cho bạn. Để đưa ra thông báo, bạn truyền đối tượng `Notification` cho hệ thống bằng cách gọi `NotificationManager.notify()`.

Những nội dung bắt buộc của một thông báo

Một đối tượng `Notification` *phải* chứa những nội dung sau:

- Một biểu tượng nhỏ, thiết lập bởi `setSmallIcon()`.
- Một tiêu đề, thiết lập bởi `setContentTitle()`.
- Văn bản chi tiết, thiết lập bởi `setContentText()`.

Các thiết lập và nội dung tùy chọn của một thông báo

Tất cả các thiết lập thông báo và nội dung khác đều không bắt buộc. Để tìm hiểu thêm về chúng, xem tài liệu tham khảo về `NotificationCompact.Builder`.

Các action của thông báo

Mặc dù chúng là tùy chọn, song bạn vẫn nên thêm ít nhất một action cho thông báo của mình. Một action cho phép người dùng đi trực tiếp từ thông báo tới một `Activity` trong ứng dụng, đây cũng là nơi họ có thể nhìn vào một hay nhiều sự kiện hoặc làm những việc khác.

Một thông báo có thể cung cấp nhiều action. Điều bạn nên làm là luôn định nghĩa action kích hoạt khi người dùng nhấn vào thông báo; thường thì action này mở một `Activity` trong ứng dụng của bạn. Bạn cũng có thể thêm các button vào thông báo thực hiện các action bổ sung như tăng thời gian cho một báo thức (alarm) hoặc phản hồi lập tức một tin nhắn văn bản (text message); tính năng này có sẵn ở phiên bản Android 4.1. Nếu sử dụng thêm các button action, bạn phải làm cho chức năng của chúng sẵn sàng trong một `Activity` của ứng dụng; xem mục "Handling compatibility" ("Xử lý vấn đề tương thích") để biết thêm thông tin chi tiết.

Trong một `Notification`, bản thân action do một `PendingIntent` định nghĩa chứa một `Intent` khởi động `Activity` của ứng dụng. Để liên kết `PendingIntent` với một thao tác của người dùng, gọi phương thức thích hợp của `NotificationCompat.Builder`. Ví dụ, nếu bạn muốn khởi động `Activity` khi người dùng nhấn vào dòng thông báo trong ngăn thông báo, hãy thêm `PendingIntent` bằng cách gọi `setContentIntent()`.

Khởi động một Activity khi người dùng nhấn vào thông báo là kịch bản action phổ biến nhất. Bạn cũng có thể khởi động một Activity khi người dùng tắt một thông báo. Trong Android 4.1 và các phiên bản mới hơn, bạn có thể khởi động một Activity từ một button action. Để tìm hiểu sâu hơn, hãy đọc hướng dẫn tham khảo cho NotificationCompat.Builder.

Tạo một thông báo đơn giản

Đoạn mã nhỏ dưới đây minh họa một thông báo đơn giản, xác định một activity sẽ mở khi người dùng nhấn vào thông báo. Chú ý rằng, đoạn mã tạo một đối tượng TaskStackBuilder và dùng nó để tạo PendingIntent cho action. Mô hình này được giải thích chi tiết hơn ở mục "Preserving Navigation when Starting an Activity" ("Bảo toàn điều hướng khi khởi động một Activity"):

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
    .setContentText("Hello World!");
// Tạo một intent tường minh cho Activity trong ứng dụng
Intent resultIntent = new Intent(this, ResultActivity.class);

// Đối tượng stack builder sẽ chứa một ngăn xếp lùi (back stack) cho
// Activity đã khởi động.
// Điều này đảm bảo rằng việc duyệt quay trở lại (backward) từ Activity
// sẽ thoát ra khỏi ứng dụng và đến màn hình Home.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Thêm ngăn xếp lùi cho Intent (nhưng không phải bản thân Intent)
stackBuilder.addParentStack(ResultActivity.class);
// Thêm Intent khởi động Activity vào đỉnh của ngăn xếp
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId cho phép bạn cập nhật thông báo về sau.
mNotificationManager.notify(mId, mBuilder.build());
```

Mọi việc đã hoàn tất! Hiện người dùng đã được thông báo.

Áp dụng một kiểu view lớn cho thông báo

Để một thông báo xuất hiện trong view lớn khi nó được mở rộng, trước hết, hãy tạo một đối tượng NotificationCompat.Builder với các tùy chọn view thông thường mà bạn muốn. Tiếp theo, gọi Builder.setStyle() bằng một đối tượng kiểu view lớn với vai trò là đối số của nó.

Nhớ rằng, kiểu thông báo mở rộng không có sẵn trong các nền tảng trước Android 4.1. Để biết cách xử lý các thông báo cho Android 4.1 và những nền tảng trước nó, tham khảo mục "Xử lý vấn đề tương thích".

Ví dụ, đoạn mã nhỏ dưới đây cho thấy cách thay đổi thông báo được tạo trong đoạn mã trước để sử dụng kiểu view lớn dạng hộp thư đến:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Event tracker")
    .setContentText("Events received")
NotificationCompat.InboxStyle inboxStyle =
        new NotificationCompat.InboxStyle();
String[] events = new String[6];
// Thiết lập một tiêu đề cho view lớn có kiểu hộp thư đến
inboxStyle.setBigContentTitle("Event tracker details:");
...
// Chuyển các sự kiện vào view lớn
for (int i=0; i < events.length; i++) {
    inboxStyle.addLine(events[i]);
}
// Chuyển đối tượng kiểu view lớn vào đối tượng thông báo.
mBuilder.setStyle(inBoxStyle);
...
// Đưa ra thông báo ở đây.
```

Xử lý vấn đề tương thích

Không phải mọi tính năng thông báo đều sẵn sàng cho phiên bản cụ thể trước đó, bất kể các phương thức dùng để thiết lập chúng đều nằm trong lớp thư viện hỗ trợ NotificationCompat.Builder. Ví dụ, button action phụ thuộc vào thông báo mở rộng chỉ xuất hiện trên Android 4.1 và cấp cao hơn, bởi bản thân những thông báo mở rộng đó đều chỉ sẵn sàng trên Android 4.1 và cấp cao hơn.

Để đảm bảo tương thích tốt nhất, hãy tạo thông báo với NotificationCompat và các lớp con của nó, đặc biệt là NotificationCompat.Builder. Ngoài ra, bạn hãy tuân theo quy trình sau khi thực thi một thông báo:

1. Cung cấp tất cả các chức năng của thông báo cho mọi người dùng, bất kể họ đang dùng phiên bản nào. Để làm điều này, hãy đảm bảo rằng tất cả các chức năng đều

có sẵn từ một Activity trong ứng dụng của bạn. Có thể bạn sẽ muốn thêm một Activity mới để làm việc này.

Ví dụ, nếu muốn sử dụng addAction() nhằm cung cấp một điều khiển để dừng và bắt đầu phát đa phương tiện (media playback), trước hết, hãy thực thi điều khiển này trong một Activity thuộc ứng dụng của bạn.

2. Đảm bảo mọi người dùng đều có thể sử dụng tính năng trong Activity, bằng cách khởi động nó khi người dùng nhấn vào thông báo. Để làm điều này, hãy tạo một PendingIntent cho Activity. Gọi setContentIntent() để thêm PendingIntent vào thông báo.
3. Lúc này, hãy thêm các tính năng của thông báo mở rộng bạn muốn dùng vào thông báo. Cần nhớ rằng, bất cứ chức năng nào bạn thêm cũng sẵn sàng trong Activity khởi động khi người dùng nhấn vào thông báo.

5.5.2 Quản lý thông báo

Khi cần đưa ra một thông báo nhiều lần cho cùng một loại sự kiện, bạn nên tránh việc tạo mới hoàn toàn thông báo. Thay vì vậy, bạn cần xem xét việc cập nhật thông báo trước đó hay thay đổi một số giá trị của nó, hoặc thêm vào giá trị mới, hay cả hai cách đều được.

Ví dụ, Gmail thông báo cho người dùng rằng các e-mail mới đã đến nơi bằng cách tăng số lần đếm những tin nhắn chưa đọc và bằng cách thêm một đoạn tóm lược về từng e-mail cho thông báo. Đó được gọi là kỹ thuật thông báo stacking (dạng ngăn xếp); kỹ thuật này sẽ được mô tả chi tiết hơn trong hướng dẫn "Notifications" ("Các thông báo").

> **Ghi chú:** Tính năng này của Gmail đòi hỏi kiểu view lớn dạng "hộp thư đến", đó là một phần trong tính năng thông báo mở rộng có sẵn từ phiên bản Anroid 4.1.

Mục dưới đây mô tả cách cập nhật và gỡ bỏ thông báo.

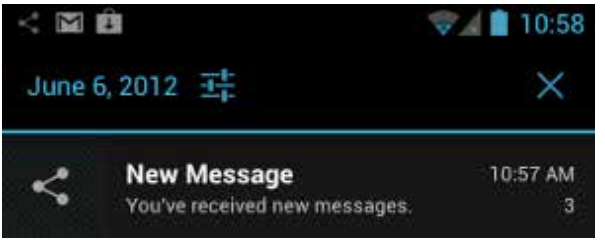
Cập nhật thông báo

Để thiết lập một thông bảo có khả năng cập nhật, hãy cấp cho nó một ID thông báo bằng cách gọi NotificationManager.notify(ID, notification). Để cập nhật thông báo khi bạn đã hiển thị nó, hãy cập nhật hoặc tạo một đối tượng NotificationCompat.Builder, xây dựng một đối tượng Notification từ đối tượng đó và hiển thị Notification với cùng ID mà bạn sử dụng trước đó. Nếu thông báo trước vẫn hiện hữu, hệ thống sẽ cập nhật nó từ nội dung của đối tượng Notification. Nếu thông báo trước đã bị tắt, một thông báo mới sẽ được tạo để thay thế.

Đoạn mã dưới đây mô tả một thông báo được cập nhật để phản ánh số lượng sự kiện đã xảy ra. Nó xếp chồng (stack) thông báo, đưa ra bản tóm lược:

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Thiết lập một ID cho thông báo để nó có thể cập nhật được
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Khởi động một vòng lặp xử lý dữ liệu rồi thông báo cho người dùng
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Do ID không đổi, thông báo đang tồn tại sẽ được cập nhật.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
...
```

Đoạn mã này tạo ra một thông báo trông như sau:



Hình 5. Thông báo được cập nhật hiển thị trong ngăn thông báo.

Gỡ bỏ thông báo

Các thông báo sẽ còn hiện hữu cho đến khi một trong những điều dưới đây xảy ra:

- Người dùng tắt từng thông báo hoặc sử dụng "Clear All" (nếu thông báo có thể bị xóa bỏ).
- Người dùng nhấn vào thông báo và bạn gọi `setAutoCancel()` khi tạo thông báo.
- Bạn gọi phương thức `cancel()` với một ID thông báo cụ thể. Phương thức này cũng xóa những thông báo đang diễn ra.
- Bạn gọi `cancelAll()` để xóa tất cả các thông báo bạn đã hiển thị trước đó.

5.5.3 Bảo toàn trải nghiệm điều hướng khi khởi động Activity

Khi khởi động Activity từ thông báo, bạn phải bảo toàn trải nghiệm điều hướng (navigation experience) của người dùng. Nhấn button Back đưa người dùng trở lại qua luồng công việc (work flow) thông thường của ứng dụng tới màn hình Home, đồng thời nhấn *Recents* để hiển thị Activity dưới dạng một tác vụ độc lập. Để bảo toàn trải nghiệm điều hướng, bạn nên khởi động Activity trong một tác vụ mới. Cách bạn thiết lập PendingIntent để cấp cho bạn một tác vụ mới phụ thuộc vào bản chất của Activity bạn đang khởi động. Có hai tình huống chính:

Activity thông thường

Bạn đang khởi động một Activity vốn là một phần trong luồng công việc thông thường của ứng dụng. Ở tình huống này, hãy thiết lập PendingIntent để khởi động một tác vụ mới, đồng thời cung cấp một ngăn xếp lùi cho PendingIntent. Ngăn xếp lùi này sẽ tái tạo hành vi Back thông thường của ứng dụng.

Các thông báo từ ứng dụng Gmail minh họa điều này. Khi nhấn vào một thông báo của một tin nhắn email, bạn sẽ thấy được chính thông điệp đó. Chạm vào button **Back** sẽ đưa bạn quay lại màn hình Home thông qua Gmail, nhưng điều này chỉ xảy ra chỉ khi bạn truy cập vào Gmail từ màn hình Home thay vì từ một thông báo.

Điều này vẫn xảy ra bất kể bạn đang ở trong ứng dụng nào khi đang chạm vào thông báo. Ví dụ, nếu bạn đang trong Gmail và soạn thảo một e-mail, đồng thời nhấn vào một thông báo của một e-mail, lập tức bạn sẽ tới e-mail đó. Chạm vào button Back sẽ đưa bạn về hộp thư đến và kế đó là màn hình Home, thay vì tới e-mail mà bạn đang soạn.

Activity đặc biệt

Người dùng chỉ thấy Activity này nếu nó được khởi động từ một thông báo. Theo một nghĩa nào đó, Activity mở rộng thông báo bằng cách cung cấp thông tin khó có thể hiển thị trong bản thân thông báo. Đối với tình huống này, hãy thiết lập PendingIntent để khởi động trong một tác vụ mới. Dù vậy, bạn không cần tạo một ngăn xếp lùi, bởi Activity đã khởi động không phải là một phần trong luồng activity của ứng dụng. Nhấn button Back sẽ đưa người dùng tới màn hình Home.

Thiết lập một activity PendingIntent thông thường

Để thiết lập một PendingIntent khởi động trực tiếp mục Activity, hãy làm theo các bước sau:

1. Định nghĩa cây phân cấp Activity của ứng dụng trong file kê khai (manifest).

   a. Thêm hỗ trợ cho Android 4.0.3 và các phiên bản trước đó. Để làm việc này, hãy xác định cha của Activity mà bạn đang khởi động bằng cách thêm một phần tử <meta-data> dưới dạng con của <activity>.

      Với phần tử này, đặt android:name="android.support.PARENT_ACTIVITY". Thiết lập android:value="<parent_

activity_name>", trong đó <parent_activity_name> là giá trị của android:name đối với phần tử<activity> cha. Xem một ví dụ ở file XML sau.

b. Đồng thời, bạn thêm hỗ trợ cho Android 4.1 và các phiên bản mới hơn. Để thực hiện điều này, hãy thêm thuộc tính android:parentActivityName vào phần tử <activity> của Activity bạn đang khởi động.

Nội dung XML cuối cùng trông sẽ như sau:

```
<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=".ResultActivity"
     android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

2. Tạo một ngăn xếp lùi dựa trên Intent khởi động Activity:

   a. Tạo Intent để khởi động Activity.

   b. Tạo một stack builder bằng cách gọi TaskStackBuilder.create().

   c. Thêm ngăn xếp lùi vào stack builder bằng cách gọi addParentStack(). Đối với từng Activity trong cây phân cấp bạn đã xác định trong file kê khai, ngăn xếp lùi chứa một đối tượng Intent khởi động Activity. Phương thức này cũng sẽ thêm các cờ khởi động ngăn xếp trong một tác vụ mới.

> **Ghi chú:** Mặc dù tham số cho addParentStack() là một tham chiếu đến Activity đã khởi động, song lời gọi phương thức này lại không thêm Intent khởi động Activity. Thay vào đó, nó được lưu ý ở bước tiếp theo.

   d. Thêm Intent khởi động Activity từ thông báo bằng cách gọi addNextIntent(). Truyền Intent bạn đã tạo ở bước đầu tiên dưới dạng tham số cho addNextIntent().

   e. Nếu cần, bạn thêm tham số cho các đối tượng Intent trong ngăn xếp bằng cách gọi TaskStackBuilder.editIntentAt(). Đôi khi, việc này rất cần thiết để đảm bảo đối tượng Activity hiển thị dữ liệu có nghĩa khi người dùng duyệt tới nó bằng cách sử dụng *Back*.

f. Lấy một PendingIntent cho ngăn xếp lùi này bằng cách gọi getPendingIntent(). Sau đó, bạn có thể sử dụng PendingIntent này dưới dạng tham số cho setContentIntent().

Đoạn mã sau minh họa quá trình này:

```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Thêm ngăn xếp lùi
stackBuilder.addParentStack(ResultActivity.class);
// Thêm Intent vào đỉnh của ngăn xếp
stackBuilder.addNextIntent(resultIntent);
// Lấy một PendingIntent chứa toàn bộ ngăn xếp lùi
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Thiết lập một activity PendingIntent đặc biệt

Phần tiếp theo mô tả cách thiết lập một activity PendingIntent đặc biệt.

Activity đặc biệt không cần ngăn xếp lùi, nên bạn không cần xác định cây phân cấp Activity của nó trong file kê khai, bạn cũng không phải gọi addParentStack() để xây dựng một ngăn xếp lùi. Thay vào đó, hãy sử dụng file kê khai để thiết lập các tùy chọn tác vụ Activity và tạo PendingIntent bằng cách gọi getActivity():

1. Trong file kê khai, thêm các thuộc tính sau vào phần tử <activity> cho Activity:

   android:name="*activityclass*"

Tên lớp đầy đủ của activity.

   android:taskAffinity=""

Kết hợp với cờ FLAG_ACTIVITY_NEW_TASK bạn thiết lập trong mã nguồn, điều này đảm bảo Activity này không đi tới tác vụ mặc định của ứng dụng. Bất cứ tác vụ đang tồn tại nào có mối liên hệ mặc định với ứng dụng đều không bị ảnh hưởng.

   android:excludeFromRecents="true"

Loại trừ tác vụ mới ra khỏi *Recents*, nên người dùng không thể vô tình duyệt quay lại nó được.

Đoạn mã nhỏ dưới đây hiển thị phần tử:

```
<activity
    android:name=".ResultActivity"
...
    android:launchMode="singleTask"
    android:taskAffinity=""
    android:excludeFromRecents="true">
</activity>
...
```

2. Xây dựng và đưa ra thông báo:
    a. Tạo một `Intent` khởi động `Activity`.
    b. Thiết lập `Activity` để khởi động trong một tác vụ mới, trống bằng cách gọi `setFlags()` với các cờ `FLAG_ACTIVITY_NEW_TASK` và `FLAG_ACTIVITY_CLEAR_TASK`.
    c. Thiết lập bất cứ tùy chọn nào khác bạn cần cho `Intent`.
    d. Tạo một `PendingIntent` từ `Intent` bằng cách gọi `getActivity()`. Sau đó, bạn có thể dùng `PendingIntent` này làm tham số cho `setContentIntent()`.

Đoạn mã sau minh họa quá trình này:

```
// Khởi tạo đối tượng Builder.
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
// Tạo một Intent cho Activity
Intent notifyIntent =
        new Intent(new ComponentName(this, ResultActivity.class));
// Thiết lập Activity để khởi động trong một tác vụ mới, trống
notifyIntent.setFlags(FLAG_ACTIVITY_NEW_TASK | FLAG_ACTIVITY_CLEAR_TASK);
// Tạo PendingIntent
PendingIntent notifyIntent =
       PendingIntent.getActivity(
       this,
       0,
       notifyIntent
       PendingIntent.FLAG_UPDATE_CURRENT
);

// Đặt PendingIntent vào bên trong trình xây dựng thông báo
builder.setContentIntent(notifyIntent);
// Các thông báo được đưa ra bằng cách gửi chúng tới
// service NotificationManager của hệ thống.
```

```
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Xây dựng một đối tượng Notification ẩn danh từ stack builder, và
// truyền nó tới NotificationManager
mNotificationManager.notify(id, builder.build());
```

5.5.4 Hiển thị tiến trình trong thông báo

Các thông báo có thể chứa một thanh mô tả trạng thái tiến trình động (animated progress indicator) cho người dùng thấy tình trạng của một hoạt động đang chạy. Nếu bạn có thể ước lượng hoạt động diễn ra trong bao lâu và lượng công việc mà nó hoàn thành vào bất cứ thời điểm nào, hãy sử dụng dạng xác định của thanh mô tả trạng thái (một dạng xác định). Nếu bạn không thể ước lượng hoạt động diễn ra trong bao lâu, hãy sử dụng dạng không xác định của thanh mô tả trạng thái (một thanh mô tả trạng thái activity).

Các thanh mô tả trạng thái tiến trình được hiển thị sử dụng một lớp thực thi của lớp `ProgressBar`.

Để sử dụng thanh mô tả trạng thái tiến trình trên các nền tảng từ Android 4.0, hãy gọi `setProgress()`. Đối với các phiên bản trước, bạn phải tự tạo layout thông báo tùy chỉnh của mình, trong đó chứa một view `ProgressBar`.

Các mục dưới đây mô tả cách hiển thị tiến trình trong một thông báo bằng cách sử dụng `setProgress()`.

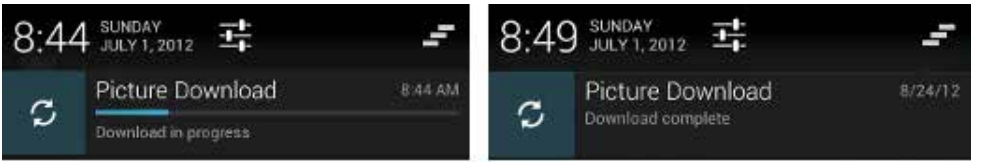
Hiển thị một thanh mô tả trạng thái tiến trình trong thời gian cố định

Để hiển thị một progress bar xác định, hãy thêm thanh đó vào thông báo của bạn bằng cách gọi `setProgress(max, progress, false)`, sau đó hiển thị thông báo. Khi hoạt động diễn ra, `progress` tăng dần và cập nhật thông báo. Vào thời điểm cuối hoạt động, `progress` nên bằng `max`. Cách thông thường để gọi `setProgress()` là đặt `max` có giá trị 100, sau đó tăng `progress` dưới dạng giá trị "phần trăm hoàn thành" ("percent complete" value) cho hoạt động.

Bạn có thể rời khỏi progress bar khi hoạt động đã hoàn thành, hoặc hủy nó. Trong cả hai trường hợp, hãy nhớ cập nhật văn bản của thông báo để cho thấy rằng hoạt động đã hoàn thành. Để hủy progress bar, gọi `setProgress(0, 0, false)`. Ví dụ:

```
...
mNotifyManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Khởi động một hoạt động dài trong luồng chạy nền (background thread)
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Chạy hoạt động "dài dòng" này 20 lần
            for (incr = 0; incr <= 100; incr+=5) {
                // Thiết lập thanh mô tả trạng thái tiến trình tới giá trị cao
                // nhất, số phần trăm hoàn thành hiện tại và trạng thái xác định
                mBuilder.setProgress(100, incr, false);
                // Hiển thị progress bar lần đầu tiên.
                mNotifyManager.notify(0, mBuilder.build());
                    // Cho luồng nghỉ và thúc đẩy hoạt động chiếm thời gian
                    try {
                        // Nghỉ trong 5 giây
                        Thread.sleep(5*1000);
                    } catch (InterruptedException e) {
                        Log.d(TAG, "sleep failure");
                    }
            }
            // Khi vòng lặp được hoàn thành, cập nhật thông báo
            mBuilder.setContentText("Download complete")
            // Hủy progress bar
                .setProgress(0,0,false);
            mNotifyManager.notify(ID, mBuilder.build());
        }
    }
// Khởi động luồng bằng cách gọi phương thức run() trong Runnable của nó
).start ();
```

Các thông báo kết quả được trình bày ở Hình 6. Bên trái là ảnh chụp nhanh của thông báo trong khi hoạt động đang diễn ra; bên phải là ảnh chụp nhanh của thông báo sau khi hoạt động đã hoàn thành.



Hình 6. Progress bar trong và sau khi diễn ra hoạt động.

Hiển thị một thanh mô tả trạng thái activity liên tục

Để hiển thị một thanh mô tả trạng thái activity không xác định, hãy thêm trình này vào thông báo bằng cách sử dụng `setProgress(0, 0, true)` (hai tham số đầu tiên bị bỏ qua), sau đó hiển thị thông báo. Kết quả nhận được sẽ là một thanh mô tả trạng thái cùng kiểu với progress bar, ngoại trừ hoạt cảnh (animation) của nó đang diễn ra.

Đưa ra thông báo vào lúc bắt đầu hoạt động. Hoạt cảnh sẽ chạy cho đến khi bạn chỉnh sửa thông báo của mình. Khi hoạt động hoàn thành, gọi `setProgress()` `setProgress(0, 0, false)`, sau đó cập nhật thông báo để hủy thanh mô tả trạng thái activity. Hãy luôn làm điều này; bằng không, hoạt cảnh sẽ chạy ngay cả khi hoạt động đã hoàn thành. Đồng thời, hãy nhớ thay đổi văn bản của thông báo để chỉ ra rằng hoạt động đã hoàn tất.

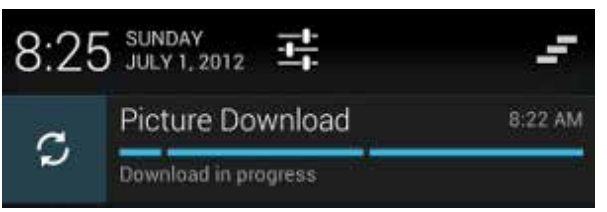
Để xem cách thức các thanh mô tả trạng thái activity hoạt động, hãy tham chiếu đến đoạn mã nhỏ trước đó. Tìm đến các dòng mã sau:

```
// Thiết lập cho thanh mô tả trạng thái tiến trình giá trị cao nhất,
// số phần trăm hoàn thành hiện tại và trạng thái "xác định"
mBuilder.setProgress(100, incr, false);
// Hiển thị thông báo
mNotifyManager.notify(0, mBuilder.build());
```

Thay thế những dòng bạn vừa tìm thấy bằng các dòng sau:

```
// Thiết lập một thanh mô tả trạng thái activity cho hoạt động có độ
// dài không xác định
mBuilder.setProgress(0, 0, true);
// Đưa ra thông báo
mNotifyManager.notify(0, mBuilder.build());
```

Thanh mô tả trạng thái kết quả được trình bày ở Hình 7:



Hình 7. Một Thanh mô tả trạng thái activity đang chạy.

Các layout thông báo tùy chỉnh

Framework thông báo cho phép bạn định nghĩa một layout thông báo tùy chỉnh (custom notification layout), layout này định nghĩa hình thức hiển thị của thông báo trong một đối tượng `RemoteViews`. Thông báo có layout tùy chỉnh tương tự các thông báo thường, ngoại trừ chúng được dựa trên một `RemoteViews` định nghĩa trong file layout XML.

Chiều cao khả dụng của một layout thông báo tùy chỉnh phụ thuộc vào view thông báo. Layout view thông thường bị giới hạn ở 64 dp, còn layout view mở rộng bị giới hạn ở 256 dp.

Để định nghĩa một layout thông báo tùy chỉnh, hãy bắt đầu bằng cách khởi tạo một đối tượng `RemoteViews` có thể điền (inflate) từ file layout XML. Sau đó, thay vì gọi các phương thức như `setContentTitle()`, hãy gọi `setContent()`. Để thiết lập nội dung chi tiết trong thông báo tùy chỉnh, hãy sử dụng các phương thức trong `RemoteViews` để thiết lập giá trị cho con của view:

1. Tạo một layout XML cho thông báo trong một file riêng biệt. Bạn có thể sử dụng bất cứ tên file nào mình muốn, nhưng phải sử dụng đuôi mở rộng `.xml`.
2. Trong ứng dụng của mình, bạn hãy sử dụng các phương thức `RemoteViews` để xác định biểu tượng và văn bản của thông báo. Đặt đối tượng `RemoteViews` vào `NotificationCompat.Builder` của bạn bằng cách gọi `setContent()`. Đừng thiết lập một `Drawable` nền trên đối tượng `RemoteViews` của bạn, bởi vì điều này có thể khiến màu văn bản trở nên khó đọc.

Lớp `RemoteViews` cũng chứa các phương thức mà bạn có thể dễ dàng sử dụng để thêm một `Chronometer` hoặc `ProgressBar` vào layout của thông báo. Để biết thêm thông tin về cách tạo layout tùy chỉnh cho thông báo, mời bạn tham khảo tài liệu `RemoteViews`.

> **Chú ý:** Khi sử dụng layout thông báo tùy chỉnh, bạn phải thực sự cẩn trọng để đảm bảo layout tùy chỉnh của mình có thể làm việc với các chiều xoay cũng như độ phân giải khác nhau của thiết bị. Lời khuyên này áp dụng cho mọi layout View, nó cũng đặc biệt quan trọng đối với các thông báo, bởi không gian trong ngăn thông báo rất hạn chế. Đừng làm cho layout tùy chỉnh của mình phức tạp quá mức, đồng thời chắc chắn rằng bạn kiểm thử nó với nhiều cấu hình khác nhau.

Sử dụng tài nguyên style cho văn bản thông báo tùy chỉnh

Hãy luôn sử dụng tài nguyên style cho các phần văn bản trong thông báo tùy chỉnh. Màu nền của thông báo có thể thay đổi qua những thiết bị và phiên bản khác nhau, song việc sử dụng tài nguyên style giúp bạn giải quyết bài toán này. Bắt đầu từ Android 2.3, hệ thống xác định một style cho văn bản layout thông báo chuẩn. Nếu sử dụng cùng style trong các ứng dụng cho Android 2.3 và cấp cao hơn, bạn sẽ đảm bảo rằng văn bản của mình nổi bật trên nền hiển thị.

5.6 Thành phần tùy chỉnh

Android đưa ra một mô hình phức tạp và cấu trúc mạnh mẽ cho việc xây dựng giao diện người dùng, dựa trên các lớp layout cơ bản là `View` và `ViewGroup`. Để bắt đầu, nền tảng bao gồm một loạt View và lớp con ViewGroup dựng sẵn - được gọi tương ứng là các widget và layout - mà bạn có thể sử dụng khi xây dựng giao diện người dùng cho mình.

Một phần danh sách widget có sẵn bao gồm `Button`, `TextView`, `EditText`, `ListView`, `CheckBox`, `RadioButton`, `Gallery`, `Spinner`, cùng với đó là nhiều widget cho những mục đích đặc biệt, gồm `AutoCompleteTextView`, `ImageSwitcher` và `TextSwitcher`.

Có mặt trong số các layout có sẵn là `LinearLayout`, `FrameLayout`, `RelativeLayout` và những đối tượng khác. Để tham khảo thêm ví dụ, xem mục "Common Layout Objects" ("Các đối tượng layout thông thường").

Nếu không có widget và layout có sẵn nào phù hợp với nhu cầu của bạn, hãy tự tạo lớp con View. Chỉ cần điều chỉnh một chút những layout hoặc widget có sẵn là bạn đã có thể tạo lớp con cho chúng và ghi đè các phương thức của chúng.

Việc tự tạo các lớp con View giúp bạn điều khiển chính xác giao diện và các chức năng của một phần tử màn hình. Để đưa ra ý tưởng về điều khiển bạn có với các view tùy chỉnh, dưới đây cung cấp vài ví dụ để bạn có thể thực hiện vài việc với chúng:

- Bạn có thể tạo một kiểu View tự thiết kế hoàn toàn, ví dụ như một nút bấm "volume control" ("điều khiển âm lượng") có được từ việc sử dụng đồ họa 2D.
- Bạn có thể kết hợp một nhóm thành phần View vào một thành phần đơn mới để tạo một vài thứ giống như ComboBox (kết hợp danh sách popup và trường văn bản nhập tự do), một điều khiển bộ chọn hai chiều (dual-pane selector control) - đây chính là một khung gồm hai phần trái và phải, mỗi phần chứa một danh sách mà tại đó bạn có thể gán lại (di chuyển) phần tử giữa hai danh sách,...
- Bạn có thể ghi đè lên cách mà thành phần EditText được dựng trên màn hình. (Bài hướng dẫn về Notepad - Notepad Tutorial sử dụng cách làm này để tạo hiệu ứng đẹp và tạo một trang ghi chú dạng dòng).
- Bạn có thể bắt những sự kiện khác như ấn phím và xử lý chúng theo cách riêng (chẳng hạn như cho một trò chơi).

Các mục dưới đây trình bày cách thức tạo View tùy chỉnh và dùng chúng trong ứng dụng. Để có thêm thông tin chi tiết, xem lớp `View`.

Phương pháp cơ bản

Dưới đây là kiến thức tổng quan ở mức nâng cao về những điều bạn cần biết để bắt đầu tạo thành phần View của riêng mình:

1. Mở rộng một lớp `View` hiện có hoặc lớp con chứa lớp của bạn.
2. Ghi đè một số phương thức từ lớp cha. Các phương thức của lớp cha bị ghi đè sẽ bắt đầu với '`on`', ví dụ như `onDraw()`, `onMeasure()` và `onKeyDown()`. Thao

tác tương tự với các sự kiện `on...` trong `Activity` hoặc `ListActivity` mà bạn ghi đè cho toàn vòng đời và các hàm khác.

3. Sử dụng lớp mở rộng mới của bạn. Một khi đã hoàn tất, bạn có thể sử dụng lớp mở rộng mới này.

> **Mách nhỏ:** Lớp mở rộng có thể được định nghĩa dưới dạng lớp nội bộ (inner class) trong activity sử dụng chúng. Điều này thực sự hữu ích, bởi nó kiểm soát các truy cập không cần thiết tới lớp mở rộng (có thể bạn muốn tạo một View ở dạng public cho những mục đích sử dụng rộng hơn trong ứng dụng của mình).

Thành phần tùy chỉnh hoàn toàn

Thành phần tùy chỉnh hoàn toàn có thể được dùng để tạo ra thành phần đồ họa hiển thị theo cách bạn muốn. Có lẽ, một bộ đo VU (VU meter) giống như đồng hồ đo xăng hoặc một danh sách bài hát theo chiều dọc, tại đây một quả bóng di chuyển dọc theo các từ để bạn có thể hát cùng với máy karaoke. Bất kể theo cách nào, cũng có thứ gì đó mà những thành phần có sẵn không làm được, cho dù bạn đã cố gắng kết hợp chúng.

Thật may là bạn có thể dễ dàng tạo ra các thành phần có giao diện cũng như hành vi mình muốn, có lẽ điều này chỉ bị hạn chế bởi khả năng tưởng tượng của bạn, kích thước màn hình và hiệu năng xử lý hiện tại (cần nhớ rằng, điều hiển nhiên là ứng dụng của bạn phải chạy những ứng dụng với hiệu năng kém hơn máy tính để bàn (workstation)).

Để tạo một thành phần tùy chỉnh hoàn toàn:

1. Thành phần phổ biến nhất mà bạn có thể mở rộng là `View`, nên thường thì bạn sẽ bắt đầu bằng cách mở rộng nó để tạo thành phần cấp cao mới cho mình.
2. Bạn có thể cung cấp một phương thức khởi tạo có khả năng lấy các thuộc tính và tham số từ XML. Ngoài ra, bạn cũng có thể sử dụng các thuộc tính và tham số của mình (có thể là màu sắc và độ dài của bộ đo VU, hoặc chiều rộng và độ giảm của kim,…).
3. Có thể bạn sẽ muốn tạo các trình lắng nghe sự kiện , phương thức truy cập và chỉnh sửa thuộc tính riêng, và cũng có thể là những hành vi phức tạp hơn trong lớp thành phần.
4. Chắc hẳn bạn sẽ muốn ghi đè lên `onMeasure()` và `onDraw()` nếu muốn thành phần đó hiển thị điều gì đó. Trong khi cả hai phương thức trên đều có hành vi mặc định, `onDraw()` mặc định sẽ không làm gì, còn mặc định `onMeasure()` luôn thiết lập kích thước 100x100 - có thể đây không phải là điều bạn muốn.
5. Các phương thức `on...` khác cũng có thể bị ghi đè theo yêu cầu.

Mở rộng `onDraw()` và `onMeasure()`

Phương thức `onDraw()` mang đến cho bạn một khung vẽ canvas (canvas); trên khung này, bạn có thể thực thi bất cứ thứ gì mình muốn: Đồ họa 2D, các thành phần chuẩn hoặc tùy chỉnh khác, văn bản được định kiểu trình bày (styled text), hoặc bất kỳ thứ gì khác bạn có thể nghĩ đến.

> **Ghi chú:** Điều này không áp dụng cho đồ họa 3D. Nếu muốn sử dụng đồ họa 3D, bạn phải mở rộng `SurfaceView` thay vì View, đồng thời vẽ từ một luồng (thread) riêng biệt khác. Xem ví dụ về GLSurfaceViewActivity để có thêm thông tin chi tiết.

`onMeasure()` là phương thức liên quan nhiều hơn một chút. `onMeasure()` là phần quan trọng trong mối liên hệ giữa thành phần của bạn và đối tượng chứa của nó. `onMeasure()` được ghi đè để báo cáo một cách hiệu quả và chính xác kết quả đo của những thành phần mà nó chứa. Việc này được thực hiện phức tạp hơn một chút bởi các yêu cầu về giới hạn từ cha (được truyền vào qua phương thức `onMeasure()`) cũng như bởi yêu cầu gọi phương thức `setMeasuredDimension()` với chiều dài và chiều cao đo lường khi chúng đã được tính toán. Nếu bạn không gọi được phương thức này từ một phương thức `onMeasure()` ghi đè, kết quả sẽ là một ngoại lệ tại thời điểm đó.

Ở mức cao, thực thi `onMeasure()` sẽ như sau:

6. Phương thức `onMeasure()` bị ghi đè sẽ được gọi với những thông số đo lường về chiều rộng và chiều cao (các tham số `widthMeasureSpec` và `heightMeasureSpec`, cả hai đều là mã dạng số nguyên biểu diễn độ dài). Các thông số đo lường trên nên được coi là yêu cầu về giới hạn chiều rộng và chiều dài đo lường mà bạn nên đưa ra. Bạn có thể tìm thấy nguồn tham khảo đầy đủ về các loại hạn chế mà những thông số này có thể đòi hỏi trong tài liệu tham chiếu dưới `View.onMeasure(int, int)` (tài liệu này đã làm rất tốt việc giải thích toàn bộ hoạt động đo lường).
7. Phương thức `onMeasure()` trong thành phần của bạn nên tính toán một chiều rộng và chiều cao đo lường sẽ được yêu cầu để dựng thành phần đó. Phương thức này nên thử làm việc với các thông số được truyền vào, mặc dù nó có thể lựa chọn bỏ qua chúng (trong trường hợp này, đối tượng cha có thể chọn việc để làm, bao gồm cắt (clipping), cuộn (scrolling), ném ra (throwing) một ngoại lệ (exception), hoặc yêu cầu `onMeasure()` thử lại lần nữa với những thông số đo lường khác).
8. Khi đã tính xong chiều rộng và chiều cao, phương thức `setMeasuredDimension(int width, int height)` phải được gọi với các thông số đo lường đã tính toán. Nếu thất bại, kết quả trả về một ngoại lệ được ném ra.

Sau đây là tổng quan về một số phương thức chuẩn khác mà framework có thể gọi trên các view:

Hạng mục	Phương thức	Mô tả
Tạo	Phương thức khởi tạo	Gồm có một dạng phương thức khởi tạo được gọi khi view được tạo ra từ mã nguồn và một dạng được gọi khi view được tạo từ một file layout. Dạng thứ hai nên phân tích và áp dụng bất cứ thuộc tính nào xác định trong file layout.
	`onFinishInflate()`	Được gọi sau khi một view và tất cả con của nó được tạo từ XML.
Layout (bố cục)	`onMeasure(int, int)`	Được gọi để xác định yêu cầu về kích thước cho view này cùng tất cả con của nó.
	`onLayout(boolean, int, int, int, int)`	Được gọi khi view này cần gán một kích thước và vị trí cho tất cả con của nó.
	`onSizeChanged(int, int, int, int)`	Được gọi khi kích thước của view này thay đổi.
Dựng hình (drawing)	`onDraw(Canvas)`	Được gọi khi view sắp dựng nội dung của nó.

Hạng mục	Phương thức	Mô tả
Xử lý sự kiện	`onKeyDown(int, KeyEvent)`	Được gọi khi một sự kiện bàn phím mới xảy ra.
	`onKeyUp(int, KeyEvent)`	Được gọi khi một sự kiện thả phím xảy ra.
	`onTrackballEvent(MotionEvent)`	Được gọi khi một sự kiện chuyển động bi lăn xảy ra.
	`onTouchEvent(MotionEvent)`	Được gọi khi một sự kiện chuyển động trên màn hình cảm ứng xảy ra.
Focus	`onFocusChanged(boolean, int, Rect)`	Được gọi khi view được focus hoặc mất focus.
	`onWindowFocusChanged(boolean)`	Được gọi khi cửa sổ chứa view được focus hoặc mất focus.
Gắn (attach)	`onAttachedToWindow()`	Được gọi khi view được gắn vào một cửa sổ.
	`onDetachedFromWindow()`	Được gọi khi view được gỡ khỏi cửa sổ gắn với nó.
	`onWindowVisibilityChanged(int)`	Được gọi khi trạng thái hiển thị (visibility) của cửa sổ chứa view thay đổi.

Ví dụ về View tùy chỉnh

Ví dụ mẫu CustomView trong tài liệu "API Demos" ("Các bản demo API") cung cấp ví dụ về một View đã được tùy chỉnh. View tùy chỉnh được định nghĩa trong lớp LabelView.

LabelView minh họa một số khía cạnh khác nhau của các thành phần tùy chỉnh:

- Mở rộng lớp View cho một thành phần tùy chỉnh hoàn toàn.
- Tham số hóa (parameterize) phương thức khởi tạo nào nhận các tham số được định nghĩa trong file XML của view. Một vài trong số chúng được truyền qua lớp View cha, nhưng quan trọng hơn, có một số thuộc tính tùy chỉnh được định nghĩa và sử dụng cho LabelView.

- Các phương thức public chuẩn của kiểu mà bạn muốn thấy cho một thành phần nhãn (label component), ví dụ như `setText()`, `setTextSize()`, `setTextColor()`...
- Phương thức `onMeasure` ghi đè xác định và thiết lập kích thước khi dựng của thành phần. (Lưu ý, trong LabelView, công việc thực sự được hoàn thành bởi một phương thức private `measureWidth()`).
- Phương thức `onDraw()` ghi đè vẽ nên nhãn trên khung canvas được cung cấp.

Bạn có thể thấy một số mẫu sử dụng của View tùy chỉnh LabelView trong custom_view_1.xml từ phần những ví dụ mẫu. Cụ thể, bạn có thể thấy sự kết hợp của cả hai bộ tham số namespace `android:` và bộ tham số namespace tùy chỉnh `app:`. Các tham số `app:` này là những tham số tùy chỉnh mà LabelView nhận diện và làm việc cùng, được định nghĩa trong một lớp nội bộ bên trong các lớp định nghĩa tài nguyên R mẫu.

Các điều khiển hỗn hợp

Nếu không muốn tạo một thành phần tùy chỉnh hoàn toàn, bạn hãy tìm cách đặt một thành phần có thể tái sử dụng chứa nhóm điều khiển đã tồn tại, sau đó tạo một thành phần hỗn hợp (Compound Component), hay còn gọi là điều khiển hỗn hợp (Compound Control) để thỏa mãn yêu cầu bạn đưa ra. Nói ngắn gọn, điều này nhóm một loạt điều khiển (hoặc view) đơn nguyên hơn thành một nhóm mục có liên quan lôgíc với nhau và nhóm này có thể được coi là một phần tử đơn lẻ. Ví dụ, một Combo Box có thể được coi như một sự kết hợp của trường EditText một dòng và một button điều chỉnh với một PopupList gắn vào. Nếu bạn nhấn button và chọn thứ gì đó từ danh sách, nó sẽ điền vào trường EditText. Bên cạnh đó, nếu muốn, người dùng cũng có thể gõ trực tiếp cái gì đó vào bên trong trường EditText.

Trong Android, thực ra đã có hai View khác sẵn sàng làm việc này: `Spinner` và `AutoCompleteTextView`, nhưng dù sao thì khái niệm về Combo Box cũng là một ví dụ dễ hiểu.

Để tạo một thành phần hỗn hợp:

1. Xuất phát điểm thường là một loại layout nào đó; vì vậy, hãy tạo một lớp mở rộng layout. Trong ví dụ này về Combo box, chúng ta sẽ sử dụng LinearLayout (layout tuyến tính) với hướng thiết bị xoay theo chiều ngang. Cần nhớ rằng, các layout khác có thể được lồng bên trong, nên thành phần hỗn hợp có thể có cấu trúc và mức độ phức tạp tùy ý. Lưu ý, tương tự một Activity, bạn có thể sử dụng phương pháp khai báo (dựa trên XML) để tạo các thành phần chứa trong đó, hoặc bạn cũng có thể lồng chúng bằng cách lập trình từ mã nguồn.
2. Trong phương thức khởi tạo của lớp mới, hãy cung cấp bất cứ tham số nào lớp cha muốn, rồi truyền chúng qua phương thức khởi tạo của lớp cha trước tiên. Sau đó, bạn có thể thiết lập các view khác để sử dụng trong thành phần mới của mình; đây là nơi bạn sẽ tạo trường EditText và PopupList. Lưu ý, bạn cũng có thể đưa các thuộc tính và tham số của mình vào trong XML, và XML có thể được phương thức khởi tạo của bạn lấy ra và sử dụng.
3. Bạn cũng có thể tạo những trình lắng nghe sự kiện mà các view chứa trong đó có thể tạo ra, ví dụ như một phương thức lắng nghe cho trình lắng nghe sự kiện click

vào một mục trong danh sách (List Item Click Listener) để cập nhật nội dung của EditText nếu người dùng chọn một mục trong danh sách đó.

4. Bạn cũng có thể tự tạo thuộc tính cùng với các phương thức truy cập và chỉnh sửa chúng, ví dụ như cho phép giá trị EditText được khởi tạo trong thành phần, cũng như truy vấn nội dung của nó khi cần.
5. Trong trường hợp mở rộng một layout, bạn không cần ghi đè lên các phương thức `onDraw()` và `onMeasure()`, vì layout sẽ có hành vi mặc định có khả năng làm tốt việc này. Tuy nhiên, nếu cần, bạn vẫn có thể ghi đè chúng.
6. Bạn có thể ghi đè những phương thức `on...` khác, ví dụ như `onKeyDown()`, để chọn những giá trị mặc định từ danh sách popup của một combo box khi một phím nào đó được nhấn.

Tóm lại, việc sử dụng một layout làm cơ sở cho một điều khiển tùy chỉnh đem lại rất nhiều lợi ích, bao gồm:

- Bạn có thể xác định layout bằng cách sử dụng các file XML khai báo như với màn hình activity, hoặc bạn có thể tạo các view một cách tự động và lồng chúng vào trong layout từ mã nguồn.
- Các phương thức `onDraw()` và `onMeasure()` (cùng với hầu hết những phương thức on... khác) sẽ có hành vi phù hợp, nên bạn không cần phải ghi đè chúng.
- Cuối cùng, bạn có thể xây dựng các view hỗn hợp phức tạp một cách tùy ý, nhanh chóng và tái sử dụng chúng nếu chúng đều là thành phần đơn.

Các ví dụ về điều khiển hỗn hợp

Trong dự án API Demos đi kèm SDK, có hai ví dụ về danh sách - Ví dụ 4 (Example 4) và Ví dụ 6 (Example 6) bên dưới Views/Lists biểu diễn một SpeechView mở rộng LinearLayout để tạo một thành phần hiển thị các trích dẫn phát biểu. Các lớp tương ứng trong mã mẫu là `List4.java` và `List6.java`.

Chỉnh sửa một kiểu View có sẵn

Có một tùy chọn giúp bạn tạo một View tùy chỉnh dễ dàng hơn, tùy chọn này cũng rất hữu ích trong một vài trường hợp nhất định. Nếu đã tồn tại sẵn một thành phần tương tự như ý muốn, bạn có thể mở rộng thành phần và chỉ việc ghi đè hành vi mà mình muốn thay đổi. Bạn có thể làm mọi thứ mình muốn với một thành phần tùy chỉnh hoàn toàn; tuy nhiên, bằng cách bắt đầu với một lớp chuyên biệt hơn trong cây phân cấp View, bạn có thể thu được nhiều hành vi thực hiện chính xác điều bạn muốn làm.

Ví dụ, SDK chứa một Ứng dụng NotePad trong các ví dụ mẫu. Điều này lý giải cho nhiều khía cạnh của việc sử dụng nền tảng Android, trong số đó là việc mở rộng View EditText để tạo ghi chú dạng dòng (lined notepad). Đây không phải một ví dụ hoàn hảo, và các hàm API thực hiện việc này có thể thay đổi từ bản xem (preview) trước đó, song nó vẫn đưa ra những lý giải thích hợp cho các nguyên lý (principle).

Nếu bạn chưa hoàn thành nó, hãy nhập mã nguồn mẫu của NotePad vào Eclipse (hoặc xem mã nguồn trực tiếp trên liên kết được cung cấp). Cụ thể là, hãy xem định nghĩa của `MyEditText` trong file NoteEditor.java tại địa chỉ: http://developer.android.com/samples/index.html.

Ở đây, có một số điểm cần lưu ý.

1. **Định nghĩa**

   Lớp được định nghĩa với dòng sau:
   ```
   public static class MyEditText extends EditText
   ```

   - Nó đóng vai trò là lớp nội bộ bên trong activity `NoteEditor`, song vì là public nên nó có thể được truy cập dưới dạng `NoteEditor.MyEditText` từ bên ngoài lớp `NoteEditor` nếu muốn.
   - Nó là `static`, nghĩa là không tạo các "phương thức tổng hợp" ("synthetic method") mà cho phép truy cập dữ liệu từ lớp cha, có nghĩa rằng nó có hành vi như một lớp riêng biệt hơn là một cái gì đó liên quan chặt chẽ với `NoteEditor`. Đây là cách rõ ràng hơn giúp tạo các lớp nội bộ nếu chúng không cần truy cập tới trạng thái từ lớp bên ngoài, giữ cho lớp tạo ra được nhỏ gọn, đồng thời cho phép nó có thể được những lớp khác sử dụng dễ dàng.
   - Nó mở rộng `EditText`, đó là View chúng ta đã chọn để tùy chỉnh trong trường hợp này. Khi chúng ta hoàn tất, lớp mới có thể thay thế cho một view `EditText` thông thường.

2. **Khởi tạo lớp**

   Như mọi khi, lớp cha được gọi trước. Hơn nữa, đây không phải phương thức khởi tạo mặc định mà là một phương thức khởi tạo được tham số hóa. EditText được tạo với những tham số đó khi nó được tạo từ một file layout XML; do đó, phương thức khởi tạo vừa phải lấy chúng về vừa phải truyền cho phương thức khởi tạo của lớp cha.

3. **Các phương thức ghi đè**

   Trong ví dụ này, chỉ có một phương thức ghi đè: `onDraw()` - tuy nhiên, có thể có nhiều phương thức khác khi bạn tạo thành phần tùy chỉnh của mình.

   Đối với ví dụ NotePad, việc ghi đè lên phương thức `onDraw()` cho phép chúng ta vẽ những dòng màu xanh lên khung canvas của view `EditText` (khung canvas được truyền tới phương thức `onDraw()` ghi đè). Phương thức cha `.onDraw()` được gọi trước khi phương thức con kết thúc. Phương thức của lớp cha nên được kích hoạt, nhưng trong trường hợp này, chúng ta làm điều đó ở bước cuối cùng sau khi đã vẽ các dòng mình muốn bao gồm.

4. **Sử dụng thành phần tùy chỉnh**

   Bây giờ, chúng ta đã có thành phần tùy chỉnh của mình, nhưng làm sao để có thể sử dụng nó? Trong ví dụ NotePad, thành phần tùy chỉnh được sử dụng trực tiếp từ layout khai báo; vậy nên, hãy nhìn vào `note_editor.xml` trong thư mục `res/layout`.

```
<view
    class="com.android.notepad.NoteEditor$MyEditText"
    id="@+id/note"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:drawable/empty"
    android:padding="10dip"
    android:scrollbars="vertical"
    android:fadingEdge="vertical" />
```

- Thành phần tùy chỉnh được tạo dưới dạng một view chung trong XML, còn lớp được xác định bởi gói đầy đủ. Lưu ý, lớp nội bộ chúng ta vừa định nghĩa được tham chiếu bằng cách sử dụng ghi chú (notation) `NoteEditor$MyEditText`, đây là cách chuẩn để tham chiếu tới các lớp nội bộ trong ngôn ngữ lập trình Java.

  Nếu thành phần View tùy chỉnh không được định nghĩa dưới dạng một lớp nội bộ thì cách thay thế là bạn có thể khai báo thuộc tính View với tên phần tử XML và loại bỏ thuộc tính class. Ví dụ:

  ```
  <com.android.notepad.MyEditText
      id="@+id/note"
      ... />
  ```

  Lưu ý, lớp MyEditText lúc này nằm trong một file riêng biệt. Khi lớp đó được lồng trong lớp NoteEditor, kỹ thuật này sẽ không hoạt động.

- Các thuộc tính và tham số khác trong định nghĩa được truyền vào phương thức khởi tạo thành phần tùy chỉnh, sau đó truyền qua phương thức khởi tạo EditText, nên chúng là những tham số giống nhau mà bạn có thể sử dụng cho một view EditText. Lưu ý, bạn cũng có thể tự thêm các tham số của mình, chúng ta sẽ bàn lại việc này sau.

Và đó là tất cả những gì bạn cần làm. Phải thừa nhận rằng đây là một trường hợp đơn giản, song lại là chìa khóa để tạo ra các thành phần tùy chỉnh chỉ vừa đủ phức tạp ở mức bạn cần.

Một thành phần phức tạp hơn có thể ghi đè lên các phương thức `on...` nhiều hơn, đồng thời đưa ra một số phương thức trợ giúp của chúng, về căn bản là để tùy chỉnh các thuộc tính và hành vi. Giới hạn duy nhất là trí tưởng tượng của bạn và những gì bạn muốn thành phần đó thực hiện.

6. Xử lý đầu vào từ bàn phím

Hệ thống Android đưa ra một bàn phím trên màn hình (on-screen keyboard) - được biết đến như một phương tiện nhập liệu mềm (soft input method) - khi một trường văn bản trong giao diện người dùng được focus. Để cung cấp trải nghiệm tốt nhất cho người dùng, bạn có thể xác định các đặc điểm về loại đầu vào mình muốn (chẳng hạn như nó là số điện thoại hay địa chỉ e-mail) và hành vi của phương tiện nhập liệu (chẳng hạn như nó có thực hiện hành vi tự sửa lỗi về ngữ pháp hay không).

Ngoài các phương tiện nhập liệu trên màn hình, Android còn hỗ trợ bàn phím cứng, nên điều quan trọng là ứng dụng của bạn phải tối ưu trải nghiệm người dùng để tương tác có thể xảy ra, cho dù đó là bàn phím gắn kèm.

Những chủ đề này được bàn luận ở các mục dưới đây.

6.1 Xác định kiểu phương tiện nhập liệu

Mỗi trường văn bản đều phục vụ cho một loại văn bản đầu vào tương ứng, chẳng hạn như địa chỉ e-mail, số điện thoại, hay chỉ đơn thuần là văn bản dạng chữ. Vì vậy, điều quan trọng là bạn phải xác định kiểu đầu vào cho từng trường văn bản trong ứng dụng của mình, nhằm giúp hệ thống hiển thị phương tiện nhập liệu mềm phù hợp (chẳng hạn như một bàn phím trên màn hình).

Bên cạnh những loại button có sẵn kèm theo phương tiện nhập liệu, bạn nên xác định các hành vi, chẳng hạn như phương tiện này có cung cấp gợi ý chính tả (spelling suggestion) không, hoặc viết hoa đầu dòng các câu mới và thay thế button ngắt dòng bằng một button action, chẳng hạn như Done hoặc Next. Mục này sẽ chỉ cho bạn thấy cách xác định những đặc điểm trên.

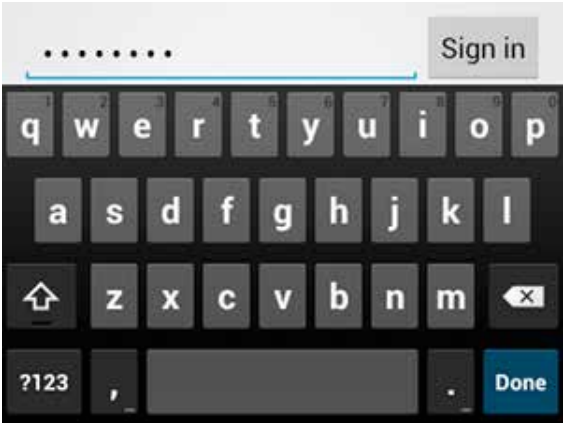
Xác định kiểu bàn phím

Bạn hãy luôn khai báo phương tiện nhập liệu cho trường văn bản của mình bằng cách thêm thuộc tính `android:inputType` vào phần tử `<EditText>`.

Hình 1. Kiểu đầu vào `phone`.

Ví dụ, nếu bạn muốn có một phương tiện nhập liệu cho việc nhập số điện thoại, hãy sử dụng giá trị "`phone`":

```
<EditText
    android:id="@+id/phone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/phone_hint"
    android:inputType="phone" />
```



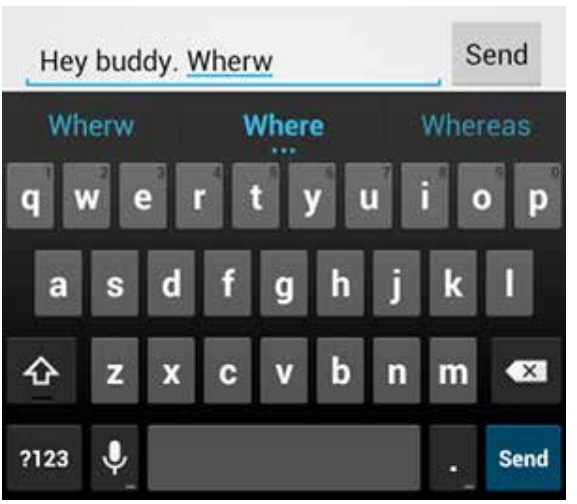
Hình 2. Kiểu đầu vào `textPassword`.

Hoặc, nếu trường văn bản là dành cho mật khẩu, hãy sử dụng giá trị "`textPassword`" để trường đó che (conceal) dữ liệu đầu vào khi người dùng nhập:

```
<EditText
    android:id="@+id/password"
    android:hint="@string/password_hint"
    android:inputType="textPassword"
    ... />
```

Một số giá trị có thể được ghi lại với thuộc tính `android:inputType` và một số giá trị có thể được kết hợp để định nghĩa giao diện cho phương tiện nhập liệu cùng những hành vi thêm vào.

Bật gợi ý chính tả và các hành vi khác



Hình 3. Thêm `textAutoCorrect` sẽ cung cấp khả năng tự sửa các lỗi chính tả.

Thuộc tính `android:inputType` cho phép bạn xác định nhiều hành vi cho phương tiện nhập liệu. Quan trọng nhất, nếu trường văn bản của bạn được dành cho đầu vào dạng văn bản cơ bản (chẳng hạn như cho một tin nhắn văn bản), bạn nên bật tính năng tự sửa chính tả với giá trị `"textAutoCorrect"`.

Bạn có thể kết hợp các hành vi và kiểu phương tiện nhập liệu khác nhau với thuộc tính `android:inputType`. Ví dụ, đây là cách để tạo một trường văn bản có chức năng viết hoa từ đầu tiên của một câu và tự sửa lỗi chính tả:

```
<EditText
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType=
        "textCapSentences|textAutoCorrect"
    .../>
```

Xác định action của phương tiện nhập liệu

Hầu hết các phương tiện nhập liệu mềm đều cung cấp một button action cho người dùng ở góc dưới cùng, phù hợp với trường văn bản hiện tại. Theo mặc định, hệ thống sử dụng button này cho cả action **Next (tiếp theo)** hoặc **Done (hoàn thành)**, trừ phi trường văn bản của bạn cho phép có nhiều dòng (chẳng hạn như với `android:inputType="textMultiLine"`); trong trường hợp này, button action là dấu ngắt dòng. Tuy nhiên, bạn có thể xác định thêm những action có khả năng phù hợp với trường văn bản hơn, chẳng hạn như **Send** hay **Go**.

Để xác định button action cho bàn phím, hãy sử dụng thuộc tính `android:imeOptions` với một giá trị action như `"actionSend"` hay `"actionSearch"`. Ví dụ:

Hình 4. Button Send xuất hiện khi bạn khai báo `android:imeOptions="actionSend"`.

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

Sau đó, bạn có thể lắng nghe phím được nhấn trên button action bằng cách xác định một `TextView.OnEditorActionListener` cho phần tử `EditText`. Trong trình

lắng nghe, hãy phản hồi ID của action IME (input method editor - trình chỉnh sửa phương tiện nhập liệu) phù hợp xác định trong lớp `EditorInfo`, chẳng hạn như `IME_ACTION_SEND`. Ví dụ:

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

6.2 Xử lý trạng thái hiển thị của phương tiện nhập liệu

Khi một trường văn bản nhận hoặc bị mất forcus Android hiển thị hoặc ẩn phương tiện nhập liệu (chẳng hạn như bàn phím trên màn hình) một cách tương ứng. Hệ thống cũng ra quyết định về cách thức giao diện người dùng và trường văn bản xuất hiện trên phương tiện nhập liệu. Ví dụ, khi không gian dọc trên màn hình bị ràng buộc, trường văn bản sẽ lấp đầy toàn bộ không gian trên phương tiện nhập liệu. Đối với hầu hết ứng dụng, các hành vi mặc định đã thỏa mãn nhu cầu.

Dù vậy, trong một số trường hợp, bạn có thể muốn điều khiển trạng thái hiển thị được của phương tiện nhập liệu trực tiếp hơn và xác định cách bạn muốn layout xuất hiện khi phương tiện nhập liệu hiện diện. Mục này sẽ giải thích cách điều khiển và phản hồi trạng thái hiển thị được của phương tiện nhập liệu.

Hiển thị phương tiện nhập liệu khi activity khởi động

Mặc dù đặt focus vào trường văn bản thứ nhất trong layout khi activity khởi động, song Android lại không hiển thị phương tiện nhập liệu. Hành vi này là phù hợp, bởi việc nhập văn bản có thể không phải là nhiệm vụ chính trong activity đó. Tuy nhiên, nếu việc nhập văn bản là nhiệm vụ chính (chẳng hạn như ở màn hình đăng nhập), bạn sẽ muốn phương tiện nhập liệu xuất hiện theo mặc định.

Để hiển thị phương tiện nhập liệu khi activity khởi động, hãy thêm thuộc tính `android:windowSoftInputMode` cho phần tử `<activity>` với giá trị `"stateVisible"`. Ví dụ:

```
<application ... >
    <activity
        android:windowSoftInputMode=”stateVisible” ... >
        ...
    </activity>
    ...
</application>
```

Ghi chú: Nếu thiết bị của người dùng có bàn phím cứng đi kèm, phương tiện nhập liệu mềm sẽ *không* xuất hiện.

Hiển thị phương tiện nhập liệu theo nhu cầu

Nếu có một phương tiện nhập liệu trong vòng đời của activity mà tại đây, bạn muốn đảm bảo phương tiện nhập liệu phải hiện diện, hãy sử dụng `InputMethodManager` để hiển thị nó.

Ví dụ, phương thức sau lấy một `View` mà tại đây, người dùng cần phải nhập liệu vào View này, gọi `requestFocus()` để focus vào View, sau đó gọi `showSoftInput()` để mở phương tiện nhập liệu:

```
public void showSoftKeyboard(View view) {
    if (view.requestFocus()) {
        InputMethodManager imm = (InputMethodManager)
                getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.showSoftInput(view, InputMethodManager.SHOW_IMPLICIT);
    }
}
```

Ghi chú: Một khi phương tiện nhập liệu đã hiển thị, bạn không nên ẩn nó bằng cách lập trình mã nguồn nữa. Hệ thống ẩn phương tiện nhập liệu khi người dùng hoàn thành tác vụ trong trường văn bản, hoặc người dùng có thể ẩn nó với một điều khiển của hệ thống (chẳng hạn như với button *Back*).

Xác định cách thức phản hồi của giao diện người dùng

Khi phương tiện nhập liệu xuất hiện trên màn hình, nó sẽ làm giảm không gian hiện có của giao diện người dùng trong ứng dụng. Hệ thống ra quyết định là nó nên điều chỉnh phần hiển thị của giao diện người dùng, nhưng có thể nó không thực hiện điều này ngay lúc đó. Để đảm bảo hành vi tốt nhất cho ứng dụng của mình, hãy chỉ ra cách bạn muốn hệ thống hiển thị giao diện người dùng của mình trong phần không gian còn lại.

Để khai báo cách phản hồi mà bạn mong muốn trong một activity, sử dụng thuộc tính `android:windowSoftInputMode` trong phần tử `<activity>` của file kê khai với một trong số những giá trị “adjust” (“điều chỉnh”).

Ví dụ, nhằm đảm bảo hệ thống thay đổi kích thước layout cho phần không gian hiện tại - điều này đảm bảo rằng mọi nội dung của layout đều có thể được truy cập - sử dụng "adjustResize":

```
<application ... >
    <activity
        android:windowSoftInputMode="adjustResize" ... >
        ...
    </activity>
    ...
</application>
```

Bạn có thể kết hợp thông số điều chỉnh ở trên với thông số về trạng thái hiển thị của phương thức nhập liệu ban đầu từ bên trên:

```
    <activity
        android:windowSoftInputMode="stateVisible|adjustResize" ... >
        ...
    </activity>
```

Xác định "adjustResize" là việc thực sự quan trọng nếu giao diện người dùng của bạn bao gồm các điều khiển mà người dùng có thể cần để truy cập ngay lập tức sau khi hoặc trong khi thực hiện nhập văn bản đầu vào. Ví dụ, nếu bạn sử dụng một layout tương đối để đặt một button bar ở cuối màn hình, hãy dùng "adjustResize" để thay đổi kích thước layout sao cho bar đó xuất hiện trên phương tiện nhập liệu.

6.3 Hỗ trợ điều hướng qua bàn phím

Ngoài các phương tiện nhập liệu mềm (chẳng hạn như bàn phím trên màn hình), Android còn hỗ trợ các bàn phím vật lý gắn vào thiết bị. Bàn phím không chỉ đem lại một chế độ thuận tiện cho việc nhập liệu văn bản mà còn cung cấp một cách thức để người dùng điều hướng và tương tác với ứng dụng. Mặc dù hầu hết thiết bị cầm tay như điện thoại đều sử dụng cảm ứng làm chế độ tương tác chính, song máy tính bảng và các thiết bị tương tự ngày càng trở nên phổ biến và nhiều người dùng thích sử dụng phụ kiện bàn phím gắn thêm vào.

Như nhiều thiết bị Android từng cung cấp loại trải nghiệm này, điều quan trọng là bạn phải tối ưu ứng dụng của mình để hỗ trợ tương tác qua bàn phím. Mục này mô tả cách bạn có thể hỗ trợ tốt hơn cho việc điều hướng với bàn phím.

> **Ghi chú:** Việc hỗ trợ điều hướng theo hướng trong ứng dụng của bạn rất quan trọng để đảm bảo người dùng có thể truy cập được mà không cần dùng các hỗ trợ trực quan. Việc hỗ trợ đầy đủ việc điều hướng theo hướng trong ứng dụng cũng giúp bạn kiểm thử giao diện người dùng với các công cụ như uiautomator.

Kiểm thử ứng dụng

Người dùng có thể đã duyệt ứng dụng của bạn bằng bàn phím, bởi vì hệ thống Android bật hầu hết các hành vi cần thiết theo mặc định.

Tất cả các widget tương tác cung cấp bởi framework Android (chẳng hạn như `Button` và `EditText`) đều có thể nhận focus. Điều này có nghĩa rằng, người dùng có thể điều hướng với các thiết bị điều khiển như D-pad hoặc bàn phím và mỗi widget sẽ sáng lên hoặc ngược lại, thay đổi giao diện của nó khi nó được focus vào.

Để kiểm thử ứng dụng của bạn, hãy:

1. Cài đặt ứng dụng trên một thiết bị có sẵn bàn phím phần cứng.

   Nếu thiết bị phần cứng của bạn không có bàn phím, hãy kết nối bàn phím Bluetooth hoặc một bàn phím USB (mặc dù không phải mọi thiết bị đều hỗ trợ phụ kiện USB).

   Bạn cũng có thể sử dụng trình giả lập (emulator) Android:

   1. Trong AVD Manager, nhấn vào **New Device** (thiết bị mới) hoặc chọn một profile có sẵn và nhấn **Clone** (tạo bản sao).
   2. Trong cửa sổ hiện ra, hãy đảm bảo rằng **Keyboard** và **DPad** được bật.

2. Để kiểm thử ứng dụng, hãy chỉ dùng phím Tab để điều hướng trên giao diện người dùng, đảm bảo rằng mỗi điều khiển trong đó đều nhận được focus khi cần.

   Tìm xem có thực thể nào được focus không như ý bạn muốn không.

3. Bắt đầu từ khi ứng dụng khởi động và thay vào đó, hãy dùng các phím điều hướng (phím mũi tên trên bàn phím) để duyệt ứng dụng.

   Từ mỗi phần tử có thể được focus trong giao diện người dùng, nhấn Up (lên), Down (xuống), Left (trái), Right (phải).

   Tìm xem có thực thể nào được focus không như ý bạn muốn không.

Nếu bạn gặp bất cứ thực thể nào mà tại đó khi bạn đang điều hướng với phím Tab hoặc điều khiển điều hướng không như ý muốn, hãy xác định vị trí nên được focus trong layout. Điều này sẽ được bàn luận ở mục sau.

Xử lý việc điều hướng dùng phím Tab

Khi người dùng điều hướng trên ứng dụng của bạn bằng phím Tab, hệ thống chuyển focus giữa các phần tử dựa trên thứ tự mà chúng xuất hiện trong layout. Ví dụ, nếu bạn sử dụng một layout tương đối (relative layout) và thứ tự các phần tử trên màn hình khác với trong file, bạn phải tự xác định thứ tự focus theo cách thủ công.

Ví dụ, trong layout dưới đây, hai button được sắp xếp ở bên phải và một trường văn bản được sắp xếp phía bên trái của button thứ hai. Để truyền focus từ button thứ nhất tới trường văn bản, sau đó là button thứ hai, layout phải xác định rõ thứ tự focus cho từng phần tử mà có thể focus vào với thuộc tính `android:nextFocusForward`:

```
<RelativeLayout ...>
    <Button
        android:id="@+id/button1"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:nextFocusForward="@+id/editText1"
        ... />
    <Button
        android:id="@+id/button2"
        android:layout_below="@id/button1"
        android:nextFocusForward="@+id/button1"
        ... />
    <EditText
        android:id="@id/editText1"
        android:layout_alignBottom="@+id/button2"
        android:layout_toLeftOf="@id/button2"
        android:nextFocusForward="@+id/button2"
        ...  />
    ...
</RelativeLayout>
```

Lúc này, thay vì chuyển focus từ `button1` tới `button2`, sau đó là `editText1`, việc chuyển focus phù hợp với giao diện trên màn hình: Từ `button1` tới `editText1` rồi tới `button2`.

Xử lý việc điều hướng theo hướng

Người dùng cũng có thể điều hướng ứng dụng của bạn bằng cách dùng các phím mũi tên trên bàn phím (hành vi tương tự như khi điều hướng với D-pad hoặc bi lăn). Hệ thống cung cấp một dự đoán tốt nhất cho view nên được focus, theo hướng đã xác định dựa trên layout của các view trên màn hình. Đôi khi, hệ thống có thể đoán sai.

Nếu hệ thống không truyền focus cho view thích hợp khi người dùng thao tác điều hướng theo hướng đã cho, hãy xác định view nào nên được focus với các thuộc tính sau:

- `android:nextFocusUp`.
- `android:nextFocusDown`.
- `android:nextFocusLeft`.
- `android:nextFocusRight`.

Mỗi thuộc tính chỉ định view kế tiếp nhận focus khi người dùng thực hiện điều hướng theo hướng bằng cách sử dụng ID của view. Ví dụ:

```
<Button
    android:id="@+id/button1"
    android:nextFocusRight="@+id/button2"
    android:nextFocusDown="@+id/editText1"
    ... />
<Button
    android:id="@id/button2"
    android:nextFocusLeft="@id/button1"
    android:nextFocusDown="@id/editText1"
    ... />
<EditText
    android:id="@id/editText1"
    android:nextFocusUp="@id/button1"
    ...  />
```

6.4 Xử lý các action từ bàn phím

Khi người dùng focus vào một view có văn bản chỉnh sửa được, chẳng hạn như một phần tử `EditText` và người dùng có một bàn phím cứng gắn vào, tất cả đầu vào sẽ được hệ thống xử lý. Tuy nhiên, nếu muốn chặn hoặc tự xử lý trực tiếp đầu vào từ bàn phím, bạn có thể thực thi các phương thức callback từ giao diện `KeyEvent.Callback`, chẳng hạn như `onKeyDown()` và `onKeyMultiple()`.

Cả hai lớp `Activity` và `View` đều thực thi giao diện `KeyEvent.Callback`; vì vậy, thường thì bạn nên ghi đè các phương thức callback trong phần mở rộng của những lớp đó, nếu thích hợp.

> **Ghi chú:** Khi xử lý các sự kiện từ bàn phím với lớp `KeyEvent` và những hàm API liên quan, bạn nên hy vọng rằng các sự kiện ấy sẽ chỉ tới từ bàn phím phần cứng. Đừng bao giờ trông cậy vào các sự kiện từ bất cứ phím ảo nào trên phương tiện nhập liệu mềm (một bàn phím trên màn hình chẳng hạn).

Xử lý các sự kiện phím đơn

Để xử lý một phím đơn được nhấn, hãy thực thi `onKeyDown()` hoặc `onKeyUp()` nếu thích hợp. Thường thì bạn nên sử dụng `onKeyUp()` nếu muốn đảm bảo chỉ nhận một sự kiện. Nếu người dùng nhấn và giữ button, `onKeyDown()` sẽ được gọi nhiều lần.

Ví dụ, phương thức sau sẽ phản hồi một số phím để điều khiển một trò chơi:

```
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
    switch (keyCode) {
        case KeyEvent.KEYCODE_D:
            moveShip(MOVE_LEFT);
            return true;
                case KeyEvent.KEYCODE_F:
```

```
            moveShip(MOVE_RIGHT);
            return true;
        case KeyEvent.KEYCODE_J:
            fireMachineGun();
            return true;
        case KeyEvent.KEYCODE_K:
            fireMissile();
            return true;
        default:
            return super.onKeyUp(keyCode, event);
    }
}
```

Xử lý các phím bổ trợ

Để phản hồi các sự kiện từ phím bổ trợ (modifier key), chẳng hạn như khi một phím được kết hợp với Shift hay Ctrl (Control), bạn có thể truy vấn (query) `KeyEvent` được truyền tới phương thức callback. Một vài phương thức cung cấp thông tin về phím bổ trợ, chẳng hạn như `getModifiers()` và `getMetaState()`. Tuy nhiên, giải pháp đơn giản nhất là kiểm tra xem liệu có đúng phím bổ trợ mà bạn đang quan tâm được nhấn hay không với các phương thức như `isShiftPressed()` và `isCtrlPressed()`.

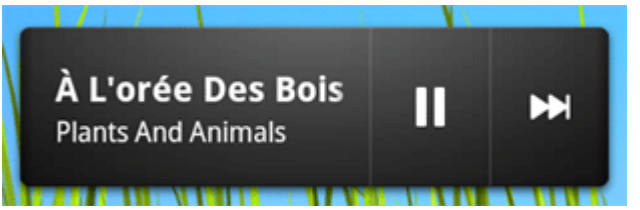
Ví dụ, sau đây là bài tập thực thi `onKeyDown()` cùng với một số xử lý bổ sung khi phím Shift được nhấn cùng với một trong số các phím:

```
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
    switch (keyCode) {
        ...
        case KeyEvent.KEYCODE_J:
            if (event.isShiftPressed()) {
                fireLaser();
            } else {
                fireMachineGun();
            }
            return true;
        case KeyEvent.KEYCODE_K:
            if (event.isShiftPressed()) {
                fireSeekingMissle();
            } else {
                fireMissile();
            }
            return true;
        default:
        return super.onKeyUp(keyCode, event);
```

```
    }
}
```

7. Widget của ứng dụng

Widget của ứng dụng là các view loại nhỏ trong ứng dụng đó, có thể được nhúng vào những ứng dụng khác (chẳng hạn như màn hình Home) và nhận cập nhật theo chu kỳ. Những view này được gọi là các Widget trong giao diện người dùng, bạn có thể phát hành chúng với một Widget provider (Widget provider). Thành phần ứng dụng có thể chứa các Widget khác được gọi là Widget chủ cho Widget của ứng dụng (App Widget host). Ảnh chụp màn hình bên dưới biểu diễn Widget của ứng dụng Music.



Tài liệu này mô tả cách công bố một Widget của ứng dụng bằng cách dùng một Widget provider. Bạn có thể tham khảo phần thảo luận về việc tự tạo `AppWidgetHost` để chứa các Widget của ứng dụng tại phần App Widget Host.

Thiết kế Widget

Để tham khảo thêm thông tin về cách thiết kế Widget cho ứng dụng, mời bạn đọc hướng dẫn về `Widget`.

Kiến thức cơ bản

Để tạo Widget cho ứng dụng, bạn cần có:

Đối tượng `AppWidgetProviderInfo`

> Mô tả siêu dữ liệu (metadata) cho một Widget, chẳng hạn như layout của Widget, tần suất cập nhật và lớp AppWidgetProvider. Có thể xác định điều này trong XML.

Thực thi lớp `AppWidgetProvider`

> Định nghĩa những phương thức cơ bản cho phép bạn lập trình giao diện với Widget của ứng dụng dựa trên các sự kiện broadcast. Thông qua đó, bạn sẽ nhận được các broadcast khi Widget của ứng dụng được cập nhật, được bật, bị vô hiệu hóa và bị xóa.

Layout của view

> Định nghĩa layout ban đầu cho Widget của ứng dụng, xác định trong XML.

Ngoài ra, bạn có thể thực thi một activity cấu hình cho Widget của ứng dụng. Đây là một Activity tùy chọn được đưa ra khi người dùng thêm Widget cho ứng dụng của bạn,

cho phép người đó chỉnh sửa các thiết lập trong Widget của ứng dụng vào thời điểm khởi tạo.

Mục dưới đây mô tả cách thiết lập mỗi thành phần đó.

Khai báo Widget của ứng dụng trong file kê khai

Trước hết, hãy khai báo lớp AppWidgetProvider trong file kê khai AndroidManifest.xml của ứng dụng. Ví dụ:

```
<receiver android:name="ExampleAppWidgetProvider" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
               android:resource="@xml/example_appwidget_info" />
</receiver>
```

Phần tử <receiver> yêu cầu thuộc tính android:name, thuộc tính này xác định AppWidgetProvider được Widget của ứng dụng dùng.

Phần tử <intent-filter> phải chứa một phần tử <action> với thuộc tính android:name. Thuộc tính này chỉ ra rằng AppWidgetProvider chấp nhận broadcast ACTION_APPWIDGET_UPDATE. Đây là broadcast duy nhất bạn phải khai báo một cách tường minh. Bộ AppWidgetManager sẽ tự động gửi tất cả các broadcast khác trong Widget của ứng dụng tới AppWidgetProvider, nếu cần.

Phần tử <meta-data> xác định tài nguyên AppWidgetProviderInfo và yêu cầu các thuộc tính sau:

- android:name - Xác định tên siêu dữ liệu. Sử dụng android.appwidget.provider để nhận diện dữ liệu dưới dạng trình mô tả (descriptor) AppWidgetProviderInfo.
- android:resource - Xác định vị trí tài nguyên AppWidgetProviderInfo.

Thêm siêu dữ liệu AppWidgetProviderInfo

AppWidgetProviderInfo xác định các đặc điểm cần thiết của một Widget cho ứng dụng, chẳng hạn như độ dài tối thiểu của layout, tài nguyên layout ban đầu, tần suất cập nhật Widget ứng dụng và một activity cấu hình (tùy chọn) để khởi động lúc tạo. Định nghĩa đối tượng AppWidgetProviderInfo trong tài nguyên XML bằng cách sử dụng một phần tử đơn <appwidget-provider> và lưu nó trong thư mục res/xml/ của dự án.

Ví dụ:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/
res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen|keyguard"
    android:initialKeyguardLayout="@layout/example_keyguard">
</appwidget-provider>
```

Dưới đây là tổng quan về các thuộc tính `<appwidget-provider>`:

- Giá trị cho các thuộc tính `minWidth` và `minHeight` xác định khoảng không gian tối thiểu mà Widget của ứng dụng chiếm *theo mặc định*. Màn hình Home định vị các Widget ứng dụng trong cửa sổ của nó dựa trên một khung lưới gồm những ô có chiều rộng và chiều cao xác định. Nếu giá trị chiều rộng hoặc chiều cao tối thiểu của Widget trong ứng dụng không khớp với kích thước các ô, kích thước Widget trong ứng dụng sẽ được làm tròn *lên* bằng kích thước của ô gần nhất.

Xem mục "App Widget Design Guidelines" ("Hướng dẫn thiết kế Widget cho ứng dụng) để biết thêm thông tin về việc thay đổi kích thước của Widget trong ứng dụng.

> **Ghi chú:** Để giúp cho Widget trong ứng dụng của bạn có thể dùng với nhiều loại thiết bị, kích thước tối thiểu của nó không nên lớn hơn 4 x 4 ô.

- Các thuộc tính `minResizeWidth` và `minResizeHeight` xác định kích thước tối thiểu tuyệt đối cho Widget của ứng dụng. Những giá trị này nên xác định kích thước mà tại đó, Widget của ứng dụng được chấp nhận hoặc ngược lại, không sử dụng được. Sử dụng những thuộc tính này cho phép người dùng thay đổi kích thước Widget tới một cỡ có thể nhỏ hơn kích thước mặc định của Widget được các thuộc tính minWidth và minHeight định nghĩa. Nội dung này được giới thiệu trong Android 3.1.

Xem phần Hướng dẫn thiết kế Widget cho ứng dụng (App Widget Design Guidelines) để biết thêm thông tin về việc thay đổi kích thước Widget trong ứng dụng.

Thuộc tính `updatePeriodMillis` xác định tần suất framework Widget trong ứng dụng nên yêu cầu một cập nhật (update) từ `AppWidgetProvider` bằng cách gọi phương thức callback `onUpdate()`. Cập nhật thực sự không đảm bảo xảy ra chính xác tại thời điểm có giá trị này; do đó, chúng tôi khuyên bạn nên cập nhật ít thường xuyên nhất nếu có thể - có thể là không quá một giờ mỗi lần để tiết kiệm pin. Bạn cũng có thể cho phép người dùng điều chỉnh tần suất bằng cách cấu hình - một số người muốn bộ đếm thời gian cho cổ phiếu (stock ticker) phải cập nhật 15 phút một lần, hoặc có thể chỉ là bốn lần mỗi ngày.

> **Ghi chú:** Nếu thiết bị đang ở chế độ "ngủ" khi đã đến thời điểm cập nhật (được xác định bởi `updatePeriodMillis`), thiết bị sẽ "thức dậy" để thực hiện cập nhật. Nếu bạn cập nhật không quá một lần mỗi giờ, điều này sẽ không gây vấn đề đáng kể đối với thời lượng pin. Tuy nhiên, nếu cần cập nhật với tần suất cao hơn và/hoặc không cần cập nhật khi thiết bị ở chế độ "ngủ", bạn có thể thực hiện cập nhật dựa trên báo thức (alarm), nó sẽ không đánh thức thiết bị. Để làm việc này, hãy đặt báo thức với một Intent mà AppWidgetProvider sẽ nhận, bằng cách sử dụng `AlarmManager`. Thiết lập loại báo thức thành `ELAPSED_REALTIME` hoặc `RTC`, tức là nó sẽ chỉ gửi báo thức khi thiết bị đang ở chế độ thức. Sau đó, thiết lập `updatePeriodMillis` thành `"0"`.

- Thuộc tính `initialLayout` chỉ tới tài nguyên layout xác định layout của Widget trong ứng dụng.
- Thuộc tính `configure` xác định `Activity` để chạy khi người dùng thêm Widget ứng dụng, nhằm giúp họ cấu hình các thuộc tính của Widget. Việc này là tùy chọn (đọc mục "Tạo activity cấu hình cho Widget trong ứng dụng" bên dưới).
- Thuộc tính `previewImage` xác định một ảnh xem trước (preview) của Widget trong ứng dụng sau khi nó được cấu hình, đó là thứ người dùng nhìn thấy khi chọn Widget ứng dụng. Nếu không có ảnh xem trước, thay vào đó người dùng sẽ nhìn thấy biểu tượng ứng dụng. Trường này tương ứng với thuộc tính `android:previewImage` trong phần tử `<receiver>` tại file `AndroidManifest.xml`. Tham khảo phần thảo luận về việc sử dụng previewImage ở mục "Thiết lập ảnh xem trước". Thuộc tính này được giới thiệu trong Android 3.0.
- Thuộc tính `autoAdvanceViewId` xác định view ID cho view con của Widget trong ứng dụng. View này nên được host của Widget tự động cập nhật. Thuộc tính này được giới thiệu trong Android 3.0.
- Thuộc tính `resizeMode` xác định các quy tắc mà tại đó, một Widget có thể được thay đổi kích thước. Bạn dùng thuộc tính này để tạo các Widget có khả năng thay đổi kích thước cho màn hình Home - theo chiều ngang, chiều dọc hoặc cả hai. Người dùng chạm và giữ vào một Widget để hiển thị bộ thay đổi kích thước (resize handle), sau đó kéo nó theo chiều ngang hoặc/và dọc để thay đổi kích thước trên khung lưới. Các giá trị cho thuộc tính `resizeMode` bao gồm "horizontal" (ngang), "vertical" (dọc) và "none" (không chọn giá trị nào). Để khai báo một Widget có khả năng thay đổi kích thước theo chiều ngang và dọc, hãy thiết lập giá trị là "horizontal|vertical". Thuộc tính này được giới thiệu trong Android 3.1.
- Thuộc tính `minResizeHeight` xác định chiều cao tối thiểu (theo dp) mà Widget có thể được thay đổi kích thước. Trường này không có tác dụng nếu lớn hơn `minHeight`, hoặc nếu việc thay đổi kích thước theo chiều dọc bị vô hiệu hóa (xem `resizeMode`). Thuộc tính này được giới thiệu trong Android 4.0.
- Thuộc tính `minResizeWidth` xác định chiều rộng tối thiểu (theo dp) mà Ưidget có thể được thay đổi kích thước. Trường này không có tác dụng nếu lớn hơn `minWidth`, hoặc nếu việc thay đổi kích thước theo chiều ngang bị vô hiệu hóa (xem `resizeMode`). Thuộc tính này được giới thiệu trong Android 4.0.

- Thuộc tính `widgetCategory` khai báo xem Widget trong ứng dụng của bạn có thể được hiển thị trên màn hình Home, màn hình khóa (lock screen) - còn gọi là bảo vệ bàn phím, hoặc cả hai hay không. Các giá trị cho thuộc tính này bao gồm "home_screen" và "keyguard". Một Widget được hiển thị trên cả hai màn hình trên cần đảm bảo theo hướng dẫn thiết kế cho cả hai lớp Widget. Để tham khảo thêm thông tin, xem mục "Bật Widget của ứng dụng trên màn hình khóa". Giá trị mặc định là "home_screen". Thuộc tính này được giới thiệu trong Android 4.2.
- Thuộc tính `initialKeyguardLayout` chỉ tới tài nguyên layout định nghĩa màn hình khóa cho layout Widget của ứng dụng. Nó hoạt động tương tự `android:initialLayout`, đồng thời cung cấp một layout có thể xuất hiện ngay lập tức cho tới khi Widget được khởi tạo và có thể cập nhật layout. Thuộc tính này được giới thiệu trong Android 4.2.

Xem lớp `AppWidgetProviderInfo` để biết thêm thông tin về các thuộc tính được phần tử `<appwidget-provider>` chấp nhận.

Tạo layout cho Widget của ứng dụng

Bạn phải định nghĩa một layout ban đầu cho Widget trong ứng dụng của mình trong file XML và lưu file vào thư mục res/layout/ của dự án. Bạn có thể thiết kế Widget cho ứng dụng bằng cách sử dụng các đối tượng View được liệt kê bên dưới, nhưng trước khi bắt đầu, hãy tìm hiểu mục "Hướng dẫn thiết kế Widget cho ứng dụng".

Nếu bạn đã quen với Layout, việc tạo layout cho Widget của ứng dụng rất đơn giản. Tuy nhiên, bạn phải nhận thức được rằng các layout Widget của ứng dụng được dựa trên `RemoteViews` vốn không hỗ trợ bất cứ loại layout hay Widget của view nào.

Một đối tượng RemoteViews (một Widget của ứng dụng) có thể hỗ trợ những lớp layout sau:

- `FrameLayout`.
- `LinearLayout`.
- `RelativeLayout`.
- `GridLayout`.

và các lớp Widget dưới đây:

- `AnalogClock`.
- `Button`.
- `Chronometer`.
- `ImageButton`.
- `ImageView`.
- `ProgressBar`.
- `TextView`.
- `ViewFlipper`.
- `ListView`.

- GridView.
- StackView.
- AdapterViewFlipper.

Con cháu của những lớp này không được hỗ trợ.

RemoteViews cũng hỗ trợ ViewStub, đó là một View vô hình, kích thước bằng không mà bạn có thể dùng để điền dần các tài nguyên layout vào thời điểm chạy.

Thêm lề cho Widget của ứng dụng

Nhìn chung, các Widget thường không nên mở rộng tới cạnh bên màn hình và không nên tràn sang (flush) những Widget khác; vì vậy, bạn nên thêm lề (margin) trên các cạnh xung quanh khung (frame) Widget.

Như ở Android 4.0, các Widget của ứng dụng được tự động thêm vùng đệm (padding) giữa khung Widget và khung viền (bounding box) của Widget để đưa ra phương án căn chỉnh tốt hơn so với những Widget khác cùng các biểu tượng trên màn hình Home của người dùng. Để tận dụng hành vi rất được khuyến khích này, hãy thiết lập targetSdkVersion trong ứng dụng của bạn thành 14 hoặc cao hơn.

Rất dễ để viết một layout đơn có các lề tùy chỉnh áp dụng cho những phiên bản trước của nền tảng; và không có các lề thêm vào cho Anroid 4.0 và cấp cao hơn:

1. Thiết lập `targetSdkVersion` của ứng dụng thành 14 hoặc cao hơn.
2. *Để tạo một layout giống như bên dưới, hãy tham khảo tài nguyên số đo (*dimension resource*) để thấy được các lề của nó:*

```
<FrameLayout
    android:layout_width=”match_parent”
    android:layout_height=”match_parent”
    android:padding=”@dimen/widget_margin”>
    <LinearLayout
        android:layout_width=”match_parent”
        android:layout_height=”match_parent”
        android:orientation=”horizontal”
        android:background=”@drawable/my_widget_background”>
        …
    </LinearLayout>
</FrameLayout>
```

3. Tạo hai tài nguyên số đo, một trong `res/values/` để cung cấp lề tùy chỉnh cho Widget của Android phiên bản trước 4.0, và một ở `res/values-v14/` để không cung cấp thêm vùng đệm cho Widget của Anroid 4.0:

   res/values/dimens.xml:

```
<dimen name=”widget_margin”>8dp</dimen>
```

res/values-v14/dimens.xml:

```
<dimen name=”widget_margin”>0dp</dimen>
```

Tùy chọn khác đơn giản hơn là xây dựng lề phụ cho các tập nền (background asset) dạng nine-patch (một dạng ảnh bitmap mở rộng) theo mặc định, đồng thời cung cấp những nine-patch (bộ 9 bản vá) khác không có lề cho các hàm API Cấp 14 hoặc cấp cao hơn.

Sử dụng lớp AppWidgetProvider

Lớp AppWidgetProvider mở rộng BroadcastReceiver dưới dạng một lớp thích hợp để xử lý các broadcast của Widget trong ứng dụng. AppWidgetProvider chỉ nhận các sự kiện broadcast có liên quan tới Widget trong ứng dụng, chẳng hạn như khi Widget được cập nhật, bị xóa, được bật và bị vô hiệu hóa. Khi những sự kiện broadcast này xảy ra, AppWidgetProvider nhận các lời gọi phương thức sau:

Bạn phải khai báo thực thi lớp AppWidgetProvider dưới dạng một broadcast receiver sử dụng phần tử `<receiver>` trong AndroidManifest (xem mục “Khai báo Widget của ứng dụng trong file kê khai” bên trên).

`onUpdate()`

Được gọi để cập nhật Widget trong ứng dụng theo chu kỳ xác định bởi thuộc tính `updatePeriodMillis` trong AppWidgetProviderInfo (xem mục “Thêm siêu dữ liệu AppWidgetProviderInfo” bên trên). Phương thức này cũng được gọi khi người dùng thêm Widget trong ứng dụng, vì vậy nó nên thực hiện các cài đặt cần thiết, chẳng hạn như xác định những trình xử lý sự kiện cho các View và khởi động một `Service` tạm thời, nếu cần. Tuy nhiên, nếu bạn đã khai báo một cấu hình cho activity, **phương thức này sẽ không được gọi** khi người dùng thêm Widget ứng dụng, nhưng sẽ được gọi cho những lần cập nhật sau. Cấu hình activity có trách nhiệm thực hiện cập nhật đầu tiên khi việc cấu hình hoàn tất. (Xem mục “Tạo activity cấu hình cho Widget của ứng dụng” bên dưới).

`onAppWidgetOptionsChanged()`

Được gọi khi Widget được đặt lần đầu tiên và tại bất cứ thời điểm nào Widget bị thay đổi kích thước. Bạn có thể sử dụng callback này để hiển thị hoặc ẩn nội dung dựa trên phạm vi kích thước của Widget. Bạn lấy phạm vi kích thước bằng cách gọi `getAppWidgetOptions()`, nó sẽ trả về một `Bundle` chứa:

- `OPTION_APPWIDGET_MIN_WIDTH` - Chứa ràng buộc (bound) dưới của chiều rộng hiện tại, tính theo đơn vị dp, của một thực thể Widget.
- `OPTION_APPWIDGET_MIN_HEIGHT` - Chứa ràng buộc dưới của chiều cao hiện tại, tính theo đơn vị dp, của một thực thể Widget.
- `OPTION_APPWIDGET_MAX_WIDTH` - Chứa ràng buộc trên của chiều rộng hiện tại, tính theo đơn vị dp, của một thực thể Widget.
- `OPTION_APPWIDGET_MAX_HEIGHT` - Chứa ràng buộc trên của chiều cao hiện tại, tính theo đơn vị dp, của một thực thể Widget.

Callback này được giới thiệu trong API Cấp 16 (Android 4.1). Nếu bạn thực thi callback này, hãy chắc chắn rằng ứng dụng của bạn không phụ thuộc vào nó, bởi callback sẽ không được gọi trên những thiết bị cũ hơn.

`onDeleted(Context, int[])`

Được gọi mỗi khi một Widget trong ứng dụng bị xóa từ host Widget của ứng dụng.

`onEnabled(Context)`

Được gọi khi một thể hiện Widget trong ứng dụng được tạo lần đầu tiên. Ví dụ, nếu người dùng thêm hai thể hiện của Widget trong ứng dụng, `onEnabled(Context)` chỉ được gọi vào lần đầu tiên. Nếu bạn cần mở một cơ sở dữ liệu mới hoặc thực hiện cài đặt khác chỉ cần xảy ra một lần trên tất cả các thể hiện Widget của ứng dụng, đây là nơi thích hợp để làm điều này.

`onDisabled(Context)`

Được gọi khi thể hiện cuối cùng của Widget ứng dụng bị xóa từ host Widget ứng dụng. Đây là nơi bạn nên dọn dẹp bất cứ công việc nào đã hoàn thành trong `onEnabled(Context)`, chẳng hạn như xóa một cơ sở dữ liệu tạm thời.

`onReceive(Context, Intent)`

Được gọi cho mỗi broadcast và trước mỗi phương thức callback bên trên. Thông thường, bạn không phải thực thi phương thức này bởi khi thực thi, AppWidgetProvider sẽ mặc định lọc tất cả các broadcast Widget trong ứng dụng và gọi những phương thức trên nếu phù hợp.

AppWidgetProvider quan trọng nhất là `onUpdate()`, vì callback này được gọi khi mỗi Widget trong ứng dụng được thêm vào một host (trừ phi bạn sử dụng một activity cấu hình). Nếu Widget trong ứng dụng của bạn chấp nhận bất cứ sự kiện tương tác với người dùng nào, bạn phải đăng ký các trình xử lý sự kiện trong callback này. Nếu Widget trong ứng dụng không tạo các file hoặc cơ sở dữ liệu tạm thời hoặc thực hiện công việc khác đòi hỏi phải dọn dẹp (clean-up), `onUpdate()` có thể là phương thức callback duy nhất bạn cần xác định. Ví dụ, nếu muốn một Widget trong ứng dụng với một button khởi động activity khi nhấn vào, bạn có thể sử dụng thực thi dưới đây của AppWidgetProvider:

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {

public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
       final int N = appWidgetIds.length;
// Thực hiện quy trình vòng lặp này đối với mỗi Widget của ứng dụng
// thuộc về provider này
       for (int i=0; i<N; i++) {
           int appWidgetId = appWidgetIds[i];

           // Tạo một Intent khởi động ExampleActivity
           Intent intent = new Intent(context, ExampleActivity.class);
```

```
            PendingIntent pendingIntent =
                        PendingIntent.getActivity(context, 0, intent, 0);

            // Lấy layout cho Widget của ứng dụng và gắn một trình lắng
            // nghe sự kiện on-click cho button
            RemoteViews views = new RemoteViews(context.getPackageName(),
                                    R.layout.appwidget_provider_layout);
            views.setOnClickPendingIntent(R.id.button, pendingIntent);

            // Ra lệnh cho AppWidgetManager thực hiện một cập nhật trên
            // widget của ứng dụng hiện tại
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

AppWidgetProvider này chỉ định nghĩa phương thức `onUpdate()` cho mục đích xác định một `PendingIntent` khởi động `Activity` và gắn nó với button của Widget trong ứng dụng bằng `setOnClickPendingIntent(int, PendingIntent)`. Lưu ý, callback này bao gồm một vòng lặp qua từng mục trong `appWidgetIds`, đó là một mảng các ID xác định mỗi Widget trong ứng dụng do provider này tạo ra. Bằng cách này, nếu người dùng tạo nhiều thể hiện của Widget trong ứng dụng, tất cả các thể hiện này sẽ được cập nhật cùng lúc. Tuy nhiên, chỉ có một lịch trình (schedule) `updatePeriodMillis` được quản lý cho tất cả các thể hiện của Widget trong ứng dụng. Ví dụ, nếu lịch trình cập nhật được xác định cứ hai giờ một lần, và thể hiện thứ hai của Widget trong ứng dụng được thêm vào một giờ sau thể hiện thứ nhất, thì chúng đều được cập nhật vào thời điểm do thể hiện thứ nhất xác định còn thời điểm cập nhật thứ hai sẽ bị bỏ qua (chúng đều được cập nhật cứ hai giờ một lần thay vì một giờ một lần).

> **Ghi chú:** Do `AppWidgetProvider` là mở rộng của `BroadcastReceiver`, nên tiến trình của bạn không được đảm bảo sẽ tiếp tục chạy sau khi các phương thức callback trả về kết quả (xem `BroadcastReceiver` để biết thêm thông tin về vòng đời broadcast). Nếu tiến trình cài đặt Widget trong ứng dụng của bạn có thể kéo dài trong vài giây (ví dụ như vào lúc thực hiện các yêu cầu Web) và bạn yêu cầu tiến trình đó tiếp tục, bạn nên khởi động một Service trong phương thức `onUpdate()`. Trong phạm vi Service này, bạn có thể thực hiện cập nhật của riêng mình cho Widget trong ứng dụng mà không phải lo `AppWidgetProvider` sẽ bị đóng do lỗi ứng dụng không phản hồi (ANR - Application Not Responding). Tham khảo `AppWidgetProvider` trong Wiktionary để xem xét ví dụ về một Widget ứng dụng chạy với một Service.

Ngoài ra, bạn cũng cần xem xét lớp mẫu ExampleAppWidgetProvider.java.

Nhận các Intent broadcast của Widget trong ứng dụng

`AppWidgetProvider` không chỉ là một lớp thuận tiện. Nếu muốn nhận trực tiếp các broadcast của Widget trong ứng dụng, bạn có thể thực thi `BroadcastReceiver` của mình hoặc ghi đè callback `onReceive(Context, Intent)`. Các Intent bạn cần quan tâm:

- ACTION_APPWIDGET_UPDATE.
- ACTION_APPWIDGET_DELETED.
- ACTION_APPWIDGET_ENABLED.
- ACTION_APPWIDGET_DISABLED.
- ACTION_APPWIDGET_OPTIONS_CHANGED.

Tạo activity cấu hình cho Widget của ứng dụng

Nếu muốn người dùng cấu hình các thiết lập khi họ thêm một Widget ứng dụng mới, bạn có thể tạo một activity cấu hình cho Widget ứng dụng. Activity này sẽ được tự động kích hoạt bởi host Widget ứng dụng (App Widget host), đồng thời cho phép người dùng cấu hình các thiết lập có sẵn cho Widget ứng dụng vào thời điểm tạo, chẳng hạn như màu sắc, kích thước, tần suất cập nhật và những thiết lập chức năng khác cho Widget của ứng dụng.

Activity cấu hình nên được khai báo dưới dạng một activity thông thường trong file kê khai của Android. Tuy nhiên, activity cấu hình sẽ được host Widget của ứng dụng khởi động bằng action ACTION_APPWIDGET_CONFIGURE, do đó activity phải chấp nhận Intent này. Ví dụ:

```
<activity android:name=".ExampleAppWidgetConfigure">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
    </intent-filter>
</activity>
```

Đồng thời, activity phải được khai báo trong file XML AppWidgetProviderInfo với thuộc tính `android:configure` (xem mục "Thêm siêu dữ liệu cho AppWidgetProviderInfo" bên trên). Ví dụ, activity cấu hình có thể được khai báo như sau:

```
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

Lưu ý, activity được khai báo với namespace đầy đủ, bởi activity này sẽ được tham chiếu từ bên ngoài phạm vi của gói.

Đó là tất cả những gì bạn cần để khởi động với một activity cấu hình. Bây giờ, thứ bạn cần chính là một activity thực thụ. Tuy nhiên, có hai điều quan trọng bạn cần ghi nhớ khi thực thi activity:

- Host Widget của ứng dụng gọi activity cấu hình và activity đó nên luôn trả về một kết quả. Kết quả nên bao gồm ID Widget của ứng dụng truyền bởi Intent khởi động activity (được lưu trong phần phụ của Intent như EXTRA_APPWIDGET_ID).

- Phương thức `onUpdate()` **sẽ không được gọi** khi Widget của ứng dụng được tạo (hệ thống sẽ không gửi broadcast ACTION_APPWIDGET_UPDATE khi một activity cấu hình được kích hoạt). Trách nhiệm của activity cấu hình là yêu cầu một cập nhật từ AppWidgetManager khi Widget của ứng dụng được tạo lần đầu. Tuy nhiên, `onUpdate()` sẽ được gọi cho những cập nhật lần sau và chỉ bỏ qua vào lần đầu tiên.

Xem các đoạn mã nhỏ ở mục dưới đây để thấy ví dụ về cách trả về kết quả từ cấu hình và cập nhật Widget của ứng dụng.

Cập nhật Widget của ứng dụng từ activity cấu hình

Khi một Widget ứng dụng dùng một activity cấu hình, trách nhiệm của activity đó là cập nhật Widget ứng dụng khi việc cấu hình hoàn tất. Bạn có thể làm điều này bằng cách yêu cầu một cập nhật trực tiếp từ `AppWidgetManager`.

Sau đây là tổng quan về quy trình cập nhật cho Widget trong ứng dụng và đóng activity cấu hình:

1. Trước hết, lấy ID của Widget trong ứng dụng từ Intent khởi động activity:

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
    mAppWidgetId = extras.getInt(
        AppWidgetManager.EXTRA_APPWIDGET_ID,
        AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

2. Thực hiện việc cấu hình Widget trong ứng dụng của bạn.
3. Khi việc cấu hình hoàn tất, lấy một thể hiện của AppWidgetManager bằng cách gọi getInstance(Context):

```
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

4. Cập nhật Widget ứng dụng với một layout `RemoteViews` bằng cách gọi `updateAppWidget(int, RemoteViews)`:

```
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

5. Cuối cùng, tạo Intent trả về, thiết lập Intent này với kết quả của activity, sau đó kết thúc activity:

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

> **Mách nhỏ:** Khi activity cấu hình mở lần đầu, hãy thiết lập kết quả activity là RESULT_CANCELED. Bằng cách này, nếu người dùng thoát khỏi activity trước khi đi đến bước cuối, host Widget ứng dụng sẽ được thông báo rằng việc cấu hình bị hủy và Widget ứng dụng sẽ không được thêm vào.

Xem lớp mẫu ExampleAppWidgetConfigure.java trong ApiDemos.

Thiết lập ảnh xem trước

Android 3.0 giới thiệu trường `previewImage` xác định một bản xem trước cho Widget của ứng dụng. Bản xem trước này được hiển thị tới người dùng từ bộ chọn Widget (Widget picker). Nếu trường này không được cung cấp giá trị, biểu tượng của Widget trong ứng dụng sẽ được dùng làm bản xem trước.

Đây là cách bạn xác định thiết lập này trong XML:

```
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:previewImage="@drawable/preview">
</appwidget-provider>
```

Nhằm hỗ trợ việc tạo một ảnh xem trước cho Widget ứng dụng của bạn (để xác định trong trường `previewImage`), trình giả lập Android bao gồm một ứng dụng có tên "Widget Preview". Để tạo ảnh xem trước, hãy khởi động ứng dụng này, chọn Widget cho ứng dụng của bạn và thiết lập Widget theo cách bạn muốn ảnh xem trước xuất hiện, sau đó lưu nó lại rồi đặt vào tài nguyên drawable trong ứng dụng của bạn.

Bật Widget của ứng dụng trên màn hình khóa

Android 4.2 giới thiệu tới người dùng khả năng thêm các Widget vào màn hình khóa. Để cho thấy Widget trong ứng dụng của bạn đã sẵn sàng để dùng trên màn hình khóa, hãy khai báo thuộc tính `android:widgetCategory` trong file XML xác định `AppWidgetProviderInfo` của bạn. Thuộc tính này hỗ trợ hai giá trị, đó là "home_screen" và "keyguard". Một Widget của ứng dụng có thể khai báo hỗ trợ cho một trong hai, hoặc cả hai giá trị trên.

Theo mặc định, mọi Widget ứng dụng đều hỗ trợ việc đặt trên màn hình Home, vì vậy "home_screen" là giá trị mặc định cho thuộc tính `android:widgetCategory`. Nếu bạn muốn Widget ứng dụng của mình sẵn sàng cho màn hình khóa, hãy thêm giá trị "keyguard":

```
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:widgetCategory="keyguard|home_screen">
</appwidget-provider>
```

Nếu bạn khai báo một Widget có thể hiển thị được trên cả màn hình khóa lẫn màn hình Home, khả năng là bạn sẽ muốn tùy chỉnh Widget tùy theo vị trí mà Widget được hiển thị. Ví dụ, bạn có thể tạo một file layout riêng cho màn hình bảo vệ bàn phím (màn hình khóa) và màn hình Home. Bước tiếp theo là phát hiện hạng mục Widget vào thời điểm chạy và phản hồi phù hợp. Bạn có thể phát hiện ứng dụng của mình đang ở màn hình khóa hay màn hình Home bằng cách gọi `getAppWidgetOptions()` để lấy các tùy chọn Widget dưới dạng một Bundle. Bundle trả về sẽ bao gồm khóa `OPTION_APPWIDGET_HOST_CATEGORY`, mà giá trị của nó sẽ là `WIDGET_CATEGORY_HOME_SCREEN` hoặc `WIDGET_CATEGORY_KEYGUARD`. Giá trị này được xác định bởi host mà tại đó, Widget được gắn vào. Trong `AppWidgetProvider`, bạn có thể kiểm tra hạng mục của Widget, ví dụ như:

```
AppWidgetManager appWidgetManager;
int widgetId;
Bundle myOptions = appWidgetManager.getAppWidgetOptions (widgetId);

// Lấy giá trị của OPTION_APPWIDGET_HOST_CATEGORY
int category = myOptions.getInt
              (AppWidgetManager.OPTION_APPWIDGET_HOST_CATEGORY, -1);

// Nếu giá trị là WIDGET_CATEGORY_KEYGUARD, đây sẽ là Widget màn hình khóa
boolean isKeyguard = category ==
                     AppWidgetProviderInfo.WIDGET_CATEGORY_KEYGUARD;
```

Khi đã biết hạng mục của Widget, bạn có thể tùy chọn tải một layout nền khác hoặc thiết lập các thuộc tính khác,... Ví dụ:

```
int baseLayout = isKeyguard ? R.layout.keyguard_widget_layout :
                                              R.layout.widget_layout;
```

Bạn cũng nên xác định một layout khởi tạo cho Widget ứng dụng khi nằm trên màn hình khóa với thuộc tính `android:initialKeyguardLayout`. Ở đây cũng tương tự như `android:initialLayout`, trong đó cung cấp một layout có thể xuất hiện ngay tức thì cho tới khi Widget được khởi tạo và có thể cập nhật layout.

Các hướng dẫn thay đổi kích thước

Khi một Widget được host trên màn hình khóa, framework sẽ bỏ qua các trường `minWidth`, `minHeight`, `minResizeWidth` và `minResizeHeight`. Nếu một Widget đồng thời là Widget của màn hình Home, những tham số này vẫn cho thấy chúng rất cần thiết khi được sử dụng ở màn hình Home; tuy nhiên, chúng sẽ bị bỏ qua cho nhiều mục đích ở màn hình khóa.

Chiều rộng của một Widget cho màn hình khóa luôn lấp đầy không gian được cấp. Đối với chiều cao của một Widget cho màn hình khóa, bạn có các tùy chọn sau:

- Nếu Widget không tự đánh dấu (mark) nó dưới dạng có thể thay đổi kích thước theo chiều dọc (`android:resizeMode="vertical"`), chiều cao của nó luôn là "small":
  - Trên thiết bị ở chế độ dọc, "small" được xác định là phần không gian còn lại khi một giao diện người dùng mở khóa (unlock) được hiển thị.
  - Trên máy tính bảng và điện thoại ở chế độ ngang, "small" được thiết lập tùy theo cơ sở của từng thiết bị.
- Nếu Widget tự đánh dấu nó dưới dạng có thể thay đổi kích thước chiều dọc, chiều cao của Widget sẽ là "small" trên điện thoại ở chế độ dọc và giao diện người dùng mở khóa. Trong tất cả các trường hợp khác, Widget sẽ điều chỉnh kích thước để lấp đầy chiều cao có thể.

Sử dụng Widget của ứng dụng với các collection

Android 3.0 giới thiệu các Widget của ứng dụng cùng với các collection. Những dạng Widget ứng dụng này dùng `RemoteViewsService` để hiển thị các collection được lấy từ dữ liệu từ xa (remote data), chẳng hạn như từ content provider. Dữ liệu do `RemoteViewsService` cung cấp được biểu diễn trong Widget ứng dụng dùng một trong những kiểu view dưới đây, chúng ta gọi là các "collection view":

`ListView`

View hiển thị các mục trong một danh sách cuộn dọc (vertically scrolling list). Ví dụ, Widget của ứng dụng Gmail.

`GridView`

View hiển thị các mục trong một lưới cuộn hai chiều (two-dimensional scrolling grid). Ví dụ, Widget của ứng dụng Bookmarks.

`StackView`

View dạng xếp chồng (stacked card view), tại đây người dùng có thể vuốt để đẩy View lên hoặc xuống và sẽ nhìn thấy View trước hoặc sau View này. Ví dụ, các Widget của ứng dụng YouTube và Books.

`AdapterViewFlipper`

Một `ViewAnimator` đơn giản hỗ trợ adapter, tạo hiệu ứng động giữa hai hay nhiều view. Chỉ có một view con được hiển thị tại một thời điểm.

Như đã đề cập ở trên, những collection view này hiển thị các collection do dữ liệu từ xa lấy về. Điều này có nghĩa là chúng sử dụng một `Adapter` để gắn giao diện người dùng với dữ liệu. Một `Adapter` gắn các mục đơn của một tập dữ liệu thành những đối tượng `View` đơn. Do các collection view này được adapter lấy về, nên framework Android phải bao gồm kiến trúc phụ để hỗ trợ sử dụng chúng trong các Widget. Trong trường hợp của một Widget, `Adapter` được

thay thế bởi `RemoteViewsFactory`, đây đơn giản là đối tượng bao một lớp mỏng quanh giao diện `Adapter`. Khi yêu cầu một mục cụ thể trong collection, `RemoteViewsFactory` tạo và trả về mục cho collection dưới dạng một đối tượng `RemoteViews`. Để bao gồm một collection view trong Widget ứng dụng của mình, bạn phải thực thi `RemoteViewsService` và `RemoteViewsFactory`.

RemoteViewsService là một service cho phép một adapter từ xa yêu cầu các đối tượng `RemoteViews`. `RemoteViewsFactory` là một giao diện cho adapter giữa một collection view (chẳng hạn như `ListView`, `GridView`,...) và dữ liệu bên dưới view đó. Từ mẫu Widget StackView, sau đây là một ví dụ về đoạn mã mẫu có sẵn mà bạn dùng để thực thi service và giao diện này:

```
public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent)
    {
        return new
StackRemoteViewsFactory(this.getApplicationContext(), intent);
    }
}

class StackRemoteViewsFactory implements
                      RemoteViewsService.RemoteViewsFactory {

// ... bao gồm các phương thức tương tự adapter ở đây.
// Xem mẫu StackView Widget.

}
```

Ứng dụng mẫu

Mã nguồn trích đoạn trong mục này được lấy từ ứng dụng mẫu Widget StackView:

Mẫu này chứa một ngăn xếp gồm 10 view, hiển thị các giá trị từ "0!" tới" 9!". Widget của ứng dụng mẫu có những hành vi chính sau:

- Người dùng có thể lướt dọc view ở trên cùng của wiget ứng dụng để hiển thị view tiếp theo hoặc sau đó. Đây là một hành vi của StackView dựng sẵn.

- Nếu không có bất cứ tương tác nào với người dùng, Widget của ứng dụng tự động duyệt qua các view của nó theo trình tự, giống như một show trình chiếu (slide show). Đó là do thiết lập `android:autoAdvanceViewId="@id/stack_view"` trong file `res/xml/stackwidgetinfo.xml`. Thiết lập này áp dụng cho ID của view, trong trường hợp này là ID của view ngăn xếp.
- Nếu người dùng chạm vào view trên cùng, Widget của ứng dụng sẽ hiển thị thông điệp `Toast` "Touched view *n*" ("Đã chạm view *n*"), trong đó *n* là chỉ số (vị trí) của view đã được chạm. Để tham khảo thêm phần thảo luận về cách thực thi điều này, xem mục "Thêm hành vi cho các mục riêng".

Thực thi Widget của ứng dụng với các collection

Để thực thi một Widget của ứng dụng với các collection, bạn phải tuân thủ một số bước cơ bản. Mục dưới đây mô tả thêm những bước bạn cần thực hiện để thực thi một Widget của ứng dụng với các collection.

Kê khai Widget của ứng dụng với các collection

Ngoài những điều kiện được liệt kê trong mục "Khai báo Widget của ứng dụng trong file kê khai", để có thể gắn các Widget của ứng dụng với các collection vào `RemoteViewsService` của mình, bạn phải khai báo service trong file kê khai với quyền `BIND_REMOTEVIEWS`. Điều này sẽ giúp ngăn các ứng dụng khác tự do truy cập vào dữ liệu trong Widget ứng dụng của bạn. Ví dụ, khi tạo một Widget ứng dụng dùng `RemoteViewsService` để tạo một collection view, mục kê khai trông sẽ như sau:

```
<service android:name="MyWidgetService"
...
android:permission="android.permission.BIND_REMOTEVIEWS" />
```

Dòng `android:name="MyWidgetService"` tham chiếu tới lớp con của `RemoteViewsService`.

Layout cho Widget của ứng dụng với các collection

Yêu cầu chính đối với file XML layout cho Widget trong ứng dụng của bạn là phải chứa một trong những collection view sau: `ListView`, `GridView`, `StackView` hoặc `AdapterViewFlipper`. Đây là `widget_layout.xml` cho ứng dụng mẫu Widget StackView:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <StackView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/stack_view"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:loopViews="true" />
    <TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/empty_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:background="@drawable/widget_item_background"
        android:textColor="#ffffff"
        android:textStyle="bold"
        android:text="@string/empty_view_text"
        android:textSize="20sp" />
</FrameLayout>
```

Lưu ý, các view trống phải là anh em của collection view để những view trống này có thể biểu diễn trạng thái trống.

Ngoài file layout cho toàn bộ Widget ứng dụng của mình, bạn phải tạo file layout khác xác định layout cho từng mục trong collection (ví dụ, một layout cho mỗi quyển sách trong collection sách). Ví dụ, ứng dụng mẫu Widget StackView chỉ có một file layout là `widget_item.xml`, do tất cả các mục đều sử dụng layout giống nhau. Tuy nhiên, ứng dụng mẫu Widget StackView có hai file layout là `dark_widget_item.xml` và `light_widget_item.xml`.

Lớp AppWidgetProvider cho Widget của ứng dụng với các collection

Giống như với một Widget của ứng dụng thông thường, phần lớn mã trong lớp con `AppWidgetProvider` của bạn thường nằm trong `onUpdate()`. Khác biệt lớn trong việc thực thi cho `onUpdate()` khi tạo một Widget của ứng dụng với các collection là bạn phải gọi `setRemoteAdapter()`. Điều này cho biết collection view, nơi lấy dữ liệu của nó. Sau đó, `RemoteViewsService` có thể trả về thực thi của `RemoteViewsFactory`, và Widget có thể đưa ra dữ liệu phù hợp. Khi gọi phương thức này, bạn phải truyền một Intent chỉ tới thực thi của `RemoteViewsService` và ID của Widget ứng dụng sẽ xác định Widget ứng dụng cần cập nhật.

Ví dụ, đây là cách mẫu Widget StackView thực thi phương thức callback `onUpdate()` để thiết lập `RemoteViewsService` thành adapter từ xa cho collection Widget của ứng dụng:

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
    // cập nhật mỗi Widget của ứng dụng với một adapter từ xa
    for (int i = 0; i < appWidgetIds.length; ++i) {

        // Thiết lập Intent khởi động StackViewService
       // cung cấp các view cho collection này.
        Intent intent = new Intent(context, StackWidgetService.class);
        // Thêm ID cho Widget của ứng dụng vào phần phụ của Intent.
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                                appWidgetIds[i]);
        intent.setData(Uri.parse(intent.toUri
                                        (Intent.URI_INTENT_SCHEME)));
        // Khởi tạo đối tượng RemoteViews cho layout
        // của Widget trong ứng dụng.
        RemoteViews rv = new RemoteViews
                  (context.getPackageName(), R.layout.widget_layout);
        // Thiết lập đối tượng RemoteViews để sử dụng adapter RemoteViews.

        // Adapter này kết nối tới một RemoteViewsService thông qua
        // Intent cụ thể. Đây là cách bạn tạo dữ liệu.
        rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view, intent);

        // View trống được hiển thị khi collection không có mục nào.
        // Nó nên nằm trong cùng layout được dùng để khởi tạo
        // đối tượng RemoteViews ở trên.

        rv.setEmptyView(R.id.stack_view, R.id.empty_view);

        //
        // Thực hiện thêm các xử lý cụ thể cho Widget ứng dụng này...
        //

        appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
    }
    super.onUpdate(context, appWidgetManager, appWidgetIds);
}
```

Lớp RemoteViewsService

Dữ liệu lưu trữ lâu dài

Về lâu dài, bạn không thể trông cậy vào một thể hiện của service hoặc bất cứ dữ liệu nào mà thể hiện này chứa. Vì vậy, bạn đừng nên chứa bất kỳ dữ liệu nào trong `RemoteViewsService` (trừ phi nó ở dạng tĩnh). Nếu bạn muốn dữ liệu trong Widget của ứng dụng được lưu trữ lâu dài, cách tốt nhất là dùng một `ContentProvider` mà dữ liệu của nó được lưu trữ trong toàn bộ vòng đời của tiến trình.

Như đã mô tả ở trên, lớp con RemoteViewsService cung cấp RemoteViewsFactory được dùng để tạo collection view từ xa.

Cụ thể, bạn cần thực hiện những bước sau:

Tạo lớp con RemoteViewsService. RemoteViewsService là service mà qua đó, một adapter từ xa có thể yêu cầu RemoteViews.

Trong lớp con RemoteViewsService của bạn, hãy bao gồm một lớp có thể thực thi giao diện RemoteViewsFactory. RemoteViewsFactory là một giao diện cho adapter giữa một collection view từ xa (chẳng hạn như ListView, GridView,...) và dữ liệu bên dưới view đó. Việc thực thi của bạn có trách nhiệm phải tạo ra một đối tượng RemoteViews cho từng mục trong tập dữ liệu. Giao diện này là một lớp mỏng bao bên ngoài Adapter.

Nội dung chính của việc thực thi RemoteViewsService là RemoteViewsFactory của nó, như mô tả dưới đây.

Giao diện RemoteViewsFactory

Lớp tùy chỉnh thực thi giao diện RemoteViewsFactory của bạn cung cấp cho Widget ứng dụng những dữ liệu chứa trong các mục trong collection của nó. Để làm việc này, nó kết hợp file XML layout của Widget ứng dụng với một nguồn dữ liệu. Nguồn dữ liệu này có thể là bất kỳ, từ một cơ sở dữ liệu tới một mảng đơn giản. Trong ứng dụng mẫu Widget StackView, nguồn dữ liệu là một mảng của `WidgetItems`. Các hàm RemoteViewsFactory đóng vai trò là adapter dán dữ liệu vào collection view từ xa.

Hai phương thức quan trọng nhất bạn cần thực thi cho lớp con RemoteViewsFactory là `onCreate()` và `getViewAt()`.

Hệ thống gọi `onCreate()` khi tạo đối tượng factory lần đầu. Đây là nơi bạn thiết lập bất kỳ kết nối và/hoặc con trỏ nào tới nguồn dữ liệu của bạn. Ví dụ, ứng dụng mẫu Widget StackView dùng `onCreate()` để khởi tạo một mảng các đối tượng `WidgetItem`. Khi Widget của ứng dụng hoạt động, hệ thống truy cập vào những đối tượng này bằng cách sử dụng chỉ số vị trí của chúng trong mảng và văn bản chúng chứa được hiển thị.

Đây là một đoạn trích từ ứng dụng mẫu Widget StackView, phần phương thức `OnCreate()` của lớp thực thi `RemoteViewsFactory`:

```
class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItems = new ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
        mAppWidgetId =
        intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
```

```
                                AppWidgetManager.INVALID_APPWIDGET_ID);
    }

    public void onCreate() {
        // Trong onCreate(), bạn thiết lập bất kỳ
        // kết nối/con trỏ nào cho nguồn dữ liệu của mình.
        // Hoạt động nặng ví dụ như tải về hoặc tạo nội dung,
        // nên được truyền tới onDataSetChanged()
        // hoặc getViewAt(). Nếu chạy quá 20 giây,
        // lời gọi này sẽ trả về trong một ANR.
        for (int i = 0; i < mCount; i++) {
            mWidgetItems.add(new WidgetItem(i + "!"));
        }
        ...
    }
...
```

Trong `RemoteViewsFactory`, phương thức `getViewAt()` trả về một đối tượng `RemoteViews` tương ứng với `position` cụ thể trong tập dữ liệu. Đây là một đoạn trích từ ứng dụng mẫu Widget StackView của lớp thực thi `RemoteViewsFactory`:

```
public RemoteViews getViewAt(int position) {
    // Xây dựng một mục các view từ xa dựa trên mục Widget của ứng dụng
    // trong file XML, và thiết lập văn bản dựa trên vị trí.
    RemoteViews rv = new RemoteViews(mContext.getPackageName(),
R.layout.widget_item);
    rv.setTextViewText(R.id.widget_item, mWidgetItems.get(position).text);
    ...
    // Trả về đối tượng các view từ xa.
    return rv;
}
```

Thêm hành vi cho các mục riêng

Các mục trên cho bạn biết cách để gán dữ liệu vào collection của Widget trong ứng dụng của bạn. Nhưng nếu bạn muốn thêm động một hành vi vào từng mục riêng lẻ trong collection view thì sao?

Như đã trình bày ở mục "Sử dụng lớp AppWidgetProvider", thường thì bạn dùng `setOnClickPendingIntent()` để thiết lập một hành vi click cho đối tượng - chẳng hạn như để khiến một button có thể kích hoạt một `Activity`. Tuy nhiên, phương pháp này không được cho phép đối với các view con trong một mục collection riêng lẻ (để làm rõ điều này, bạn có thể dùng `setOnClickPendingIntent()` để thiết lập một button toàn cục (global button) trong Widget ứng dụng của Gmail, button này sẽ khởi động ứng dụng, nhưng không phải trên từng mục đơn lẻ trong danh sách). Thay vào đó, để thêm hành vi click cho các mục đơn lẻ trong một collection, bạn sử dụng

`setOnClickFillInIntent()`. Điều này sẽ thiết lập một Intent mẫu cho Intent ở trạng thái chờ (pending Intent) cho collection view của bạn, sau đó thiết lập một Intent trên mỗi mục trong collection thông qua `RemoteViewsFactory`.

Mục này sử dụng ứng dụng mẫu Widget StackView để mô tả cách thêm hành vi cho từng mục đơn lẻ. Trong mẫu Widget StackView, nếu người dùng chạm vào view trên cùng, Widget của ứng dụng sẽ hiển thị thông điệp `Toast` như sau "Touched view *n*", trong đó *n* là chỉ số (vị trí) của view được chạm đó. Đây là cách nó làm việc:

- `StackWidgetProvider` (một lớp con `AppWidgetProvider`) tạo một Intent ở trạng thái chờ có một action tùy chỉnh gọi là `TOAST_ACTION`.
- Khi người dùng chạm vào một view, Intent sẽ được truyền đi và sẽ truyền `TOAST_ACTION`.
- Broadcast này bị `StackWidgetProvider` của phương thức `onReceive()` chặn lại, còn Widget trong ứng dụng hiển thị thông điệp Toast cho view đó. Dữ liệu cho các mục của collection được `RemoteViewsFactory` cung cấp, thông qua `RemoteViewsService`.

> **Ghi chú:** ứng dụng mẫu Widget StackView sử dụng một broadcast. Tuy nhiên, thường thì một Widget của ứng dụng sẽ kích hoạt một activity trong tình huống như vậy.

Thiết lập mẫu Intent ở trạng thái chờ

`StackWidgetProvider` (lớp con `AppWidgetProvider`) thiết lập một Intent ở trạng thái chờ. Các mục riêng lẻ của một collection không thể tự thiết lập các Intent ở trạng thái chờ của chúng. Thay vào đó, collection đóng vai trò thiết lập một mẫu Intent ở trạng thái chờ, còn các mục riêng lẻ thiết lập một Intent để tạo hành vi duy nhất trên cơ sở từng mục một (item-by-item).

Lớp này cũng nhận broadcast được gửi khi người dùng chạm vào một view. Nó xử lý sự kiện này trong phương thức `onReceive()`. Nếu hành động của Intent là `TOAST_ACTION`, Widget trong ứng dụng hiển thị một thông điệp `Toast` cho view hiện tại.

```
public class StackWidgetProvider extends AppWidgetProvider {
    public static final String TOAST_ACTION =
                    "com.example.android.stackwidget.TOAST_ACTION";
    public static final String EXTRA_ITEM =
                    "com.example.android.stackwidget.EXTRA_ITEM";

    ...

    // Được gọi khi BroadcastReceiver nhận một Intent broadcast.
    // Kiểm tra xem action của Intent có phải là TOAST_ACTION
    // hay không. Nếu đúng, Widget trong ứng dụng hiển thị một thông
    // điệp Toast cho mục hiện tại.
    @Override
    public void onReceive(Context context, Intent intent) {
        AppWidgetManager mgr = AppWidgetManager.getInstance(context);
        if (intent.getAction().equals(TOAST_ACTION)) {
```

```
            int appWidgetId =
              intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                 AppWidgetManager.INVALID_APPWIDGET_ID);
            int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
            Toast.makeText(context, "Touched view " + viewIndex,
                                        Toast.LENGTH_SHORT).show();
        }
        super.onReceive(context, intent);
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
        // cập nhật từng Widget của ứng dụng với adapter từ xa
        for (int i = 0; i < appWidgetIds.length; ++i) {

            // Thiết lập Intent trỏ tới StackViewService cung cấp
            // các view cho collection này.
            Intent intent = new Intent(context, StackWidgetService.class);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                                    appWidgetIds[i]);
            // Khi các Intent được so sánh, phần phụ bị bỏ qua, nên
            // chúng ta cần nhúng phần phụ vào dữ liệu để chúng không
            // bị bỏ qua nữa.
            intent.setData(Uri.parse(intent.toUri
                                        (Intent.URI_INTENT_SCHEME)));
            RemoteViews rv = new RemoteViews(context.getPackageName(),
                                            R.layout.widget_layout);
            rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view, intent);

            // View trống được hiển thị khi collection không có mục nào
            // Nó nên là anh em của collection view.
            rv.setEmptyView(R.id.stack_view, R.id.empty_view);
            // Phần này cho phép các mục có hành vi độc lập. Nó làm việc
            // này bằng cách thiết lập một mẫu Intent ở trạng thái chờ.
            // Các mục độc lập của một collection không thể tự thiết
            // lập các Intent ở trạng thái chờ của chúng. Thay vào đó,
            // collection sẽ thiết lập một mẫu Intent ở trạng thái chờ,
            // và các mẫu độc lập sẽ thiết lập một fillInIntent để tạo
            // hành vi duy nhất trên cơ sở từng mục một.
            Intent toastIntent = new Intent(context, StackWidgetProvider.class);
            // Thiết lập action cho Intent. Khi người dùng chạm vào
            // một view cụ thể, nó sẽ có hiệu ứng của
            // TOAST_ACTION broadcast.
            toastIntent.setAction(StackWidgetProvider.TOAST_ACTION);
```

```
            toastIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                                    appWidgetIds[i]);
            intent.setData(Uri.parse(intent.toUri
                                        (Intent.URI_INTENT_SCHEME)));
            PendingIntent toastPendingIntent =
                   PendingIntent.getBroadcast(context, 0, toastIntent,
                                    PendingIntent.FLAG_UPDATE_CURRENT);
            rv.setPendingIntentTemplate
                                (R.id.stack_view, toastPendingIntent);

            appWidgetManager.updatesAppWidget(appWidgetIds[i], rv);
        }
    super.onUpdate(context, appWidgetManager, appWidgetIds);
    }
}
```

Thiết lập các Intent thay thế

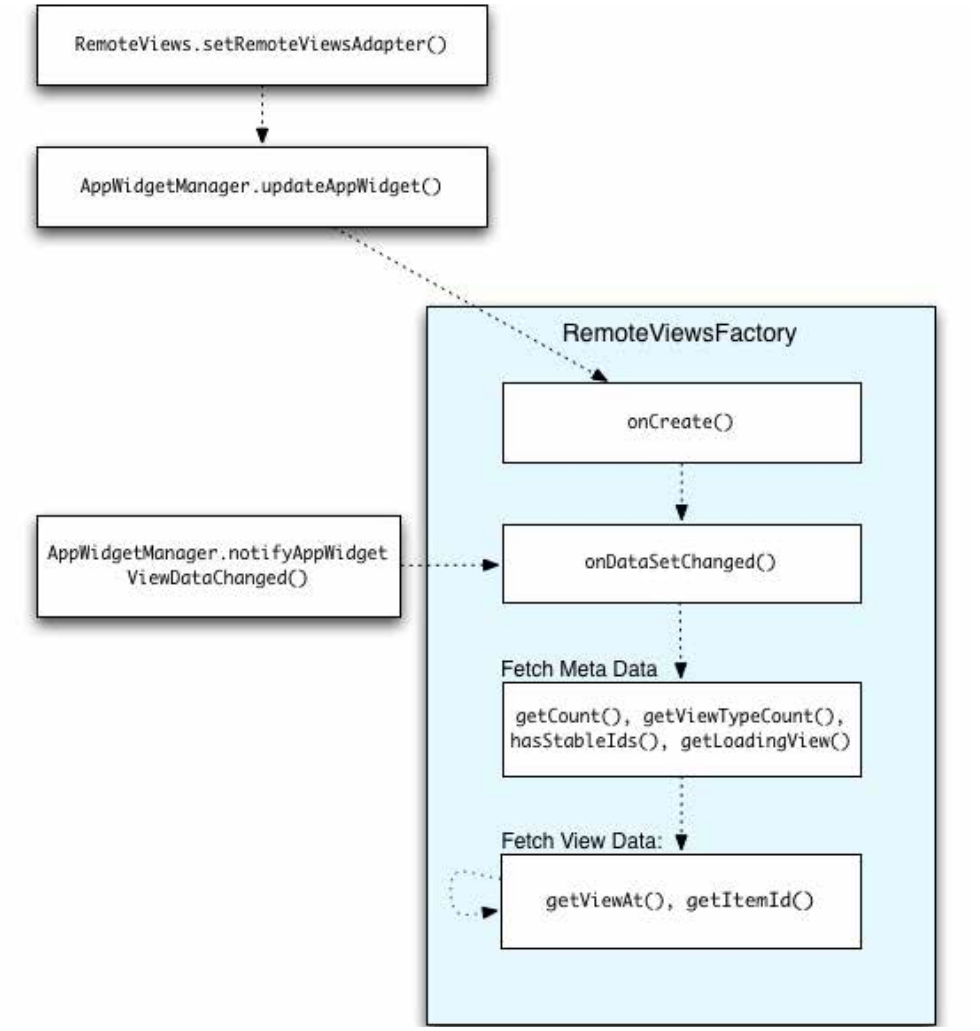
RemoteViewsFactory phải thiết lập một Intent thay thế (fill-in Intent) trên mỗi mục trong collection. Điều này giúp chúng ta có thể phân biệt action on-click riêng lẻ của một mục được chọn. Sau đó, Intent thay thế được kết hợp với template (mẫu) `PendingIntent` để xác định Intent cuối cùng được thực thi khi mục đó được nhấn.

```
public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new StackRemoteViewsFactory(this.getApplicationContext(),
                                                                intent);
    }
}

class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItems = new ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
        mAppWidgetId =
intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                    AppWidgetManager.INVALID_APPWIDGET_ID);
    }

    // Khởi tạo tập dữ liệu.
        public void onCreate() {
```

```
            // Trong onCreate(), bạn thiết lập bất kỳ kết nối con
            // trỏ nào cho nguồn dữ liệu của mình. Hoạt động nặng ,
            // ví dụ như tải về hoặc tạo nội dung, nên được truyền
            // tới onDataSetChanged() hoặc getViewAt(). Nếu chạy quá
            // 20 giây, lời gọi này sẽ trả về trong một ANR.
            for (int i = 0; i < mCount; i++) {
                mWidgetItems.add(new WidgetItem(i + "!"));
            }
            ...
        }
        ...
        // Vị trí (chỉ số) của một WidgetItem trong mảng, sử dụng giá
        // trị văn bản của mục trong phần kết hợp với mục Widget của ứng
        // dụng trong  file XML để khởi tạo một đối tượng RemoteViews.
        public RemoteViews getViewAt(int position) {
            // vị trí luôn chạy từ 0 đến getCount() - 1.
            // Khởi tạo một mục RemoteViews dựa trên mục Widget của ứng dụng
            // trong file XML, và thiết lập văn bản dựa trên vị trí.
            RemoteViews rv = new RemoteViews
                    (mContext.getPackageName(), R.layout.widget_item);
            rv.setTextViewText(R.id.widget_item,
                               mWidgetItems.get(position).text);
            // Tiếp theo, thiết lập một fill-intent sẽ được sử dụng
            // trong template Intent ở trạng thái chờ
            // được thiết lập trên collection view trong StackWidgetProvider.
            Bundle extras = new Bundle();
            extras.putInt(StackWidgetProvider.EXTRA_ITEM, position);
            Intent fillInIntent = new Intent();
            fillInIntent.putExtras(extras);
            // Cho phép phân biệt action on-click đơn của một
            // mục đã cho
            rv.setOnClickFillInIntent(R.id.widget_item, fillInIntent);
            ...
            // Trả về đối tượng RemoteViews.
            return rv;
        }
    ...
    }
```

Giữ cho dữ liệu của collection luôn cập nhật

Minh họa bên dưới cho thấy luồng (flow) xảy ra trong một Widget của ứng dụng dùng các collection khi cập nhật xảy ra. Nó chỉ ra cách thức mã nguồn của Widget ứng dụng tương tác với `RemoteViewsFactory` và cách bạn có thể kích hoạt các cập nhật:



Một tính năng của những Widget ứng dụng dùng các collection là cung cấp cho người dùng nội dung mới nhất. Ví dụ, hãy xem xét Widget ứng dụng của Gmail trong Android 3.0, cung cấp cho người dùng một ảnh chụp nhanh của hộp thư đến. Để làm điều này, bạn cần kích hoạt `RemoteViewsFactory` và collection view để lấy về cũng như hiển thị dữ liệu mới. Bạn thực hiện điều này với `AppWidgetManager` bằng cách gọi `notifyAppWidgetViewDataChanged()`. Lời gọi này trả về kết quả trong một callback tới phương thức `onDataSetChanged()` của `RemoteViewsFactory`, cho phép bạn lấy bất kỳ dữ liệu mới nào. Lưu ý, bạn có thể thực hiện các hoạt động xử lý phức tạp (processing-intensive operation) một cách đồng bộ trong callback `onDataSetChanged()`. Bạn được bảo đảm rằng lời gọi này sẽ hoàn tất trước

khi siêu dữ liệu hoặc dữ liệu của view được lấy từ `RemoteViewsFactory`. Ngoài ra, bạn có thể thực hiện những hoạt động được xử lý đặc biệt trong phương thức `getViewAt()`. Nếu lời gọi này tốn nhiều thời gian, view đang nạp (xác định bởi phương thức `getLoadingView()` của `RemoteViewsFactory`) sẽ được hiển thị tại vị trí tương ứng của collection view cho tới khi nó trả về.

8. Activity

`Activity` là thành phần của ứng dụng cung cấp một màn hình cho người dùng có thể tương tác cùng để làm việc gì đó, chẳng hạn như gọi điện thoại, chụp ảnh, gửi e-mail hoặc xem bản đồ. Mỗi activity được cấp một cửa sổ để vẽ giao diện người dùng trên đó. Cửa sổ thường chiếm toàn bộ màn hình, song cũng có thể nhỏ hơn màn hình và nổi (float) lên phía trên các cửa sổ khác.

Một ứng dụng thường chứa nhiều activity gắn với nhau khá lỏng lẻo. Thông thường, một activity trong ứng dụng được xác định là activity "chính" ("main" activity), được hiển thị cho người dùng khi khởi động ứng dụng lần đầu. Sau đó, mỗi activity có thể khởi động một activity khác để thực hiện những action khác. Mỗi khi một activity mới khởi động, activity trước đó sẽ dừng lại, nhưng hệ thống vẫn bảo lưu activity này trong một ngăn xếp - đó là "ngăn xếp lùi" ("back stack"). Khi một activity mới khởi động, nó được đẩy vào ngăn xếp lùi và khiến người dùng tập trung vào đó. Ngăn xếp lùi tuân thủ cơ chế "last in, first out" (còn gọi là "LIFO", tức "vào sau, ra trước") cơ bản của ngăn xếp. Do vậy, khi người dùng làm việc xong với activity hiện tại và nhấn button *Back*, activity này sẽ được lấy khỏi ngăn xếp (và bị hủy), còn activity trước đó sẽ phục hồi. (Ngăn xếp lùi được thảo luận kỹ hơn trong tài liệu Tác vụ và ngăn xếp lùi (Tasks and Back Stack).

Khi một activity bị dừng do một activity mới khởi động, nó được thông báo về sự thay đổi trạng thái này thông qua các phương thức callback trong vòng đời của activity. Có vài phương thức callback mà một activity có thể nhận, đó là kết quả từ sự thay đổi trạng thái của activity này - hệ thống tạo, dừng, phục hồi hay phá hủy nó - và mỗi callback đem đến cho bạn cơ hội thực hiện công việc cụ thể thích hợp với sự thay đổi trạng thái đó. Ví dụ, khi dừng, activity của bạn nên giải phóng bất kỳ đối tượng lớn nào, chẳng hạn như mạng hoặc các kết nối cơ sở dữ liệu. Khi activity phục hồi, bạn có thể lấy lại những tài nguyên cần thiết và phục hồi các action bị ngắt trước đó. Những chuyển dịch trạng thái này là tất cả các phần của vòng đời (lifecycle) activity.

Phần còn lại của tài liệu này thảo luận về cách xây dựng và sử dụng một activity ở mức cơ bản, trong đó bao gồm một thảo luận hoàn chỉnh về cách làm việc của vòng đời activity. Do đó, bạn có thể quản lý sát sao việc chuyển đổi giữa nhiều trạng thái khác nhau của activity.

Tạo activity

Để tạo một activity, bạn phải tạo một lớp con mới (hoặc một lớp con đã tồn tại) của `Activity`. Trong lớp con của mình, bạn cần thực thi những phương thức callback do hệ thống gọi trong khi chuyển đổi giữa các trạng thái của vòng đời activity đó, chẳng hạn như khi activity được tạo, bị dừng, phục hồi hoặc bị phá hủy. Hai phương thức callback quan trọng nhất là:

`onCreate()`

Bạn phải thực thi phương thức này. Hệ thống gọi nó khi tạo activity. Trong lúc thực thi, bạn nên khởi tạo những thành phần cần thiết của activity. Quan trọng nhất, đây chính là nơi bạn phải gọi `setContentView()` để xác định layout cho giao diện người dùng của activity.

`onPause()`

Hệ thống gọi phương thức này dưới dạng dấu hiệu đầu tiên cho thấy người dùng đang rời khỏi activity của bạn (dù không phải lúc nào nó cũng truyền tải ý nghĩa là hoạt động bị hủy). Đây thường là nơi bạn nên xác nhận (commit) bất cứ thay đổi nào lẽ ra nên được lưu trữ lâu dài ngoài phiên hiện tại của người dùng (bởi người dùng có thể sẽ không trở lại).

Bạn nên sử dụng vài phương thức callback khác để cung cấp một trải nghiệm người dùng đa dạng giữa các activity, đồng thời xử lý các lần ngắt (interuption) bất thường khiến activity bị dừng lại, thậm chí bị hủy bỏ. Tất cả các phương thức callback trên toàn vòng đời của activity đều được thảo luận ở mục “Quản lý vòng đời của activity” bên dưới.

Thực thi một giao diện người dùng

Giao diện người dùng cho một activity được cung cấp bởi một cây phân cấp các view - đối tượng được kế thừa từ lớp `View`. Mỗi view điều khiển một không gian hình chữ nhật cụ thể trong cửa sổ của activity và có thể phản hồi tương tác từ người dùng. Ví dụ, một view có thể là button khởi tạo action khi người dùng chạm vào nó.

Android cung cấp một số view được làm sẵn mà bạn có thể dùng để thiết kế và tổ chức layout của mình. “Widget” là các view cung cấp một bộ phần tử trực quan (và có tính tương tác) cho màn hình, chẳng hạn như một button, trường văn bản, checkbox, hay đơn giản chỉ là một bức ảnh. Các “layout” là những view lấy từ `ViewGroup` đóng vai trò cung cấp một mô hình layout duy nhất cho các view con của nó, chẳng hạn như layout tuyến tính (linear layout), layout dạng lưới (grid layout), hoặc layout tương đối (relative layout). Bạn cũng có thể dùng các lớp con của `View` và các lớp `ViewGroup` (hoặc những lớp con có sẵn) để tự tạo widget và layout cho mình, sau đó áp dụng chúng cho layout activity của bạn.

Cách thông dụng nhất để xác định một layout sử dụng các view là với một file layout XML được lưu trong tài nguyên ứng dụng của bạn. Bằng cách này, bạn có thể duy trì thiết kế của giao diện người dùng tách khỏi mã nguồn xác định hành vi của activity. Bạn có thể thiết lập layout với vai trò giao diện người dùng cho activity của mình với `setContentView()`, sau đó truyền ID tài nguyên cho layout. Tuy nhiên, bạn cũng có thể tạo những `View` mới trong mã nguồn của activity và xây dựng một cây phân cấp view bằng cách chèn các `View` mới vào một `ViewGroup`, sau đó sử dụng layout đó bằng cách truyền `ViewGroup` gốc cho `setContentView()`.

Để biết thêm thông tin về việc tạo một giao diện người dùng, tham khảo tài liệu Giao

diện người dùng.

Khai báo activity trong file kê khai

Bạn phải khai báo activity của mình trong file kê khai để hệ thống có thể truy cập nó. Để khai báo activity, hãy mở file kê khai và thêm phần tử <activity> dưới dạng con của phần tử <application>. Ví dụ:

```
<manifest ... >
    <application ... >
        <activity android:name=".ExampleActivity" />
        ...
    </application ... >
    ...
</manifest >
```

Có vài thuộc tính khác mà bạn có thể bao gồm trong phần tử này, nhằm xác định các thuộc tính hạn như: Nhãn (label) cho activity, biểu tượng (icon) cho activity, hoặc một theme để thay đổi style cho giao diện người dùng của activity. Trong đó, android:name là thuộc tính duy nhất cần thiết - đây là thuộc tính xác định tên lớp của activity. Khi phát hành ứng dụng, bạn không nên đổi tên này bởi nếu làm vậy, bạn có thể phá hỏng một số chức năng, chẳng hạn như các shortcut của ứng dụng (đọc bài blog Things That Cannot Change (Một số thứ không thể thay đổi)).

Xem phần tham khảo về phần tử <activity> để biết thêm thông tin về cách khai báo activity trong file kê khai.

Sử dụng bộ lọc intent

Một phần tử <activity> cũng có thể xác định nhiều bộ lọc intent - bằng cách dùng phần tử <intent-filter> - để khai báo cách mà các thành phần của ứng dụng khác có thể kích hoạt nó.

Khi bạn tạo một ứng dụng mới sử dụng các công cụ SDK của Android, activity gốc được tạo tự động bao gồm một bộ lọc intent khai báo activity tương ứng với action "chính" và nên đặt hạng mục là "launcher" ("màn hình chính"). Bộ lọc intent trông như sau:

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Phần tử <action> xác định đây là điểm bắt đầu "main" cho ứng dụng. Phần tử <category> cho thấy rằng activity này nên được liệt kê trong bộ khởi động ứng dụng

của hệ thống (để cho phép người dùng kích hoạt activity này).

Nếu dự định cho ứng dụng của mình có thể tự kiểm soát và không cho phép các ứng dụng khác kích hoạt activity của nó, bạn không cần bất kỳ bộ lọc intent nào khác. Chỉ duy nhất một activity nên có action "main" và hạng mục "launcher", như ở ví dụ trước. Những activity bạn không muốn có mặt trong các ứng dụng khác thì không nên có bộ lọc intent và bạn có thể tự khởi động chúng bằng cách dùng các intent tường minh (explicit intent), như thảo luận ở mục dưới đây.

Tuy nhiên, nếu muốn activity của mình phản hồi các intent ngầm định (implicit intent) được gửi từ những ứng dụng khác (và cả ứng dụng của bạn), bạn phải xác định thêm các bộ lọc intent cho activity của mình. Đối với mỗi loại intent bạn muốn phản hồi tới, bạn phải bao gồm một `<intent-filter>` chứa một phần tử `<action>`, đồng thời có thể tùy chọn thêm một phần tử `<category>` và/hoặc `<data>`. Những phần tử này xác định kiểu intent mà activity của bạn có thể phản hồi tới.

Để biết thêm thông tin về cách activity của bạn phản hồi lại intent, tham khảo tài liệu Intent và bộ lọc Intent tại địa chỉ:
http://developer.android.com/guide/components/intents-filters.html.

Khởi động activity

Bạn có thể khởi động một activity khác bằng cách gọi `startActivity()`, truyền cho nó một `Intent` mô tả activity mà bạn muốn khởi động. Intent hoặc xác định chính xác activity bạn muốn khởi động, hoặc mô tả kiểu action bạn muốn thực hiện (và hệ thống sẽ chọn activity phù hợp cho bạn, activity đó có thể tới từ ứng dụng khác). Mỗi intent cũng có thể chứa một lượng nhỏ dữ liệu được activity sử dụng khi nó khởi động.

Khi làm việc trong ứng dụng của mình, bạn sẽ phải thường xuyên kích hoạt một activity đã biết. Bạn có thể làm vậy bằng cách tạo một intent xác định rõ ràng activity mình muốn khởi động qua việc sử dụng tên lớp. Ví dụ, đây là cách một activity khởi động activity khác có tên `SignInActivity`:

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

Tuy nhiên, ứng dụng của bạn có thể muốn thực hiện một số action, chẳng hạn như gửi e-mail, tin nhắn văn bản, hoặc cập nhật trạng thái, bằng cách sử dụng dữ liệu từ activity của bạn. Trong trường hợp này, ứng dụng có thể không có các activity của mình để thực hiện những action ấy. Do vậy, bạn có thể dùng các activity do những ứng dụng khác cung cấp trên thiết bị, chúng có thể thực hiện action cho bạn. Đây là nơi các intent thật sự có giá trị - bạn có thể tạo một intent mô tả action mình muốn thực hiện và hệ thống sẽ khởi động activity phù hợp từ ứng dụng khác. Nếu có nhiều activity có thể xử lý intent đó, người dùng có thể chọn một activity nào đó để sử dụng. Ví dụ, nếu muốn cho phép người dùng gửi một e-mail, bạn có thể tạo intent sau:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Phần phụ EXTRA_EMAIL được thêm vào intent là một mảng các chuỗi địa chỉ e-mail mà mail này sẽ được gửi tới. Khi một ứng dụng e-mail phản hồi intent này, nó đọc mảng chuỗi cung cấp trong phần phụ trên và đặt chúng trong trường "to" ("gửi đến") của form soạn thảo e-mail. Trong trường hợp này, activity của ứng dụng e-mail khởi động và khi người dùng hoàn tất, activity sẽ phục hồi.

Khởi động activity để trả về kết quả

Đôi khi, bạn muốn nhận một kết quả từ activity mà mình khởi động. Trong trường hợp đó, hãy khởi động activity bằng cách gọi startActivityForResult() (thay vì startActivity()). Sau đó, để nhận kết quả từ activity tiếp theo, hãy thực thi phương thức callback onActivityResult(). Khi activity tiếp theo hoàn tất, nó trả về kết quả trong một Intent tới phương thức onActivityResult().

Ví dụ, có thể bạn muốn người dùng chọn một trong các thông tin liên lạc của mình; từ đó, activity của bạn có thể làm việc gì đó với thông tin ấy. Đây là cách bạn có thể tạo intent như vậy và xử lý kết quả:

```
private void pickContact() {
    // Tạo một intent để "chọn" thông tin liên lạc,
    // như đã định nghĩa bởi content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    // Nếu yêu cầu thành công (OK) và là PICK_CONTACT_REQUEST
if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST)
{
        // Thực hiện một truy vấn tới content provider
        // của danh bạ cho tên danh bạ
        Cursor cursor = getContentResolver().query(data.getData(),
        new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True nếu con trỏ không trống
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Làm gì đó với tên danh bạ đã chọn...
        }
    }
}
```

Ví dụ này cho thấy lôgíc cơ bản mà bạn nên sử dụng trong phương thức onActivityResult() để xử lý kết quả của activity. Điều kiện đầu tiên kiểm tra xem yêu cầu có thành công hay không - nếu có, resultCode sẽ có giá trị RESULT_OK - và yêu cầu tới kết quả này có đang phản hồi hay không - trong trường hợp này,

`requestCode` khớp với tham số thứ hai gửi bởi `startActivityForResult()`. Từ đây, mã nguồn xử lý kết quả activity bằng cách truy vấn dữ liệu trả về trong một `Intent` (tham số `data`).

Điều xảy ra là, một `ContentResolver` thực hiện một truy vấn tới một content provider, provider này trả về một `Cursor` cho phép dữ liệu đã truy vấn có thể được đọc. Để biết thêm thông tin, tham khảo tài liệu Content providers.
Để tham khảo thông tin về cách sử dụng intent, xem tài liệu "Intents and Intent Filters" ("Intent và bộ lọc intent".

Tắt activity

Bạn có thể tắt một activity bằng cách gọi phương thức `finish()` của activity này. Bạn cũng có thể tắt một activity riêng đã khởi động trước đó bằng cách gọi `finishActivity()`.

> **Ghi chú:** Trong hầu hết trường hợp, bạn không nên tắt một activity một cách tường minh bằng cách sử dụng các phương thức trên. Như nội dung thảo luận ở mục sau về vòng đời của activity, hệ thống Android quản lý vòng đời của một activity cho bạn, do đó bạn không cần tắt các activity. Việc gọi các phương thức trên có thể ảnh hưởng không tốt đến trải nghiệm người dùng dự kiến và chỉ nên được sử dụng khi bạn hoàn toàn không muốn người dùng quay về thể hiện này của activity.

Quản lý vòng đời của activity

Quản lý vòng đời của activity bằng cách thực thi phương thức callback là vấn đề quan trọng trong việc phát triển một ứng dụng mạnh mẽ, linh hoạt. Vòng đời của một activity bị ảnh hưởng trực tiếp bởi mối liên hệ giữa nó với những activity khác, với tác vụ và ngăn xếp lùi của nó.

Một activity có thể tồn tại ở ba trạng thái chính:

Resumed (Được khôi phục lại)

Activity đang ở màn hình chính và người dùng đang focus vào đó. (Trạng thái này đôi khi còn được gọi là "running" ("đang chạy")).

Paused (Bị tạm dừng)

Một activity khác trong tiền cảnh màn hình và được focus, song activity này vẫn đang hiển thị. Nghĩa là, một activity khác đang hiển thị ở trên cùng của activity này và activity đó trong suốt một phần hoặc không che toàn bộ màn hình. Một activity bị tạm dừng là hoàn toàn đang tồn tại (đối tượng `Activity` đang nằm trong bộ nhớ, duy trì tất cả trạng thái cũng như thông tin thành viên, đồng thời vẫn được gắn với trình quản lý cửa sổ (window manager)), nhưng lại có thể bị hệ thống hủy trong các trường hợp bộ nhớ ở mức rất thấp.

Stopped (Bị dừng)

Activity bị activity khác che hoàn toàn (hiện activity này đang ở "hậu cảnh" ("background"). Một activity bị dừng cũng đang tồn tại (đối tượng `Activity` được giữ trong bộ nhớ, duy trì tất cả trạng thái và thông tin thành viên, nhưng *không*

được gắn với trình quản lý cửa sổ). Tuy nhiên, activity bị dừng không còn hiển thị đối với người dùng và có thể bị hệ thống hủy khi bộ nhớ cần cho việc khác.

Nếu một activity bị tạm dừng hoặc dừng, hệ thống có thể loại bỏ activity này khỏi bộ nhớ bằng cách yêu cầu activity kết thúc (gọi phương thức `finish()` của activity), hoặc đơn giản là hủy tiến trình của nó. Khi activity được mở lại (sau khi bị kết thúc hoặc hủy), nó phải được tạo từ đầu.

Thực thi các callback trong vòng đời của activity

Khi một activity chuyển đổi thành hoặc thoát khỏi các trạng thái khác nhau như mô tả ở trên, activity sẽ được thông báo qua nhiều phương thức callback khác nhau. Bạn có thể ghi đè tất cả các phương thức callback để làm việc phù hợp khi trạng thái của activity thay đổi. Activity khung sau bao gồm những phương thức cơ bản trong vòng đời activity:

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Activity đang được tạo.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // Activity chuẩn bị hiển thị.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // Activity đã ẩn (giờ ”được phục hồi”).
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Activity khác đang được focus (activity sắp “bị tạm dừng”).
    }
    @Override
    protected void onStop() {
        super.onStop();
        // Activity không còn hiển thị (hiện nó đang “bị dừng”)
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Activity sắp bị hủy bỏ.
```

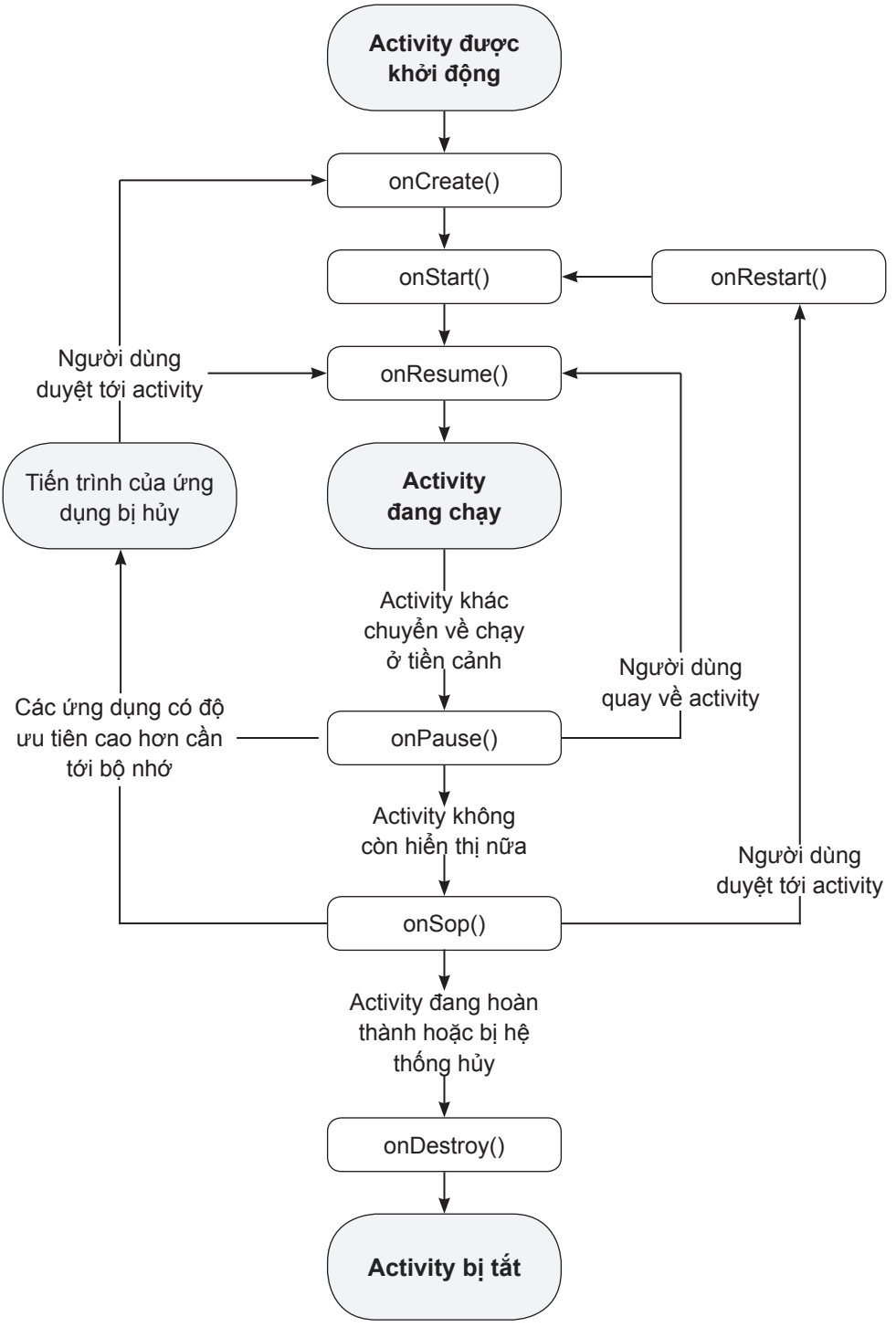
```
    }
}
```

Ghi chú: Thực thi của các phương thức vòng đời này phải luôn gọi phương thức của lớp cha trước khi làm bất cứ việc gì, như đã nêu trong ví dụ trên.

Các phương thức đó cùng nhau xác định toàn bộ vòng đời của một activity. Bằng việc thực thi những phương thức này, bạn có thể theo dõi ba vòng lặp lồng nhau trong vòng đời activity:

- **Toàn bộ vòng đời (entire lifetime)** của một activity được tính giữa lời gọi tới phương thức `onCreate()` và lời gọi tới phương thức `onDestroy()`. Activity của bạn nên thực hiện cài đặt trạng thái "global" (chẳng hạn như định nghĩa layout) trong `onCreate()`, đồng thời giải phóng tất cả tài nguyên còn lại trong `onDestroy()`. Ví dụ, nếu activity có một luồng chạy ngầm để tải về dữ liệu từ mạng, nó có thể tạo luồng đó trong `onCreate()`, sau đó dừng luồng này trong `onDestroy()`.
- **Vòng đời thấy được (visible lifetime)** của một activity được tính giữa lời gọi tới `onStart()` và lời gọi tới `onStop()`. Trong suốt thời gian này, người dùng có thể thấy activity trên màn hình và tương tác với nó. Ví dụ, `onStop()` được gọi để làm cho một activity mới khởi động không còn hiển thị nữa. Giữa hai phương thức này, bạn có thể duy trì tài nguyên cần thiết để hiển thị activity cho người dùng. Ví dụ, bạn có thể đăng ký một `BroadcastReceiver` trong `onStart()` để theo dõi các thay đổi ảnh hưởng tới giao diện người dùng, đồng thời hủy đăng ký nó trong `onStop()` khi người dùng không còn thấy cái mà bạn đang hiển thị. Hệ thống có thể gọi `onStart()` và `onStop()` nhiều lần trong toàn bộ vòng đời của activity, dẫn tới việc activity hiển thị và ẩn đối với người dùng.
- **Vòng đời nền trước (foreground lifetime)** của một activity được tính giữa lời gọi tới `onResume()` và lời gọi tới `onPause()`. Trong suốt thời gian này, activity ở phía trước của tất cả các activity khác trên màn hình và được focus. Một activity có thể thường xuyên chuyển tiếp vào và ra khỏi tiền cảnh - ví dụ, `onPause()` được gọi khi thiết bị đi vào chế độ ngủ hoặc khi hộp thoại xuất hiện. Vì trạng thái này có thể chuyển đổi thường xuyên, mã nguồn trong hai phương thức đó nên gọn nhẹ nhằm tránh trình trạng chậm trễ khi chuyển đổi làm người dùng phải đợi.

Hình 1 minh họa các vòng lặp này và những đường chuyển trạng thái khác nhau mà một activity có thể có. Hình chữ nhật đại diện cho phương thức callback mà bạn có thể thực thi để thực hiện hoạt động khi chuyển tiếp activity giữa các trạng thái.



Hình 1. Vòng đời activity.

Các phương thức callback của vòng đời activity giống nhau được liệt kê ở Bảng 1, mô

tả chi tiết hơn về từng phương thức callback và xác định vị trí của chúng trong toàn bộ vòng đời của activity, bao gồm hệ thống có thể hủy activity hay không sau khi phương thức callback hoàn tất.

Bảng 1. Tổng quan về các phương thức callback của vòng đời activity.

Phương thức	Mô tả	Có thể bị hủy sau đó không?	Phương thức được gọi tiếp theo
`onCreate()`	Được gọi khi activity được tạo lần đầu. Đây là nơi bạn nên thực hiện tất cả các cài đặt tĩnh thông thường - tạo view, gán dữ liệu cho danh sách,… Phương thức này được truyền một đối tượng Bundle chứa trạng thái trước của activity, nếu trạng thái đó đã được lưu giữ (xem mục "Lưu trạng thái của Activity" ở phần sau). Luôn được `onStart()` theo sau.	**Không**	`onStart()`
`onRestart()`	Được gọi sau khi activity bị dừng, ngay trước khi được khởi động lại. Luôn được `onStart()` theo sau.	**Không**	`onStart()`
`onStart()`	Được gọi ngay trước khi activity hiển thị đối với người dùng. Được `onResume()` theo sau nếu activity chuyển đến tiền cảnh, hoặc được `onStop()` theo sau nếu nó bị ẩn.	**Không**	onResume() hoặc onStop()
`onResume()`	Được gọi ngay trước khi activity bắt đầu tương tác với người dùng. Tại thời điểm này, activity ở trên cùng của ngăn xếp activity, cùng với đầu vào từ người dùng tại đó. Luôn được `onPause()` theo sau.	**Không**	onPause()

Phương thức	Mô tả	Có thể bị hủy sau đó không?	Tiếp theo

onPause()	Được gọi khi hệ thống bắt đầu phục hồi activity khác. Phương thức này thường được dùng để commit những thay đổi chưa được lưu để lưu trữ dữ liệu lâu dài, dừng các hoạt cảnh và những thứ khác có thể chiếm tài nguyên CPU,... Phương thức này sẽ thực thi nhanh chóng, bởi activity kế tiếp sẽ không được phục hồi cho tới khi phương thức trả về.
	Phương thức `OnPause()` được theo sau bởi `onResume()` nếu activity trở lại màn hình chính, hoặc `onStop()` nếu activity không hiển thị với người dùng.
		Có	`onResume()`
			hoặc
			`onStop()`

onStop()	Được gọi khi activity không còn hiển thị với người dùng. Điều này có thể xảy ra bởi activity sắp bị hủy, hoặc do activity khác (hay một activity có sẵn hoặc activity mới) đã được phục hồi và đang che phủ nó.
	Được `onRestart()` theo sau nếu muốn activity trở lại để tương tác với người dùng, hoặc được `onDestroy()` theo sau nếu activity này sắp bị hủy.
		Có	`onRestart()`
			hoặc
			`onDestroy()`

onDestroy()	Được gọi trước khi activity bị hủy. Đây là lời gọi cuối cùng mà activity nhận được. Nó có thể được gọi, hoặc do activity đang chuẩn bị kết thúc (một số người gọi `finish()` cho nó), hoặc do hệ thống sẽ tạm thời hủy Activity này để tiết kiệm không gian. Bạn có thể phân biệt giữa hai tình huống này bằng phương thức `isFinishing()`.	**Có**	*không có gì*

Cột có nhãn "Có thể bị hủy sau đó không?" cho ta thấy hệ thống có thể hủy tiến trình quản lý activity vào bất cứ thời điểm nào *sau khi phương thức trả về*, mà không cần thực thi dòng mã nào khác của activity hay không. Ba phương thức được đánh dấu "Có" là (`onPause()`, `onStop()` và `onDestroy()`). Do `onPause()` đứng đầu tiên trong ba phương thức này, nên sau khi activity được tạo, `onPause()` sẽ là phương thức cuối cùng bảo đảm được gọi trước khi tiến trình *có thể* bị hủy - nếu hệ thống phải phục hồi bộ nhớ trong trường hợp khẩn cấp, `onStop()` và `onDestroy()` có thể không được gọi. Do đó, bạn nên sử dụng `onPause()` để ghi dữ liệu quan trọng cần lưu trữ lâu dài (chẳng hạn như các chỉnh sửa của người dùng) để thực hiện việc lưu. Tuy nhiên, bạn nên lựa chọn thông tin phải giữ lại trong `onPause()`, bởi bất cứ thủ tục chặn nào trong phương thức này cũng ngăn chặn việc chuyển tiếp tới activity tiếp theo và làm chậm trải nghiệm người dùng.

Các phương thức được đánh dấu "Không" trong cột “Có thể bị hủy sau đó không” bảo vệ tiến trình host activity khỏi bị hủy từ thời điểm chúng được gọi. Vì vậy, một activity có thể bị hủy từ lúc `onPause()` trả về cho tới lúc `onResume()` được gọi. Nó sẽ không thể bị hủy lần nữa cho tới khi `onPause()` được gọi và trả về lần nữa.

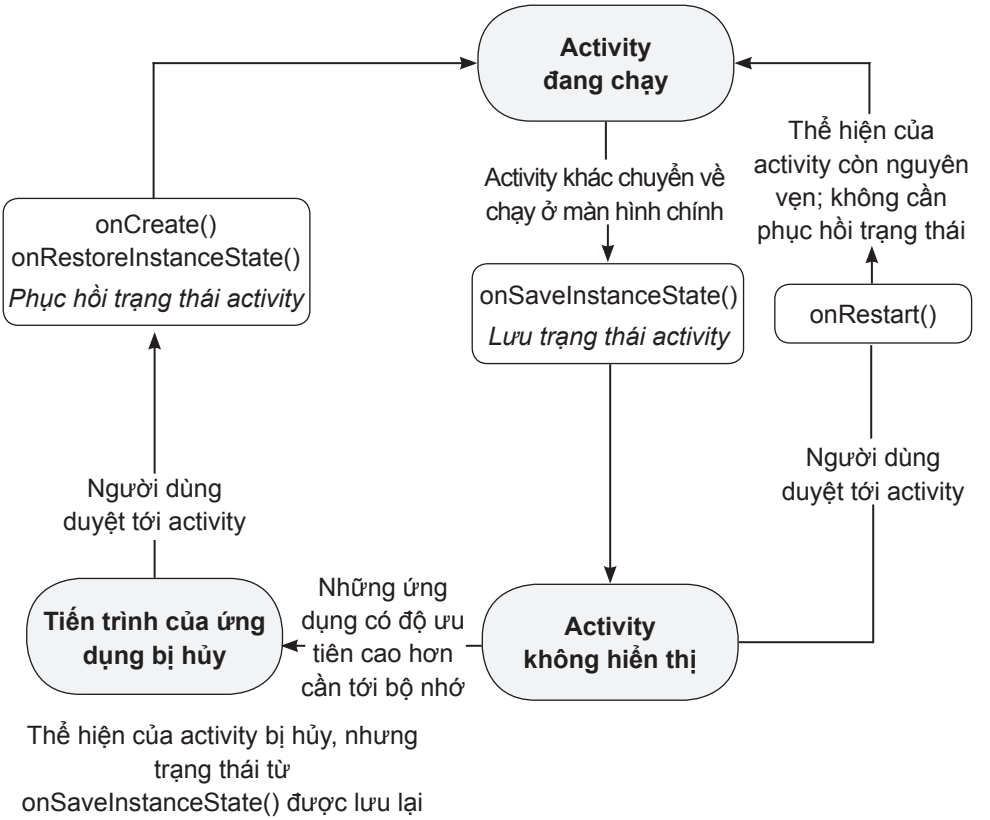
Ghi chú: Một activity không thể bị hủy theo định nghĩa ở Bảng 1 vẫn có thể bị hệ thống loại bỏ - tuy nhiên, điều này chỉ xảy ra trong các hoàn cảnh không còn tài nguyên nào khác. Thời điểm một activity có thể bị hủy được thảo luận kỹ hơn trong tài liệu “Processes and Threads” (“Tiến trình và luồng”).

Lưu trạng thái của activity

Mục “Quản lý vòng đời của Activity” đã đề cập ngắn gọn rằng khi một activity bị dừng hoặc tạm dừng, trạng thái của activity đó được bảo lưu. Điều này là đúng, bởi đối tượng `Activity` vẫn được giữ trong bộ nhớ khi nó bị dừng hoặc tạm dừng - mọi thông tin về các thành viên và trạng thái hiện tại của nó vẫn tồn tại. Vì vậy, bất cứ thay đổi nào do người dùng đã tạo ra trong activity đều được lưu giữ, nên khi activity quay về màn hình chính (khi nó “phục hồi”), những thay đổi này vẫn ở đó.

Tuy nhiên, khi hệ thống hủy một activity để phục hồi bộ nhớ, đối tượng `Activity` bị loại bỏ, nên hệ thống không thể chỉ đơn giản phục hồi nó về trạng thái nguyên vẹn. Thay vào đó, hệ thống phải tạo lại đối tượng Activity nếu người dùng điều hướng tới activity này. Tuy nhiên, người dùng không biết được rằng hệ thống đã hủy activity và tạo lại nó, nên họ lầm tưởng là activity trở lại như lúc trước. Trong trường hợp này, bạn có thể chắc chắn rằng thông tin quan trọng về trạng thái của activity được bảo vệ bằng việc thực thi thêm một phương thức callback có tên `onSaveInstanceState()`, cho phép bạn lưu thông tin về trạng thái của activity.

Hệ thống gọi `onSaveInstanceState()` trước khi làm cho activity này có thể bị hủy. Hệ thống truyền cho phương thức này một `Bundle`, trong đó bạn có thể lưu thông tin trạng thái về activity như các cặp tên-giá trị bằng cách sử dụng những phương thức như `putString()` và `putInt()`. Sau đó, nếu hệ thống hủy tiến trình của ứng dụng và người dùng duyệt trở lại activity của bạn, hệ thống sẽ tạo lại activity và truyền `Bundle` cho cả `onCreate()` lẫn `onRestoreInstanceState()`. Sử dụng các phương thức này, bạn có thể lấy trạng thái đã lưu từ `Bundle` và phục hồi trạng thái của activity. Nếu không có thông tin trạng thái để phục hồi, `Bundle` được truyền cho phương thức sẽ là null (đó là trường hợp khi activity được tạo lần đầu tiên).



> **Ghi chú:** Không có gì bảo đảm rằng `onSaveInstanceState()` sẽ được gọi trước khi activity bị hủy, bởi có nhiều trường hợp không cần thiết phải lưu trạng thái (chẳng hạn như khi người dùng rời khỏi activity bằng cách nhấn button *Back*, do người dùng muốn đóng activity một cách tường minh). Hệ thống sẽ gọi `onSaveInstanceState()` trước khi gọi `onStop()` và có thể cả `onPause()`.

Tuy nhiên, ngay cả khi bạn không làm gì và không thực thi `onSaveInstanceState()`, một số trạng thái của activity sẽ được phương thức `onSaveInstanceState()` mặc định của lớp `Activity` phục hồi. Cụ thể, phương thức mặc định sẽ gọi phương thức `onSaveInstanceState()` tương ứng đối với từng `View` trong layout, cho phép mỗi view cung cấp thông tin sẽ được lưu về bản thân view đó. Hầu hết mọi widget trong framework của Android đều thực thi phương thức này khi thích hợp, như là bất cứ thay đổi có thể thấy được nào tới giao diện người dùng đều được tự động lưu và phục hồi khi activity của bạn được tạo lại. Ví dụ, widget `EditText` lưu bất cứ văn bản nào do người dùng nhập vào và widget `CheckBox` lưu tình trạng được đánh dấu của nó. Điều duy nhất cần bạn làm là cung cấp một ID duy nhất (với thuộc tính `android:id`) cho mỗi widget mà bạn muốn lưu trạng thái. Nếu widget không có ID, hệ thống không thể lưu trạng thái của widget được.

Bạn cũng có thể ngăn không cho một view trong layout khỏi việc bị lưu trạng thái bằng cách thiết lập thuộc tính `android:saveEnabled` thành `"false"`, hoặc bằng cách

gọi phương thức `setSaveEnabled()`. Thông thường, bạn không nên vô hiệu hóa nó, song bạn có thể làm vậy nếu muốn phục hồi trạng thái của giao diện người dùng activity theo cách khác.

Mặc dù thực thi mặc định của `onSaveInstanceState()` lưu thông tin hữu ích về giao diện người dùng của activity, song bạn vẫn phải ghi đè nó để lưu thêm thông tin. Ví dụ, bạn có thể phải lưu các giá trị thành viên được thay đổi trong suốt vòng đời của activity (điều này có thể liên quan tới những giá trị được phục hồi trong giao diện người dùng; tuy nhiên, theo mặc định thì các thành viên nắm giữ giá trị đều không được phục hồi).

Do thực thi mặc định của `onSaveInstanceState()` giúp lưu trạng thái của giao diện người dùng, nên nếu muốn ghi đè phương thức này để lưu thêm thông tin thì bạn nên luôn gọi phương thức `onSaveInstanceState()` của lớp cha trước khi làm bất cứ việc gì. Tương tự, bạn cũng nên gọi phương thức `onRestoreInstanceState()` của lớp cha nếu bạn ghi đè phương thức này; vì vậy, thực thi mặc định có thể phục hồi các trạng thái view.

Ghi chú: Vì `onSaveInstanceState()` không bảo đảm được gọi, bạn chỉ nên dùng nó để ghi lại những sự chuyển đổi trạng thái của activity (trạng thái của giao diện người dùng) - đừng bao giờ dùng phương thức này để lưu trữ dữ liệu lâu dài. Thay vào đó, hãy sử dụng `onPause()` để làm việc này (chẳng hạn như dữ liệu nên được lưu vào cơ sở dữ liệu) khi người dùng rời khỏi activity.

Một cách hay để kiểm thử tính năng của ứng dụng trong việc phục hồi trạng thái của nó là bạn chỉ cần xoay thiết bị sao cho hướng của màn hình thay đổi. Khi hướng màn hình thay đổi, hệ thống sẽ hủy và tạo lại activity để áp dụng tài nguyên thay thế đã sẵn sàng có mặt cho cấu hình màn hình mới. Do vậy, điều quan trọng là activity của bạn phục hồi hoàn toàn trạng thái khi activity được tạo lại, vì người dùng thường xoay màn hình khi dùng các ứng dụng.

Xử lý các thay đổi cấu hình

Một số cấu hình thiết bị có thể thay đổi trong lúc chạy (chẳng hạn như chiều xoay màn hình, bàn phím có sẵn, ngôn ngữ). Khi thay đổi đó xảy ra, Android sẽ tạo lại activity đang chạy (hệ thống gọi `onDestroy()`, sau đó lập tức gọi `onCreate()`). Hành vi này được thiết kế nhằm giúp ứng dụng của bạn thích nghi với các cấu hình mới bằng cách tự động nạp lại ứng dụng với những tài nguyên thay thế mà bạn đã cung cấp (chẳng hạn như các layout khác nhau đối với những chiều xoay và kích thước màn hình khác nhau).

Nếu bạn thường thiết kế activity để xử lý việc khởi động lại do chiều xoay màn hình thay đổi và phục hồi trạng thái của activity như mô tả ở trên, ứng dụng của bạn sẽ vững hơn trước các sự kiện ngoài mong muốn trong suốt vòng đời của activity.

Cách tốt nhất để xử lý việc khởi động lại là lưu, sau đó phục hồi trạng thái của activity bằng cách sử dụng onSaveInstanceState() và onRestoreInstanceState() (hoặc onCreate()), như đã thảo luận ở mục trước.

Để biết thêm thông tin về cấu hình thay đổi xảy ra vào thời điểm chạy và cách xử lý chúng, hãy đọc hướng dẫn "Handling Runtime Changes" ("Xử lý các thay đổi khi chạy").

Phối hợp các activity

Khi một activity khởi động activity khác, chúng đều trải qua các chuyển tiếp vòng đời. Activity đầu tiên tạm dừng và dừng (dù nó sẽ không dừng nếu vẫn hiển thị trong hậu cảnh), trong khi activity khác được tạo. Trong trường hợp các activity chia sẻ dữ liệu đã lưu vào đĩa hoặc nơi nào đó, điều quan trọng là activity thứ nhất không hoàn toàn bị dừng khi activity thứ hai được tạo. Thay vào đó, tiến trình khởi động activity thứ hai sẽ xảy ra cùng lúc dừng activity thứ nhất.

Thứ tự của các phương thức callback trong vòng đời đã được xác định, đặc biệt là khi hai activity đều trong cùng tiến trình và một cái khởi động cái còn lại. Dưới đây là thứ tự của các hoạt động xảy ra khi Activity A khởi động Activity B:

Phương thức `onPause()` của Activity A được thực thi.

Các phương thức `onCreate()`, `onStart()` và `onResume()` của Activity B thực thi theo thứ tự. (Activity B giờ được người dùng focus vào).

Sau đó, nếu Activity A không còn hiển thị trên màn hình, phương thức `onStop()` của nó sẽ thực thi.

Việc tiên đoán trình tự của các callback cho phép bạn quản lý việc chuyển đổi thông tin từ activity này sang activity khác. Ví dụ, nếu phải ghi vào một cơ sở dữ liệu khi activity thứ nhất dừng lại để activity sau có thể đọc nó, bạn nên ghi vào cơ sở dữ liệu trong suốt thời điểm chạy `onPause()` thay vì `onStop()`.

9. Tùy chọn lưu trữ

Android cung cấp một số tùy chọn để bạn có thể lưu trữ dữ liệu lâu dài cho ứng dụng. Giải pháp bạn chọn phụ thuộc vào từng nhu cầu cụ thể, chẳng hạn như dữ liệu là của riêng ứng dụng này hay còn cho phép ứng dụng (và người dùng) khác truy cập vào, dung lượng cần dùng là bao nhiêu.

Bạn có thể chọn một trong những tùy chọn lưu trữ dữ liệu sau:

Tùy chỉnh chia sẻ (Shared Preferences)

: Lưu dữ liệu riêng kiểu cơ sở dưới dạng cặp khóa-giá trị.

Bộ nhớ trong (Internal Storage)

: Lưu dữ liệu riêng trên bộ nhớ thiết bị.

Bộ nhớ ngoài (External Storage)

: Lưu dữ liệu công cộng (public data) trên bộ nhớ ngoài để cùng chia sẻ.

Cơ sở dữ liệu SQLite (SQLite Databases)

: Lưu dữ liệu có cấu trúc (structured data) trong cơ sở dữ liệu riêng.

Kết nối mạng (Network Connection)

: Lưu dữ liệu trên Web bằng network server của bạn.

Android cung cấp cho bạn một cách để hiển thị dữ liệu riêng cho các ứng dụng khác - với một Content Provider. Content provider là một thành phần tùy chọn, dùng để cung cấp quyền đọc/ghi (read/write) đối với dữ liệu của ứng dụng, tùy thuộc vào giới hạn quyền bạn muốn cấp. Để tìm hiểu thêm về cách sử dụng content provider, hãy tham khảo tài liệu "Content Providers".

Sử dụng tùy chỉnh chia sẻ

Lớp `SharedPreferences` cung cấp một framework làm việc chung cho phép bạn lưu trữ và truy xuất các cặp khóa-giá trị cố định của các kiểu dữ liệu cơ sở. Bạn có thể sử dụng lớp `SharedPreferences` để lưu các dữ liệu cơ sở như: Boolean, float, int, long và string. Dữ liệu này sẽ tồn tại qua các phiên làm việc (session) của người dùng (thậm chí ngay cả khi ứng dụng đã kết thúc).

Tùy chỉnh cá nhân

Tùy chỉnh chia sẻ không chỉ dùng để lưu "tùy chỉnh cá nhân" ("user preference"), chẳng hạn như nhạc chuông (ringtone) mà người dùng lựa chọn. Nếu bạn muốn tạo phần tùy chỉnh cá nhân cho ứng dụng, hãy tìm hiểu thêm về `PreferenceActivity`, lớp này sẽ cung cấp một framework Activity để bạn tạo phần tùy chỉnh cá nhân, phần này sẽ tự động được lưu lại (bằng cách sử dụng tùy chỉnh chia sẻ).

Để tạo một đối tượng `SharedPreferences` trong ứng dụng, hãy dùng một trong hai phương thức sau:

- `getSharedPreferences()` - Sử dụng phương thức này nếu bạn cần nhiều file tùy chỉnh xác định bằng tên; bạn có thể xác định tên này thông qua tham số đầu tiên.
- `getPreferences()` - Sử dụng phương thức này nếu bạn chỉ cần một file tùy chỉnh cho Activity của mình. Do phương thức này chỉ tạo một file tùy chỉnh, nên bạn không cần cung cấp tên.

Để lưu các giá trị:

1. Gọi `edit()` để lấy một đối tượng `SharedPreferences.Editor`.
2. Thêm các giá trị với những phương thức như `putBoolean()` và `putString()`.
3. Gọi `commit()` để cập nhật các chỉnh sửa .

Để đọc giá trị, sử dụng phương thức `SharedPreferences`, chẳng hạn như `getBoolean()` và `getString()`.

Sau đây là một ví dụ về việc lưu tùy chọn cho chế độ nhấn phím im lặng trong ứng dụng máy tính cá nhân (calculator):

```
public class Calc extends Activity{
    public static final String PREFS_NAME =”MyPrefsFile”;

    @Override
    protectedvoid onCreate(Bundle state){
        super.onCreate(state);
         ...

        // Lưu tùy chỉnh
        SharedPreferences settings = getSharedPreferences(PREFS_NAME,0);
        boolean silent = settings.getBoolean(“silentMode”,false);
        setSilent(silent);
    }

    @Override
    protected void onStop(){
        super.onStop();

        // Chúng ta cần một đối tượng Editor để thay đổi tùy chỉnh.
        // Tất cả các đối tượng này đều nằm trong gói android.context.Context
        SharedPreferences settings = getSharedPreferences(PREFS_NAME,0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean(“silentMode”, mSilentMode);

        // Xác nhận các thay đổi!
        editor.commit();
    }
}
```

Sử dụng bộ nhớ trong

Bạn có thể trực tiếp lưu file vào bộ nhớ trong của thiết bị. Mặc định, các file được lưu vào bộ nhớ trong là của riêng ứng dụng đó, nên các ứng dụng (hay người dùng) khác không thể truy cập vào. Khi người dùng gỡ ứng dụng, những file này sẽ bị xóa bỏ.

Để tạo và ghi một file riêng vào bộ nhớ trong:

1. Gọi phương thức `openFileOutput()` với tên file và chế độ hoạt động. Kết quả trả về sẽ là một `FileOutputStream`.
2. Ghi vào file với phương thức `write()`.
3. Đóng luồng dữ liệu bằng phương thức `close()`.

Ví dụ:

```
String FILENAME =”hello_file”;
Stringstring=”hello world!”;

FileOutputStream fos = openFileOutput(FILENAME,Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Chế độ MODE_PRIVATE sẽ tạo file mới (hoặc thay thế một file cùng tên) và đặt file này là của riêng ứng dụng. Bạn có thể dùng các chế độ khác như: MODE_APPEND, MODE_WORLD_READABLE và MODE_WORLD_WRITEABLE.

Để đọc file từ bộ nhớ trong:

1. Gọi phương thức openFileInput() và truyền tên file cần đọc cho hàm này. Kết quả trả về là một FileInputStream.
2. Đọc từng byte của file này bằng phương thức read().
3. Sau đó, đóng luồng này bằng lệnh close().

> **Mách nhỏ:** Nếu bạn muốn lưu một file tĩnh (static) trong ứng dụng vào thời điểm biên dịch chương trình, hãy lưu file này vào thư mục `res/raw/` trong dự án của bạn. Bạn có thể mở file này bằng phương thức openRawResource(), truyền vào ID tài nguyên `R.raw.`*<tên file>*. Phương thức này sẽ trả về một InputStream để bạn có thể đọc file (nhưng bạn không thể ghi vào file gốc).

Lưu file cache

Nếu chỉ muốn lưu cache (nhớ tạm) thay vì lưu lâu dài một vài dữ liệu, bạn nên dùng phương thức getCacheDir() để mở một File đại diện cho thư mục nội bộ, đây chính là nơi ứng dụng của bạn tạm lưu các file cache.

Khi thiếu dung lượng bộ nhớ trong của thiết bị, Android có thể xóa những file cache này để khôi phục không gian bộ nhớ. Tuy nhiên, bạn không nên để cho hệ thống tự động xóa các file của bạn. Hãy quản lý các file cache của mình và đặt vào một không gian nhớ có dung lượng hợp lý, chẳng hạn như 1MB. Khi người dùng gỡ ứng dụng của bạn, những file này sẽ bị xóa bỏ.

Các phương thức hữu dụng khác

getFilesDir()

Lấy đường dẫn tuyệt đối của thư mục chứa file hệ thống, cũng là nơi lưu file nội bộ trong ứng dụng của bạn.

getDir()

Tạo thư mục mới (hoặc mở thư mục đã có) nằm trong bộ nhớ trong.

deleteFile()

Xóa một file lưu ở bộ nhớ trong.

`fileList()`

Trả về một mảng các file đang được lưu trong ứng dụng của bạn.

Sử dụng bộ nhớ ngoài

Mọi thiết bị tương thích với Android đều hỗ trợ một "bộ nhớ ngoài" ("external storage") cùng chia sẻ cho bạn lưu file. Bộ nhớ ngoài này có thể là một thiết bị lưu trữ có thể tháo rời (removable storage media) (chẳng hạn như một thẻ nhớ SD) hoặc một bộ nhớ trong (không thể tháo rời). Người dùng có thể đọc được và có thể chỉnh sửa các file được lưu vào bộ nhớ ngoài khi sử dụng chế độ lưu trữ bộ nhớ USB (USB mass storage) để đưa file vào một máy tính.

Một thiết bị có thể dùng một phân vùng của bộ nhớ trong làm bộ nhớ ngoài có thể vẫn cung cấp một khe cắm thẻ SD (SD card slot). Trong trường hợp này, thẻ SD không phải là một phần của bộ nhớ ngoài và ứng dụng của bạn không thể truy cập vào đó (phần lưu trữ bổ sung này chỉ dành cho các file đa phương tiện mà người dùng cung cấp và hệ thống có thể quét được).

Lưu ý: Bộ nhớ ngoài có thể mất tác dụng nếu người dùng gắn (mount) bộ nhớ này vào một máy tính hoặc tháo rời thẻ nhớ; bởi khi đó, những file bạn đã lưu vào bộ nhớ ngoài sẽ không được bảo mật. Tất cả các ứng dụng có thể đọc và ghi file đặt trong bộ nhớ ngoài và người dùng có thể xóa chúng.

Kiểm tra sự sẵn sàng của các thẻ nhớ

Trước khi làm việc với bộ nhớ ngoài, bạn nên gọi hàm `getExternalStorageState()` để kiểm tra xem thẻ nhớ có sẵn sàng hay không. Thẻ nhớ có thể đang được gắn vào máy tính, bị lỗi, chỉ cho phép đọc, hoặc đang trong một vài tình trạng khác. Sau đây là ví dụ về cách kiểm tra sự sẵn sàng của thẻ nhớ:

```
boolean mExternalStorageAvailable =false;
boolean mExternalStorageWriteable =false;
String state =Environment.getExternalStorageState();

if(Environment.MEDIA_MOUNTED.equals(state)){
    // Chúng ta có thể đọc và ghi thẻ nhớ
    mExternalStorageAvailable = mExternalStorageWriteable =true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
    // Chúng ta chỉ có thể đọc thẻ nhớ
    mExternalStorageAvailable =true;
    mExternalStorageWriteable =false;
} else {
    // Có một số lỗi xảy ra. Thiết bị có thể ở trong một trang thái
    // nào đó khác, nhưng tất cả những gì chúng ta cần biết là
    // không thể đọc hoặc ghi vào bộ nhớ ngoài
    mExternalStorageAvailable = mExternalStorageWriteable =false;
}
```

Ví dụ trên nhằm kiểm tra xem bộ nhớ ngoài có sẵn sàng đọc và ghi hay không.

Phương thức `getExternalStorageState()` trả về một số trạng thái mà bạn có thể muốn kiểm tra, chẳng hạn như thẻ nhớ có đang bị chia sẻ hay không (đang được kết nối với máy tính), đang bị lỗi, hoặc đã bị tháo rời,... Bạn có thể sử dụng những thông tin này để thông báo chi tiết cho người dùng biết khi ứng dụng của bạn cần truy cập thẻ nhớ.

Truy cập file trên bộ nhớ ngoài

Nếu bạn sử dụng API (Application programming interface - Giao diện lập trình ứng dụng) Cấp 8 hoặc cấp cao hơn, hãy sử dụng phương thức `getExternalFileDir()` để mở một `File` đại diện cho thư mục bộ nhớ ngoài - là nơi bạn nên dùng để lưu các file của mình. Phương thức này cần truyền vào một tham số `type` để xác định kiểu thư mục con (subdirectory) bạn muốn dùng, chẳng hạn như `DIRECTORY_MUSIC` và `DIRECTORY_RINGTONES` (nếu truyền vào `null` sẽ nhận được thư mục gốc chứa các file trong ứng dụng của bạn). Phương thức này cũng sẽ tạo ra một thư mục phù hợp, nếu cần thiết. Bằng việc xác định kiểu thư mục, bạn có thể đảm bảo rằng media scanner (trình quét đa phương tiện) của Android có thể phân loại đúng các file của bạn trong hệ thống (ví dụ, chuông điện thoại được nhận diện đúng là chuông điện thoại và khác với âm nhạc). Nếu người dùng gỡ ứng dụng ra, thư mục này cùng với nội dung bên trong sẽ bị xóa.

Nếu bạn sử dụng API Cấp 7 hoặc cấp thấp hơn, hãy dùng phương thức `getExternalStorageDirectory()`, để mở `File` đại diện cho thư mục gốc của bộ nhớ ngoài. Sau đó, bạn nên ghi dữ liệu vào thư mục sau:

```
/Android/data/<package_name>/files/
```

Trong đó, `<package_name>` là tên gói theo định dạng Java, chẳng hạn như `"com.example.android.app"`. Nếu thiết bị của người dùng đang chạy API Cấp 8 hoặc cấp cao hơn, khi gỡ ứng dụng, thư mục này cùng với nội dung bên trong sẽ bị xóa.

Việc lưu file chia sẻ

Ẩn file khỏi media scanner

Thêm một file trống có tên là `.nomedia` vào thư mục chứa các file lưu trong bộ nhớ ngoài (lưu ý tiền tố dấu chấm trong tên file). Việc này sẽ ngăn không cho media scanner đọc file đa phương tiện của bạn hoặc đưa chúng vào những ứng dụng như Gallery hoặc Music.

Nếu bạn muốn lưu file không chỉ dành riêng cho ứng dụng của bạn và không bị xóa khi người dùng gỡ ứng dụng đó, hãy lưu các file này vào một trong những thư mục public (công khai) của bộ nhớ ngoài. Các thư mục này nằm ở thư mục gốc của bộ nhớ ngoài, chẳng hạn như `Music/, Pictures/, Ringtones/,...`

Trong API Cấp 8 hoặc cấp cao hơn, hãy sử dụng `getExternalStoragePublicDirectory()`, truyền cho phương thức này kiểu thư mục public muốn dùng, chẳng hạn như `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`,... Phương thức này sẽ tạo ra một thư mục phù hợp, nếu cần.

Nếu bạn sử dụng API Cấp 7 hoặc cấp thấp hơn, hãy dùng phương thức `getExternalStorageDirectory()` để mở `file` đại diện cho thư mục gốc của bộ nhớ ngoài, sau đó lưu các file chia sẻ này vào một trong những thư mục dưới đây:

- `Music/` - Media scanner xác định tất cả các file đa phương tiện trong thư mục này là âm nhạc của người dùng.
- `Podcasts/` - Media scanner xác định tất cả các file đa phương tiện trong thư mục này là podcast (file âm thanh).
- `Ringtones/` - Media scanner xác định các file đa phương tiện ở đây là nhạc chuông.
- `Alarms/` - Media scanner xác định các file ở đây là âm báo thức (alarm sound).
- `Notifications/` - Media scanner xác định các file ở đây là âm nhắc nhở (notification sound).
- `Pictures/` - Chứa tất cả các ảnh (ngoại trừ những ảnh chụp bằng chức năng chụp ảnh).
- `Movies/` - Tất cả các đoạn phim (ngoại trừ những đoạn video từ chức năng quay phim của thiết bị).
- `Download/` - Các file hỗn hợp tải về.

Lưu file cache

Nếu bạn sử dụng API Cấp 8 hoặc cấp cao hơn, hãy dùng `getExternalCacheDir()` để mở một `File` đại diện cho thư mục bộ nhớ ngoài - là nơi dùng để lưu các file cache. Nếu người dùng gỡ bỏ ứng dụng của bạn, những file này sẽ tự động bị xóa. Tuy nhiên, trong suốt vòng đời của ứng dụng, bạn nên quản lý các file cache này và loại bỏ những file không cần thiết để tiết kiệm không gian nhớ.

Nếu bạn dùng API Cấp 7 hoặc cấp thấp hơn, hãy sử dụng phương thức `getExternalStorageDirectory()` để mở một `File` đại diện cho thư mục gốc của bộ nhớ ngoài, sau đó ghi dữ liệu cache vào thư mục dưới đây:

`/Android/data/<package_name>/cache/`

Trong đó, `<package_name>/` là tên gói theo định dạng Java, chẳng hạn như "`com.example.android.app`".

Sử dụng cơ sở dữ liệu

Hệ điều hành Android hỗ trợ đầy đủ cho cơ sở dữ liệu SQLite. Bất cứ cơ sở dữ liệu nào bạn tạo đều có thể truy cập được theo tên từ bất kỳ lớp nào trong ứng dụng, nhưng không thể truy cập được từ bên ngoài ứng dụng.

Phương pháp được khuyên dùng để tạo cơ sở dữ liệu SQLite mới là tạo một lớp con của `SQLiteOpenHelper` và ghi đè lên phương thức `onCreate()`, tại đây bạn có thể thực thi lệnh SQLite để tạo bảng cho cơ sở dữ liệu. Ví dụ:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION =2;
    private static final String DICTIONARY_TABLE_NAME =”dictionary”;
    private static final String DICTIONARY_TABLE_CREATE =
            “CREATE TABLE “+ DICTIONARY_TABLE_NAME +” (“+
            KEY_WORD +” TEXT, “+
            KEY_DEFINITION +” TEXT);”;

    DictionaryOpenHelper(Context context){
        super(context, DATABASE_NAME,null, DATABASE_VERSION);
    }

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

Sau đó, bạn có thể lấy một thể hiện của `SQLiteOpenHelper` đã cài đặt bằng cách sử dụng phương thức khởi tạo đã định nghĩa. Để ghi và đọc từ cơ sở dữ liệu, gọi các phương thức `getWritableDatabase()` và `getReadableDatabase()` tương ứng. Cả hai phương thức này đều trả về một đối tượng `SQLiteDatabase` đại diện cho cơ sở dữ liệu, đồng thời cung cấp các phương thức để thao tác với SQLite.

Hệ điều hành Android hỗ trợ không giới hạn đối với những khái niệm SQLite chuẩn. Chúng ta được khuyến nghị nên thêm một trường khóa giá trị tăng tự động (autoincrement value key field) để có thể sử dụng làm trường định danh duy nhất nhằm tăng khả năng tìm kiếm nhanh một bản ghi (record). Điều này không bắt buộc đối với dữ liệu riêng, nhưng nếu cài đặt một content provider, bạn phải thêm vào một trường định danh duy nhất bằng cách sử dụng hằng (constant) `BaseColumns._ID`.

Bạn có thể thực thi truy vấn SQLite nhờ sử dụng phương thức `SQLiteDatabase.query()`. Phương thức này có thể nhận nhiều tham số truy vấn khác nhau, chẳng hạn như bảng để truy vấn, phép chiếu (projection), phép chọn (selection), cột (column), nhóm (grouping),… Đối với những truy vấn phức tạp, chẳng hạn như truy vấn cần định danh cột, bạn nên sử dụng `SQLiteQueryBuilder`, vì trình này cung cấp một số phương thức thuận tiện để xây dựng truy vấn.

Mỗi truy vấn SQLite sẽ trả về một con trỏ (`cursor`) trỏ tới toàn bộ hàng do truy vấn tìm được. Con trỏ luôn luôn là một cơ chế giúp bạn có thể duyệt kết quả từ một truy vấn cơ sở dữ liệu cũng như đọc dữ liệu trên hàng và cột.

Để xem xét ứng dụng mẫu minh họa cho cách sử dụng cơ sở dữ liệu SQLite trong Android, mời bạn tham khảo ứng dụng Note Pad và Searchable Dictionary (Từ điển có thể tra cứu được).

Gỡ lỗi cơ sở dữ liệu

Android SDK bao gồm công cụ cơ sở dữ liệu `sqlite3` cho phép bạn tìm duyệt nội dung bảng, chạy lệnh SQL và thực hiện một số chức năng hữu ích khác trên cơ sở dữ liệu SQLite. Xem mục "Examining sqlite3 databases from a remote shell" ("Tìm hiểu về cơ sở dữ liệu sqlite3 từ chương trình dòng lệnh ở xa") để học thêm về cách chạy công cụ này.

Sử dụng kết nối mạng

Bạn có thể sử dụng mạng (khi mạng đã sẵn sàng) để lưu trữ và truy xuất dữ liệu trên các dịch vụ dựa trên Web của riêng bạn. Để thao tác trên mạng, sử dụng các lớp trong các gói sau đây:

- `java.net.*`
- `android.net.*`

10. Content Provider

Content provider quản lý việc truy cập tập dữ liệu có cấu trúc. Content provider đóng gói dữ liệu, đồng thời cung cấp cơ chế để quy định việc bảo mật dữ liệu. Content provider là giao diện chuẩn giúp kết nối dữ liệu trong một tiến trình với mã chạy trong tiến trình khác.

Nếu muốn truy cập dữ liệu trong content provider, bạn sử dụng đối tượng ContentResolver trong Context (ngữ cảnh) của ứng dụng để kết nối content provider như một client. Đối tượng ContentResolver trao đổi với đối tượng provider, một thể hiện của lớp thực thi lớp ContentProvider. Đối tượng provider này nhận dữ liệu yêu cầu từ client, thực hiện các hoạt động được yêu cầu đó và trả về kết quả.

Nếu không có ý định chia sẻ dữ liệu với các ứng dụng khác, bạn không nhất thiết phải phát triển provider riêng. Tuy nhiên, nếu muốn cung cấp gợi ý cho khách hàng để tìm kiếm trong ứng dụng, bạn bắt buộc phải tạo content provider. Hoặc, nếu muốn sao chép và dán file hoặc dữ liệu phức tạp từ ứng dụng của mình sang ứng dụng khác, bạn cũng cần tới provider này.

Bản thân hệ điều hành Android đã bao gồm các content provider quản lý dữ liệu như nhạc, video, hình ảnh và thông tin liên lạc cá nhân. Bạn có thể thấy một số provider này được liệt kê trong tài liệu tham khảo của gói `android.provider`. Bất cứ ứng dụng Android nào cũng có thể truy cập vào các content provider này.

10.1 Cơ bản về content provider

Content provider quản lý việc truy cập tới kho chứa dữ liệu trung tâm. Content provider là một phần của ứng dụng Android, thường cung cấp giao diện người dùng riêng để làm việc với dữ liệu. Tuy nhiên, dự định ban đầu là content provider sẽ được ứng dụng khác dùng, ứng dụng này truy cập vào provider bằng cách sử dụng một đối tượng client provider. Content provider và đối tượng client cung cấp đồng thời hỗ trợ một giao diện

chuẩn, nhất quán cho dữ liệu cũng như quá trình giao tiếp của các tiến trình bên trong (inter-process) và bảo mật việc truy cập dữ liệu.

Mục này sẽ giới thiệu những khái niệm cơ bản sau đây:

- Content Provider hoạt động như thế nào.
- API bạn dùng để truy xuất dữ liệu trong content provider.
- API bạn dùng để chèn, cập nhật hoặc xóa dữ liệu trong content provider.
- Các tính năng API khác khiến việc thao tác với content provider trở nên dễ dàng hơn.

Tổng quan

Content provider biểu diễn dữ liệu với các ứng dụng bên ngoài dưới dạng một hoặc một vài bảng tương tự với bảng thường thấy trong cơ sở dữ liệu quan hệ (relational database - RDB). Mỗi hàng đại diện cho một thể hiện của một kiểu dữ liệu bất kỳ mà content provider thu thập; mỗi cột trong hàng thể hiện một phần dữ liệu thu thập được của thể hiện.

Ví dụ, một content provider có sẵn trong nền tảng Android là user dictionary (từ điển người dùng), nơi lưu trữ cách viết của một số từ đặc biệt mà người dùng muốn lưu lại. Bảng 1 giới thiệu cách biểu diễn dữ liệu trong content provider này.

Bảng 1: Ví dụ về bảng từ điển của người dùng

word	app id	frequency	locale	_ID
Mapreduce	user1	100	en_US	1
Precompiler	user14	200	fr_FR	2
Applet	user2	225	fr_CA	3
Const	user1	255	pt_BR	4
Int	user5	100	en_UK	5

Trong Bảng 1, mỗi hàng tương ứng với một thể hiện của một từ không có trong từ điển chuẩn. Mỗi cột đại diện cho một dữ liệu nào đấy của từ, chẳng hạn như mã địa điểm (locale) - nơi từ xuất hiện lần đầu tiên . Tiêu đề cột là tên cột được lưu trong content provider. Muốn tham chiếu tới mã địa điểm trong một hàng, bạn có thể chỉ tới cột locale của hàng đó. Với content provider này, cột `_ID` đóng vai trò là "khóa chính" ("primary key") được content provider quản lý tự động.

> **Ghi chú:** Content provider không yêu cầu phải có trường khóa chính, content provider cũng không yêu cầu phải sử dụng `_ID` làm tên cột của trường khóa chính nếu như đã có một trường khác. Tuy nhiên, nếu muốn ràng buộc (bind) dữ liệu của một content provider với một ListView, bạn nhất định phải có tên cột là `_ID`. Yêu cầu này được giải thích cụ thể trong mục "Hiển thị kết quả của truy vấn" bên dưới.

Truy cập content provider

Một ứng dụng có thể truy cập vào dữ liệu của content provider bằng đối tượng client `ContentResolver`. Đối tượng này có những phương thức có tên giống y hệt đối tượng trong đối tượng provider, một thể hiện của một trong những lớp con của `ContentProvider`. Các phương thức của `ContentResolver` cung cấp các chức năng "CRUD" cơ bản của bộ nhớ lưu trữ lâu dài; trong đó, "CRUD" bao gồm: Create (tạo), retrieve (truy xuất), update (cập nhật) và delete (xóa).

Đối tượng `ContentResolver` trong tiến trình của ứng dụng và đối tượng `ContentProvider` trong ứng dụng có nhiệm vụ quản lý provider, tự động điều khiển việc giao tiếp của tiến trình bên trong. `ContentProvider` cũng hoạt động giống như một tầng trừu tượng (abstraction layer) nằm giữa kho chứa dữ liệu và việc hiển thị dữ liệu ra bên ngoài dưới dạng bảng.

> **Ghi chú**: Để truy cập một provider, ứng dụng thường phải yêu cầu các quyền cụ thể trong file kê khai của nó. Hướng dẫn chi tiết cho vấn đề này được trình bày trong mục "Quyền đối với content provider" bên dưới.

Ví dụ, để lấy ra một danh sách các từ (word) và mã địa điểm (locale) của chúng từ content provider user dictionary, bạn gọi phương thức `ContentResolver.query()`. Phương thức `query()` gọi phương thức `ContentProvider.query()` được content provider user dictionary định nghĩa. Các dòng mã sau minh họa lời gọi của phương thức `ContentResolver.query()`:

```
// Truy vấn user dictionary và trả về kết quả
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // URI của bảng lưu các từ
    mProjection,        // Các cột trong hàng kết quả
    mSelectionClause   // Điều kiện chọn
    mSelectionArgs,    // Điều kiện chọn
    mSortOrder);       // Trình tự sắp xếp của các hàng kết quả trả về
```

Bảng 2 giải thích các đối số của lệnh query(Uri, projection, selection, selectionArgs, sortOrder) tương ứng với câu lệnh SELECT trong SQL:

Bảng 2: So sánh lệnh Query() với truy vấn trong SQL.

Đối số của lệnh query()	Từ khóa/tham số của câu lệnh SELECT	Ghi chú
`Uri`	FROM *ten_bang*	Uri tương ứng với bảng trong content provider, có tên là `ten_bang`.
`Projection`	*cot,cot,cot,...*	`projection` là một mảng các cột nên được bao gồm trong mỗi hàng kết quả truy vấn.
`Selection`	WHERE *cot =gia_tri*	`selection` xác định điều kiện chọn hàng.
`selectionArgs`	Không có tương đương. Tham số lựa chọn thay thế "?" trong mệnh đề select.	
`sortOrder`	ORDER BY *cot,cot,...*	`sortOrder` xác định thứ tự các hàng xuất hiện trong con trỏ (cursor) được trả về.

Content URI

Content URI là định danh tài nguyên thống nhất (*uniform resource identifier - URI*) xác định dữ liệu trong content provider. Content URI bao gồm **chuỗi định danh** (tên tượng trưng cho toàn bộ provider) và **một đường dẫn** (một tên trỏ tới một bảng). Khi bạn gọi một phương thức client để truy cập vào bảng trong content provider, content URI của bảng đó chính là một trong các đối số.

Ở các dòng mã trên, hằng `CONTENT_URI` chứa content URI của bảng "words" trong từ user dictionary. Đối tượng `ContentResolver` phân tích để lấy ra chuỗi định danh (authority) của URI, đồng thời sử dụng chuỗi này để "phân giải" content provider bằng cách so sánh chuỗi định danh với bảng của hệ thống chứa những content provider đã khai báo. Sau đó, `ContentResolver` có thể gửi các đối số của truy vấn đến đúng provider.

`ContentProvider` sử dụng phần đường dẫn (path) của content URI để chọn bảng cần truy cập. Mỗi content provider thường có một **đường dẫn** cho mỗi bảng mà nó quản lý.

Trong các dòng mã trên, một URI đầy đủ của bảng "words" là:

```
content://user_dictionary/words
```

Trong đó, chuỗi `user_dictionary` là chuỗi định danh của provider, còn chuỗi `words` là đường dẫn của bảng. Chuỗi `content://` là **scheme** (thông tin để cung cấp cho Content Provider) luôn tồn tại, xác định đây là một content URI.

Nhiều content provider cho phép bạn truy cập vào một hàng đơn của một bảng bằng cách gắn thêm một giá trị ID vào cuối URI. Ví dụ, để truy xuất một hàng có `_ID` là 4 trong user dictionary, bạn có thể sử dụng content URI sau:

```
Uri singleUri =ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

Thông thường, bạn sử dụng giá trị id khi truy xuất một tập gồm nhiều hàng, sau đó muốn cập nhật hoặc xóa một trong những hàng đó.

Ghi chú: Lớp `Uri` và `Uri.Builder` chứa nhiều phương thức thuật tiện cho việc khởi tạo đúng các đối tượng `Uri` từ chuỗi một cách dễ dàng. `ContentUri` chứa các phương thức có thể dễ dàng gắn thêm giá trị id vào URI. Đoạn mã trên sử dụng phương thức `withAppendedId()` để gắn một `id` vào content URI của content provider user dictionary.

Truy xuất dữ liệu từ content provider

Mục này miêu tả cách truy xuất dữ liệu từ content provider, lấy content provider user dictionary làm ví dụ.

Để rõ ràng, các đoạn mã nhỏ trong mục này đều gọi phương thức `ContentResolver.query()` trong "luồng UI", hay "luồng giao diện người dùng" ("UI thread"). Tuy nhiên, khi viết mã chương trình thực, bạn nên thực hiện truy xuất không đồng bộ trong một luồng riêng. Để làm điều này, bạn sử dụng lớp `CursorLoader`, sẽ được miêu tả chi tiết trong tài liệu hướng dẫn "Loaders". Mặt khác, các dòng mã ở đây chỉ là những đoạn mã nhỏ; chúng không thể hiện một ứng dụng hoàn chỉnh.

Để truy xuất dữ liệu từ một content provider, bạn thực hiện theo các bước cơ bản sau:

1. Yêu cầu quyền truy cập đọc đối với content provider.
2. Định nghĩa đoạn mã gửi truy vấn đến content provider.

Yêu cầu quyền truy cập đọc

Để truy xuất dữ liệu trong một trình cung cấp, ứng dụng của bạn cần "quyền truy cập đọc" (read) cho content provider. Bạn không thể yêu cầu quyền này trong lúc chạy chương trình; thay vào đó, bạn phải chỉ ra là cần quyền này trong file kê khai bằng cách sử dụng phần tử `<uses-permission>` và tên chính xác của quyền được định nghĩa trong content provider. Khi sử dụng phần tử này trong file kê khai, có nghĩa là bạn đang "yêu cầu" quyền này cho ứng dụng của mình. Nếu người dùng cài ứng dụng của bạn, ngầm hiểu là họ đồng ý với yêu cầu này.

Để tìm đúng tên của quyền truy cập đọc cho content provider bạn dùng cũng như các tên quyền truy cập khác, hãy tìm trong tài liệu hướng dẫn sử dụng của content provider đó.

Các vai trò (role) của quyền truy cập content provider được miêu tả chi tiết trong mục "Quyền đối với content provider" bên dưới.

Content provider user dictionary định nghĩa quyền `android.permission.READ_USER_DICTIONARY` trong file kê khai; do đó, nếu muốn đọc content provider này, ứng dụng phải yêu cầu với tên quyền như vậy.

Xây dựng truy vấn

Bước tiếp theo trong việc truy xuất dữ liệu từ một content provider là xây dựng truy vấn. Đoạn mã nhỏ đầu tiên định nghĩa một số biến dùng để truy cập content provider user dictionary:

```
// Một mảng chuỗi định nghĩa các cột sẽ được trả về cho mỗi hàng (hay
// Projection)
String[] mProjection =
{
    UserDictionary.Words._ID,     // Hằng thuộc lớp Contract tương
                                  // ứng với tên cột _ID
    UserDictionary.Words.WORD, // Hằng thuộc lớp Contract với tên
                               // cột word
    UserDictionary.Words.LOCALE  // Hằng thuộc lớp Contract tương ứng
                                 // với tên cột locale
};

// Định nghĩa chuỗi chứa mệnh đề chọn
String mSelectionClause =null;

// Khởi tạo một mảng chứa các đối số chọn
String[] mSelectionArgs ={""};
```

Đoạn mã nhỏ tiếp theo giới thiệu cách sử dụng hàm `ContentResolver.query()`, dùng content provider user dictionary làm ví dụ. Một truy vấn client tới content provider tương tự một truy vấn SQL, truy vấn này cũng chứa một tập cột trả về, một tập điều kiện chọn và thứ tự sắp xếp.

Một tập cột mà truy vấn trả về được gọi là một phép chiếu (projection) (tương ứng với biến `mProjection`).

Biểu thức xác định các hàng truy vấn được chia làm mệnh đề chọn và đối số chọn (selection argument).

Mệnh đề chọn là sự kết hợp của biểu thức Boolean và biểu thức lôgíc (biểu thức lôgíc có giá trị true/false), tên cột và giá trị (tương ứng với biến `mSelectionClause`). Liệu có thể xác định tham số có thể thay thế được không? Thay vì một giá trị, phương thức query truy xuất giá trị từ mảng đối số chọn (tương ứng với biến `mSelectionArgs`).

Trong đoạn mã nhỏ tiếp theo, nếu người dùng không nhập vào một từ, mệnh đề chọn được thiết lập giá trị là `null`, truy vấn sẽ trả về tất cả các từ trong content provider. Nếu người dùng nhập vào một từ, mệnh đề chọn được đặt là

UserDictionary.Words.WORD + " = ?" và phần tử đầu tiên của mảng đối số chọn được thiết lập bằng từ mà người dùng nhập vào.

```
 /*
 * Lệnh sau đây định nghĩa một mảng chuỗi gồm một phần tử chứa đối
số chọn.
 */
String[] mSelectionArgs ={""};

// Lấy một từ trong giao diện người dùng
mSearchString = mSearchWord.getText().toString();

// Hãy nhớ chèn thêm mã vào đây để kiểm tra đầu vào không hợp lệ và
// mã độc (malicious) làm hỏng chương trình.

// Nếu từ do người dùng nhập vào là chuỗi rỗng, lấy ra toàn bộ từ điển
if(TextUtils.isEmpty(mSearchString)){
    // Việc thiết lập mệnh đề chọn là null sẽ trả về toàn bộ từ trong
    // từ điển
    mSelectionClause = null;
    mSelectionArgs[0]= "";

} else {
    // Khởi tạo một mệnh đề chọn phù hợp với từ mà người dùng nhập vào.
    mSelectionClause =UserDictionary.Words.WORD +" = ?";

    // Gán chuỗi người dùng nhập vào cho đối số chọn.
    mSelectionArgs[0]= mSearchString;

}

// Thực hiện truy vấn trên bảng dữ liệu và trả về một đối tượng
// Cursor (con trỏ)
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,  // content URI của bảng words
    mProjection,        // Các cột trả về trong từng hàng kết quả
    mSelectionClause   // Hoặc null, hoặc là từ mà người dùng nhập vào
    mSelectionArgs,   // Hoặc rỗng, hoặc là chuỗi người dùng nhập vào
    mSortOrder);     // Thứ tự sắp xếp của các hàng được trả về

// Một số content provider trả về null nếu có lỗi xảy ra, một // số
khác sẽ ném (throw) ra một ngoại lệ (exception)
if(null== mCursor){
/*
    * Chèn thêm mã vào đây để xử lý lỗi. Hãy nhớ, đừng sử dụng con
    * trỏ! Có thể bạn muốn gọi phương thức android.util.Log.e() để
    * ghi lại lỗi xảy ra.
    *
```

```
    */
// Nếu con trỏ rỗng, content provider không tìm được từ phù hợp
} else if (mCursor.getCount()<1){
    /*
    * Chèn thêm mã vào đây để thông báo với người dùng rằng việc tìm
    * kiếm không thành công.  Việc này không nhất thiết là do lỗi. Có
    * thể ở đây, bạn muốn cung cấp cho người dùng tùy chọn để chèn
    * thêm hàng mới, hoặc nhập lại từ muốn tìm.
    */
} else {
    // Chèn thêm mã vào đây để thực hiện một số thao tác với kết quả trả về
}
```

Truy vấn trên tương tự câu lệnh SQL sau:

```
SELECT _ID, word, locale FROM words WHERE word = <dữ liệu người dùng
nhập> ORDER BY word ASC;
```

Câu lệnh SQL này sử dụng trực tiếp tên cột thực thụ thay vì các hằng định nghĩa trong lớp contract.

Bảo vệ ứng dụng khỏi việc nhập mã độc

Nếu dữ liệu được content provider quản lý là một cơ sở dữ liệu SQL, chứa cả những dữ liệu không đáng tin bên ngoài đưa vào câu lệnh SQL thô, có thể dẫn đến tình trạng làm hỏng cơ sở dữ liệu.

Hãy xem mệnh đề chọn dưới đây:

```
// Tạo một mệnh đề chọn bằng cách ghép dữ liệu người dùng nhập vào
// với tên cột
String mSelectionClause =  "var = "+ mUserInput;
```

Nếu thực hiện điều này, vô hình trung bạn đang cho phép người dùng ghép mã SQL độc vào câu lệnh SQL. Ví dụ, người dùng có thể nhập "nothing; DROP TABLE *;" cho mUserInput, kết quả được mệnh đề chọn var = nothing; DROP TABLE *;. Nếu mệnh đề chọn này được coi như một câu lệnh SQL, nó có thể khiến content provider xóa toàn bộ bảng trong cơ sở dữ liệu do SQLite quản lý, trừ phi content provider được thiết lập để bắt các câu lệnh SQL gây hại (SQL injection).

Để tránh vấn đề này, sử dụng mệnh đề chọn có dùng dấu ? làm tham số thay thế và một mảng riêng bao gồm các đối số chọn. Khi thực hiện điều này, dữ liệu người dùng nhập vào được gắn trực tiếp với truy vấn thay vì được biên dịch dưới dạng câu lệnh SQL. Do không được xử lý dưới dạng câu lệnh SQL, nên dữ liệu người dùng nhập không thể đưa mã SQL độc vào. Thay vì ghép trực tiếp dữ liệu người dùng nhập vào, có thể sử dụng mệnh đề chọn sau:

```
// Tạo mệnh đề chọn với đối số có thể thay thế
String mSelectionClause =  "var = ?";
```

Tạo một mảng đối số chọn như sau:

```
// Định nghĩa một mảng chứa các đối số đề chọn
String[] selectionArgs ={""};
```

Đưa một giá trị vào mảng các đối số chọn như sau:

```
// Thiết lập đối số chọn là dữ liệu người dùng nhập vào
selectionArgs[0]= mUserInput;
```

Một mệnh đề chọn sử dụng dấu ? làm tham số có thể thay thế và một mảng các đối số chọn là cách được ưa dùng hơn khi xây dựng một phép chọn, ngay cả với trường hợp content provider không dựa trên cơ sở dữ liệu SQL.

Hiển thị kết quả truy vấn

Một phương thức client `ContentResolver.query()` luôn trả về đối tượng `Cursor` chứa các hàng phù hợp với điều kiện chọn của truy vấn, và mỗi hàng lại bao gồm các cột do đối số projection chỉ định. Một đối tượng `Cursor` cung cấp khả năng truy cập ngẫu nhiên tới hàng và cột nó chứa. Bằng cách sử dụng các phương thức của `Cursor`, bạn có thể duyệt qua các hàng kết quả, xác định kiểu dữ liệu của từng cột, lấy dữ liệu ra khỏi một cột, đồng thời kiểm tra các thuộc tính khác của kết quả. Một số `Cursor` được cài đặt để tự động cập nhật đối tượng khi dữ liệu của content provider thay đổi, hoặc sử dụng phương thức trigger bên trong một đối tượng giám sát (observer) mỗi khi `Cursor` thay đổi, hoặc cả hai.

> **Ghi chú:** Một số content provider có thể hạn chế việc truy cập tới các cột dựa trên đặc tính tự nhiên của đối tượng thực hiện truy vấn. Ví dụ, Provider Contacts giới hạn việc truy cập trên một số cột để đồng bộ các adapter; do đó, provider này sẽ không trả về những cột bị giới hạn cho một activity hay service.

Nếu không có hàng nào phù hợp với điều kiện chọn, content provider sẽ trả về một đối tượng `Cursor` với phương thức `Cursor.getCount()` cho kết quả là 0 (một con trỏ rỗng).

Nếu xảy ra lỗi nội bộ, kết quả của truy vấn sẽ tùy thuộc vào từng provider cụ thể: Có thể trả về `null`, hoặc ném ra một ngoại lệ (Exception).

Vì đối tượng `Cursor` là một "danh sách" các hàng, nên cách tốt nhất để hiển thị nội dung của `Cursor` là liên kết nó với một `ListView` thông qua một đối tượng `SimpleCursorAdapter`.

Đoạn mã nhỏ dưới đây là phần tiếp theo của đoạn mã nhỏ trước. Phần này sẽ tạo một đối tượng `SimpleCursorAdapter` chứa `Cursor` là kết quả của truy vấn, đồng thời đặt đối tượng `SimpleCursorAdapter` này làm adapter của `ListView`:

```
// Định nghĩa một danh sách cột để truy xuất dữ liệu từ Cursor và đưa
// vào một hàng đầu ra
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,   // Hằng của lớp contract chứa tên cột word
    UserDictionary.Words.LOCALE  // Hằng của lớp contract chứa tên cột locale
};

// Định nghĩa một danh sách gồm các View ID sẽ nhận các cột của mỗi hàng
int[] mWordListItems ={ R.id.dictWord, R.id.locale};

// Tạo một đối tượng SimpleCursorAdapter mới
mCursorAdapter =newSimpleCursorAdapter(
    getApplicationContext(),    // Đối tượng Context của ứng dụng
    R.layout.wordlistrow, // Cách biểu diễn trong XML cho mỗi hàng ListView
    mCursor,           // Kết quả của truy vấn
    mWordListColumns, // Một mảng chuỗi chứa các tên cột của con trỏ
    mWordListItems,   // Một mảng số nguyên chứa các ID hiển thị trong
                      // layout hàng
    0);              // Cờ (thường không dùng tới)

// Gán adapter cho ListView
mWordList.setAdapter(mCursorAdapter);
```

Ghi chú: Để đưa `Cursor` về dạng `ListView`, con trỏ phải chứa một cột có tên là `_ID`. Do đó, truy vấn ở trên mới lấy ra cột `_ID` của bảng "words", ngay cả khi `ListView` không hiển thị cột này. Hạn chế này cũng lý giải tại sao hầu hết content provider đều có cột `_ID` trong mỗi bảng.

Lấy dữ liệu từ kết quả truy vấn

Thay vì chỉ hiển thị kết quả truy vấn, bạn có thể sử dụng kết quả này cho các mục đích khác. Ví dụ, bạn có thể truy xuất spelling (cách viết từ/đánh vần) trong user dictionary, sau đó tìm kiếm chúng trong content provider khác. Để thực hiện điều này, bạn duyệt qua các hàng trong `Cursor`:

```
// Xác định chỉ số cột cho cột có tên "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
* Chỉ tiến hành duyệt khi con trỏ trả về hợp lệ. Content provider user
* dictionary trả về null nếu có lỗi bên trong xảy ra. Các  trình cung
* cấp nội dung khác có thể sẽ ném ra một ngoại lệ thay vì trả về null.
*/

if(mCursor != null){
    /*
    * Di chuyển tới hàng tiếp theo trong con trỏ. Trước lần di chuyển
```

```
    * đầu tiên, "con trỏ chỉ hàng" là -1, và nếu cố gắng truy xuất dữ
    * liệu tại vị trí này, bạn sẽ nhận được một ngoại lệ.
    */
    while(mCursor.moveToNext()){

        // Lấy giá trị từ cột.
        newWord = mCursor.getString(index);

        // Chèn thêm mã vào đây để thao tác với từ truy vấn được.

        ...

        // kết thúc vòng lặp while
    }
} else {

    // Chèn thêm mã vào đây để báo cáo lỗi khi con trỏ null hoặc
    // content provider ném ra một ngoại lệ.
}
```

Đối tượng `Cursor` thực thi chứa một số phương thức "get" dùng cho vài kiểu truy xuất dữ liệu khác nhau. Ví dụ, đoạn mã nhỏ trước sử dụng `getString()`. Chúng ta cũng có thể dùng phương thức `getType()` trả về giá trị cho biết kiểu dữ liệu của cột.

Quyền đối với content provider

Một ứng dụng của content provider có thể quy định quyền mà những ứng dụng khác phải đăng ký để truy cập dữ liệu của content provider này. Các quyền này đảm bảo người dùng biết được một ứng dụng sẽ truy cập vào dữ liệu nào. Dựa trên yêu cầu của từng content provider, các ứng dụng khác yêu cầu quyền mà chúng cần theo quy định để truy cập vào provider. Người dùng cuối (end user) sẽ thấy các quyền yêu cầu này khi cài đặt ứng dụng.

Nếu ứng dụng của content provider không xác định bất cứ quyền nào, ứng dụng khác sẽ không có quyền truy cập vào dữ liệu của provider này. Tuy nhiên, các thành phần trong ứng dụng của content provider luôn có đủ quyền đọc và ghi, mà không phải đăng ký những quyền được xác định này.

Như đã lưu ý từ trước, content provider user dictionary yêu cầu quyền `android.permission.READ_USER_DICTIONARY` để truy xuất dữ liệu trong provider này. Content provider user dictionary cũng định nghĩa quyền `android.permisstion.WRITE_USER_DICTIONARY` riêng cho việc chèn, cập nhật hoặc xóa dữ liệu.

Để nhận được các quyền cần thiết khi truy cập content provider, ứng dụng yêu cầu quyền này bằng phần tử `<uses-permission>` trong file kê khai của ứng dụng đó. Khi trình quản lý gói ứng dụng Android (Android Package Manager) cài đặt ứng dụng, người dùng phải chấp thuận tất cả quyền mà ứng dụng yêu cầu. Nếu người dùng chấp thuận tất cả những quyền này, trình quản lý gói ứng dụng mới tiếp tục cài đặt; nếu người dùng không đồng ý, trình quản lý gói ứng dụng sẽ hủy quá trình cài đặt.

Phần tử <uses-permission> dưới đây yêu cầu quyền đọc đối với content provider user dictionary:

```
<uses-permissionandroid:name="android.permission.READ_USER_DICTIONARY">
```

Tác dụng của quyền đối với việc truy cập content provider được giải thích chi tiết hơn trong tài liệu hướng dẫn "Security and Permissions" ("Quyền và bảo mật").

Chèn, cập nhật và xóa dữ liệu

Tương tự như cách truy xuất dữ liệu trong provider, bạn cũng có thể sử dụng sự tương tác giữa một provider client và ContentProvider của provider để chỉnh sửa dữ liệu. Bạn gọi một phương thức của ContentResolver với các đối số truyền tương ứng với phương thức của ContentProvider. Content provider và provider client sẽ tự động xử lý việc bảo mật cũng như giao tiếp bên trong tiến trình.

Chèn dữ liệu

Để chèn dữ liệu vào một content provider, bạn gọi phương thức ContentResolver.insert(). Phương thức này sẽ chèn một hàng mới vào content provider và trả về một content URI (định danh tài nguyên thống nhất của nội dung) của hàng đó. Đoạn mã nhỏ dưới đây minh họa cách chèn một hàng mới vào content provider user dictionary:

```
// Định nghĩa một đối tượng Uri mới sẽ nhận kết quả trả về khi chèn
// dữ liệu
Uri mNewUri;

...

// Định nghĩa một đối tượng chứa giá trị mới sẽ được chèn thêm
ContentValues mNewValues = new ContentValues();

/*
* Thiết lập giá trị cho từng cột và chèn từ vào từ điển. Đối số của
* phương thức "put" là "tên cột" và "giá trị"
*/
mNewValues.put(UserDictionary.Words.APP_ID,"example.user");
mNewValues.put(UserDictionary.Words.LOCALE,"en_US");
mNewValues.put(UserDictionary.Words.WORD,"insert");
mNewValues.put(UserDictionary.Words.FREQUENCY,"100");

mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // content URI của user dictionary
     mNewValues                      // các giá trị để chèn
);
```

Dữ liệu của hàng mới được đưa vào một đối tượng ContentValues, đối tượng này có dạng tương tự như một con trỏ một hàng. Các cột trong đối tượng này không nhất

thiết phải cùng kiểu dữ liệu; nếu không muốn xác định giá trị nào, bạn có thể thiết lập cột đó là null bằng cách sử dụng phương thức ContentValues.putNull().

Đoạn mã nhỏ trên không thêm cột _ID, bởi cột này được duy trì tự động. Content provider gán một giá trị ID duy nhất cho mỗi hàng mới được thêm vào. Các content provider thường dùng giá trị này làm khóa chính của bảng.

Content URI trả về một URI mới xác định hàng mới thêm vào, dưới dạng sau:

```
content://user_dictionary/words/<id_value>
```

Trong đó, <id_value> là nội dung cột _ID của hàng mới. Đa số content provider đều có thể tự động nhận diện dạng content URI này, sau đó thực hiện các thao tác được yêu cầu trên hàng cụ thể đó.

Để lấy giá trị của cột _ID từ Uri được trả về, gọi phương thức ContentUris.parseId().

Cập nhật dữ liệu

Để cập nhật một hàng, bạn sử dụng đối tượng ContentValues() với các giá trị cập nhật như đã làm với thao tác chèn, và điều kiện chọn giống như khi bạn thực hiện một truy vấn. Phương thức client nên dùng là ContentResolver.update(). Bạn chỉ cần thêm các giá trị vào đối tượng ContentValues cho những cột cần cập nhật. Nếu muốn xóa nội dung một cột, hãy thiết lập giá trị đó là null.

Đoạn mã nhỏ dưới đây sẽ thay đổi tất cả các hàng có cột locale là “en” thành locale là null. Giá trị trả về là số lượng hàng được cập nhật:

```
// Tạo một đối tượng chứa các giá trị được cập nhật
ContentValues mUpdateValues = new ContentValues();

// Xác định điều kiện chọn cho những hàng muốn cập nhật
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs ={"en_%"};

// Tạo một biến chứa số lượng hàng được cập nhật
int mRowsUpdated =0;

...

/*
* Thiết lập giá trị được cập nhật và cập nhật những từ được chọn.
*/
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,  // content URI của user dictionary
    mUpdateValues                      // Các cột cập nhật
    mSelectionClause                  // cột được chọn
```

```
    mSelectionArgs                // giá trị để so sánh
);
```

Bạn cũng nên xử lý dữ liệu người dùng nhập vào trước khi gọi phương thức `ContentResolver.update()`. Để tìm hiểu thêm về cách xử lý, tham khảo mục "Bảo vệ ứng dụng khỏi việc nhập mã độc" bên trên.

Xóa dữ liệu

Xóa hàng tương tự như việc truy xuất hàng dữ liệu: Bạn xác định điều kiện chọn cho các hàng muốn xóa và phương thức client sẽ trả về số lượng hàng đã xóa. Đoạn mã nhỏ dưới đây sẽ xóa những hàng có cột "app id" bằng "user". Phương thức xóa trả về số lượng hàng được xóa.

```
// Xác định điều kiện chọn cho các hàng muốn xóa
String mSelectionClause = UserDictionary.Words.APP_ID +" LIKE ?";
String[] mSelectionArgs ={"user"};

// Tạo một biến lưu số lượng hàng được xóa
int mRowsDeleted =0;

...

// Xóa các từ phù hợp với điều kiện chọn
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,   // content URI của user dictionary
    mSelectionClause                   // cột để chọn
    mSelectionArgs                    // giá trị để so sánh
);
```

Bạn cũng nên xử lý trước dữ liệu người dùng nhập vào khi gọi phương thức `ContentResolver.delete()`. Để tìm hiểu thêm về cách xử lý, tham khảo mục "Bảo vệ ứng dụng khỏi việc nhập mã độc" bên trên.

Kiểu dữ liệu của provider

Content provider có thể lưu nhiều kiểu dữ liệu khác nhau. Content provider user dictionary chỉ xử lý dữ liệu dạng văn bản, nhưng content provider có thể lưu các định dạng dưới đây:

- integer.
- long integer (long).
- floating point.
- long floating point (double).

Một kiểu dữ liệu khác mà content provider thường sử dụng là đối tượng nhị phân lớn (Binary Large Object - BLOB), được cài đặt như một mảng byte 64 KB. Bạn có thể biết những kiểu dữ liệu có sẵn bằng cách xem các phương thức "get" của lớp `Cursor`.

Kiểu dữ liệu của mỗi cột trong content provider thường được liệt kê trong tài liệu hướng dẫn của trình đó. Các kiểu dữ liệu của content provider user dictionary được liệt kê trong tài liệu tham khảo về lớp contract `UserDictionary.Words` của trình này (các lớp contract sẽ được giới thiệu trong mục "Lớp contract" bên dưới). Bạn cũng có thể xác định kiểu dữ liệu bằng cách gọi phương thức `Cursor.getType()`.

Content provider cũng lưu giữ thông tin về kiểu dữ liệu MIME cho mỗi content URI mà trình đó định nghĩa. Bạn có thể sử dụng thông tin về kiểu MIME để khám phá xem liệu ứng dụng của bạn có thể xử lý dữ liệu mà content provider dùng hay không, hoặc để chọn một kiểu xử lý dựa trên kiểu MIME. Bạn thường cần kiểu MIME khi làm việc với content provider chứa cấu trúc hoặc file dữ liệu phức tạp. Ví dụ, bảng ContactsContract.Data trong Content Provider Contacts sử dụng kiểu MIME để dán nhãn kiểu dữ liệu liên lạc được lưu ở mỗi hàng. Để lấy kiểu MIME tương ứng với từng content URI, gọi phương thức ContentResolver.getType().

Mục "Tài liệu tham khảo về kiểu MIME" bên dưới sẽ giới thiệu cú pháp của cả kiểu MIME chuẩn lẫn kiểu MIME tùy chỉnh.

Các dạng truy cập content provider khác

Ba cách truy cập content provider quan trọng khác rất cần thiết cho việc phát triển ứng dụng là:

- Truy cập hàng loạt (batch access): Bạn có thể tạo một nhóm các lời gọi truy cập với những phương thức nằm trong lớp `ContentProviderOperation`, sau đó áp dụng nhóm này vào chương trình bằng phương thức `ContentResolver.applyBatch()`.
- Truy vấn không đồng bộ: Bạn nên thực hiện truy vấn trong một luồng riêng. Một cách để thực hiện điều này nhờ sử dụng đối tượng `CursorLoader`. Các ví dụ trong tài liệu hướng dẫn Loaders sẽ minh họa cách làm này.
- Truy cập dữ liệu thông qua intent: Mặc dù không thể gửi trực tiếp một intent tới content provider, song bạn có thể gửi intent tới ứng dụng của content provider - đây chính là công cụ chỉnh sửa dữ liệu tốt nhất trong content provider.

Hai cách truy cập và chỉnh sửa hàng loạt và thông qua intent sẽ được miêu tả trong các mục dưới đây.

Truy cập hàng loạt

Truy cập content provider hàng loạt là một cách hữu dụng để chèn một số lượng hàng lớn, hoặc để chèn hàng trong nhiều bảng cùng sử dụng lời gọi phương thức như nhau, hoặc thường là để thực hiện một chuỗi thao tác giống như một giao dịch (một thao tác đơn lẻ).

Để truy cập vào content provider theo "chế độ hàng loạt", bạn tạo một mảng các đối tượng `ContentProviderOperation`, sau đó đưa chúng vào content provider bằng phương thức `ContentResolver.applyBatch()`. Bạn truyền cho phương thức này *chuỗi định danh* của chính provider, thay vì một content URI cụ thể. Điều này cho phép mỗi đối tượng `ContentProviderOperation` trong mảng làm việc trên các

bảng khác nhau. Một lời gọi tới phương thức `ContentResolver.applyBatch()` trả về một mảng chứa các kết quả.

Phần giới thiệu về lớp contract `ContactsContract.RawContacts` chứa một đoạn mã nhỏ minh họa cho việc chèn hàng loạt. Ứng dụng mẫu Contact Manager chứa một ví dụ về việc truy cập hàng loạt trong file nguồn `ContactAdder.java`.

Hiển thị dữ liệu sử dụng ứng dụng trợ giúp

Nếu ứng dụng của bạn *đã* có quyền truy cập, bạn vẫn muốn sử dụng một intent để hiển thị dữ liệu trong ứng dụng khác. Ví dụ, ứng dụng Calendar (lịch ngày tháng) cho phép nhận intent `ACTION_VIEW` hiển thị một ngày tháng hoặc một sự kiện cụ thể. Điều này cho phép bạn hiển thị thông tin về lịch mà không phải tạo giao diện người dùng riêng. Để tìm hiểu thêm về tính năng này, xem tài liệu hướng dẫn "Calendar Provider" ("Provider Calendar").

Ứng dụng được gửi intent tới không cần phải là ứng dụng có liên kết với content provider. Ví dụ, bạn có thể truy xuất một liên lạc từ Content Provider Contacts, sau đó gửi một intent `ACTION_VIEW` chứa content URI của hình ảnh liên lạc tới một trình hiển thị hình ảnh (image viewer).

Truy cập dữ liệu thông qua intent

Intent có thể giúp bạn gián tiếp truy cập tới content provider. Bạn cho phép người dùng truy cập vào dữ liệu của content provider ngay cả khi ứng dụng của bạn không có quyền này, hoặc bằng cách lấy intent kết quả trả cho một ứng dụng có quyền, hoặc bằng việc kích hoạt một ứng dụng có quyền truy cập và cho phép người dùng làm việc với ứng dụng đó.

Thực hiện truy cập bằng quyền tạm

Ngay cả khi không có quyền truy cập dữ liệu trong content provider, bạn vẫn có thể làm điều này bằng cách gửi một intent tới ứng dụng có quyền và nhận về intent kết quả chứa quyền "URI". Đây là các quyền đối với một content URI nhất định sẽ tồn tại cho đến khi activity nhận intent này kết thúc. Ứng dụng có quyền cố định cấp quyền tạm bằng cách thiết lập một cờ (flag) trong intent kết quả:

- **Quyền đọc**: `FLAG_GRANT_READ_URI_PERMISSION`
- **Quyền ghi**: `FLAG_GRANT_WRITE_URI_PERMISSION`

> **Ghi chú:** Các cờ này không cấp quyền đọc và ghi chung tới content provider có chuỗi định danh nằm trong content URI. Các quyền truy cập chỉ có tác dụng trên chính URI đó.

Content provider định nghĩa các quyền URI cho content URI trong file kê khai của nó, sử dụng thuộc tính `android:grantUriPermission` của phần tử `<provider>`, có chức năng tương tự như phần tử con `<grant-uri-permission>` của phần tử `<provider>`. Cơ chế quyền URI được giải thích chi tiết hơn tại tài liệu hướng dẫn "Security and Permissions" ("Quyền và bảo mật"), trong mục "URI permissions" ("Quyền URI").

Ví dụ, ngay cả khi không có quyền READ_CONTACTS, bạn vẫn có thể truy xuất dữ liệu một liên lạc trong Content Provider Contacts. Có thể bạn muốn thực hiện điều này trong một ứng dụng gửi lời chúc điện tử tới người bạn muốn liên lạc trong ngày sinh nhật của anh/cô ấy. Thay vì yêu cầu quyền READ_CONTACTS, để có thể truy cập vào tất cả các liên lạc của người dùng và toàn bộ thông tin của họ, bạn muốn để người dùng quản lý những liên lạc được ứng dụng của bạn dùng. Để thực hiện điều này, bạn sử dụng tiến trình xử lý sau:

1. Ứng dụng của bạn gửi đi một intent chứa action ACTION_PICK và kiểu MIME chứa "liên lạc" CONTENT_ITEM_TYPE, dùng phương thức startActivityForResult().
2. Do intent trên phù hợp với bộ lọc intent của activity lựa chọn (selection) trong ứng dụng của người dùng (People), nên activity này sẽ hiển thị ở màn hình chính.
3. Trong activity lựa chọn (selection) này, người dùng chọn một liên lạc để cập nhật. Khi đó, activity lựa chọn sẽ gọi phương thức setResult(resultcode, intent) để tạo một intent trả về cho ứng dụng của bạn. Intent mới này chứa content URI của liên lạc mà người dùng chọn, đồng thời "thêm" cờ FLAG_GRANT_READ_URI_PERMISSION. Các cờ này sẽ gán quyền URI cho ứng dụng của bạn để đọc dữ liệu trong liên lạc được content URI trỏ tới. Sau đó, activity lựa chọn sẽ gọi phương thức finish() để trả điều khiển về cho ứng dụng của bạn.
4. Activity của bạn trở lại màn hình chính, và hệ thống gọi phương thức onActivityResult() trong activity của bạn. Phương thức này sẽ nhận intent kết quả được activity lựa chọn tạo ra trong ứng dụng People.
5. Với content URI trong intent kết quả, bạn có thể đọc dữ liệu liên lạc của Content Provider Contacts, ngay cả khi không yêu cầu quyền đọc lâu dài trong file kê khai. Sau đó, bạn có thể lấy thông tin về sinh nhật của liên lạc hoặc địa chỉ e-mail của người muốn liên lạc để gửi lời chúc điện tử.

Sử dụng ứng dụng khác

Một cách đơn giản để người dùng có thể chỉnh sửa trên dữ liệu mà bạn không có quyền truy cập là kích hoạt một ứng dụng có quyền và để người dùng thực hiện chỉnh sửa trên đó.

Ví dụ, ứng dụng Calendar (lịch ngày tháng) cho phép intent ACTION_INSERT, là intent cho phép bạn kích hoạt giao diện người dùng chèn của ứng dụng này. Bạn có thể truyền dữ liệu mới vào intent này, đây là nơi ứng dụng dùng giao diện để hiển thị. Do việc lặp lại sự kiện có cú pháp phức tạp, nên cách tốt hơn để chèn sự kiện vào Provider Calendar là kích hoạt ứng dụng Calendar với một ACTION_INSERT, sau đó để người dùng chèn sự kiện vào đó.

Lớp contract

Lớp contract định nghĩa các hằng số giúp ứng dụng làm việc với content URI, tên cột, action của intent và những tính năng khác của content provider. Lớp contract không tự động có trong content provider, nên nhà phát triển content provider phải định nghĩa chúng, sau đó xử lý để những nhà phát triển chương trình khác có thể sử dụng được.

Nhiều nhà phát triển tích hợp với nền tảng Android có các lớp contract tương ứng trong gói `android.provider`.

Ví dụ, content provider user dictionary có lớp contract `UserDictionary` chứa các hằng content URI và tên cột. Content URI của bảng "words" được định nghĩa trong hằng `UserDictionary.Words.CONTENT_URI`. Lớp `UserDictionary.Words` cũng chứa hằng tên cột, được sử dụng trong đoạn mã nhỏ minh họa dưới đây. Chẳng hạn, chúng ta có thể định nghĩa một phép chiếu truy vấn như sau:

```
String[] mProjection =
{
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};
```

Một lớp contract khác là `ContactsContract` của Provider Contacts. Tài liệu tham khảo của lớp này có bao gồm ví dụ về các đoạn mã nhỏ. Một lớp con của nó, `ContactsContract.Intents.Insert`, là một lớp contract chứa các hằng intent và dữ liệu intent.

Giới thiệu thêm về kiểu MIME

Content provider có thể trả về kiểu đa phương tiện MIME chuẩn, hay kiểu MIME tùy chỉnh, hoặc cả hai.

Kiểu MIME có định dạng như sau:

```
kiểu/kiểu con
```

Ví dụ, kiểu MIME thường được biết đến là `text/html` có kiểu là `text` và kiểu con là `html`. Nếu Provider trả kiểu này về cho một URI, có nghĩa là truy vấn sử dụng URI này sẽ trả về text chứa các thẻ HTML.

Chuỗi kiểu MIME tùy chỉnh, còn gọi là kiểu MIME có "đặc trưng của nhà cung cấp" ("vendor-specific" MIME type), có thêm nhiều giá trị *kiểu* và *kiểu con* phức tạp. Giá trị *kiểu* thường là

```
vnd.android.cursor.dir
```

cho nhiều hàng, hoặc

```
vnd.android.cursor.item
```

cho một hàng đơn.

Kiểu con phụ thuộc vào từng content provider cụ thể. Các content provider kiểu con trong Android thường có kiểu con đơn giản. Ví dụ, khi ứng dụng Contacts tạo một hàng

mới cho một số điện thoại, ứng dụng này sẽ thiết lập kiểu MIME dưới đây cho hàng mới đó:

```
vnd.android.cursor.item/phone_v2
```

kiểu con, giá trị `subtype` chỉ đơn giản là `phone_v2`.

Các nhà phát triển content provider khác có thể tạo những mẫu kiểu con của riêng họ dựa vào chuỗi định danh và tên bảng. Ví dụ, giả sử một content provider có bảng lưu thông tin thời gian các chuyến tàu (train time). Chuỗi định danh của trình này là com.`example.trains`, đồng thời chứa các bảng `Line1`, `Line2` và `Line3`. Để đáp ứng content URI

```
content://com.example.trains/Line1
```

cho bảng Line1, content provider trả về kiểu MIME

```
vnd.android.cursor.dir/vnd.example.line1
```

Để đáp ứng content URI

```
content://com.example.trains/Line2/5
```

cho hàng 5 trong bảng Line2, content provider trả về kiểu MIME

```
vnd.android.cursor.item/vnd.example.line2
```

Hầu hết content provider đều định nghĩa hằng cho các kiểu MIME mà trình này sử dụng trong lớp contract. Ví dụ, lớp contract của Provider Contacts là `ContactsContract.RawContacts`, định nghĩa hằng `CONTENT_ITEM_TYPE` cho kiểu MIME của một hàng liên lạc đơn.

Content URI của hàng đơn được miêu tả trong mục Content URIs.

10.2 Tạo content provider

Content Provider quản lý việc truy cập tới kho lưu trữ dữ liệu trung tâm. Bạn có thể cài đặt content provider thành một hoặc nhiều lớp trong ứng dụng Android, cùng với đó là các thành phần trong file kê khai . Một trong các lớp sẽ thực thi lớp con `ContentProvider`, đây chính là giao diện giữa provider của bạn và các ứng dụng khác. Mặc dù content provider đồng nghĩa với việc quản lý dữ liệu để những ứng dụng khác có thể truy cập được, nhưng hiển nhiên là ứng dụng của bạn cũng có các chức năng để người dùng truy vấn và chỉnh sửa dữ liệu được content provider của bạn quản lý.

Phần này sẽ giới thiệu các bước cơ bản để xây dựng content provider và liệt kê các API cần dùng.

Chuẩn bị trước khi xây dựng content provider

Trước khi bắt đầu xây dựng content provider, bạn thực hiện các bước sau:

1. **Quyết định xem liệu bạn có cần content provider hay không.** Bạn cần xây dựng một content provider nếu muốn cung cấp một hoặc nhiều tính năng sau:

   - Muốn cấp dữ liệu hoặc file phức tạp cho các ứng dụng khác.
   - Muốn cho phép người dùng sao chép dữ liệu phức tạp từ ứng dụng của bạn vào những ứng dụng khác.
   - Muốn cung cấp các gợi ý tìm kiếm tùy chỉnh bằng cách sử dụng framework tìm kiếm.

   Bạn *không* cần tới provider để dùng cơ sở dữ liệu SQLite, nếu việc sử dụng này hoàn toàn xảy ra trong ứng dụng của bạn.

2. Nếu vẫn chưa xác định được yếu tố trên, bạn có thể tìm hiểu thêm ở phần 10.1: "Cơ bản về content provider".

Tiếp theo, thực hiện các bước sau đây để xây dựng content provider:

1. Thiết kế không gian lưu trữ thô (raw storage) cho dữ liệu. Content provider cung cấp dữ liệu dưới hai dạng:

   Dữ liệu dưới dạng file

   Là những dữ liệu thường được tồn tại dưới dạng, chẳng hạn như ảnh, âm thanh hoặc video. Lưu các file này vào không gian lưu trữ riêng trong ứng dụng của bạn. Để phản hồi yêu cầu file từ ứng dụng khác, content provider bạn xây dựng có thể cung cấp việc xử lý file này.

   Dữ liệu "có cấu trúc" ("structured" data)

   Là những dữ liệu thường được đưa vào cơ sở dữ liệu, mảng hoặc cấu trúc tương tự. Lưu trữ dữ liệu này dưới dạng tương ứng với các bảng chứa hàng và cột. Mỗi hàng đại diện cho một thực thể (entity), chẳng hạn như một người hoặc một mục trong bản kiểm kê hàng hóa. Mỗi cột đại diện cho một số dữ liệu của thực thể này, chẳng hạn như tên người hoặc giá hàng hóa. Cách phổ biến để lưu kiểu dữ liệu này là trong một cơ sở dữ liệu SQLite, nhưng bạn có thể sử dụng bất cứ kiểu bộ nhớ lâu dài nào. Để tìm hiểu thêm về các kiểu bộ nhớ mà hệ thống Android cung cấp, xem mục "Thiết kế bộ nhớ lưu trữ dữ liệu".

2. Định nghĩa bản cài đặt cụ thể của lớp `ContentProvider` và các phương thức cần thiết của lớp này. Lớp `ContentProvider` là giao diện giữa dữ liệu của bạn và các phần còn lại trong hệ thống Android. Để tìm hiểu thêm về lớp này, tham khảo mục "Cài đặt lớp `ContentProvider`".

3. Định nghĩa chuỗi định danh (authority string) và content URI của provider, cùng với tên các cột. Nếu muốn ứng dụng của provider có thể xử lý được intent, bạn cũng phải định nghĩa những action của intent, dữ liệu bổ sung và các cờ. Đồng thời, bạn định nghĩa những quyền sẽ yêu cầu ứng dụng phải đăng ký khi muốn truy cập dữ liệu. Bạn nên tính đến việc định nghĩa tất cả các giá trị này dưới dạng hằng nằm trong một llớp contract riêng biệt; về sau, bạn có thể cung cấp lớp này cho các

nhà phát triển ứng dụng khác. Để tìm hiểu thêm về content URI, xem mục "Thiết kế content URI". Để tìm hiểu thêm về intent, xem mục "Intent và truy cập dữ liệu".

4. Thêm vào các tùy chọn khác, chẳng hạn như dữ liệu mẫu hoặc phần cài đặt của lớp `AbstractThreadedSyncAdapter` để có thể đồng bộ dữ liệu giữa content provider và dữ liệu trên đám mây (cloud-based data).

Thiết kế bộ lưu trữ dữ liệu

Content provider là giao diện để lưu dữ liệu dưới định dạng có cấu trúc. Trước khi tạo giao diện này, bạn phải quyết định cách lưu trữ dữ liệu. Bạn có thể lưu dữ liệu dưới bất kỳ dạng nào mình muốn, sau đó thiết kế giao diện để đọc và ghi dữ liệu, nếu cần.

Trong Android có sẵn một số kỹ thuật lưu trữ dữ liệu:

- Hệ thống Android chứa bộ API của cơ sở dữ liệu SQLite mà các content provider của chính Android dùng để lưu trữ dữ liệu dạng bảng. Lớp `SQLiteOpenHelper` giúp bạn tạo cơ sở dữ liệu, còn `SQLiteDatabase` là lớp cơ sở để truy cập cơ sở dữ liệu.

  Hãy nhớ rằng, bạn không nhất thiết phải sử dụng cơ sở dữ liệu để cài đặt kho lưu trữ dữ liệu. Một content provider được thể hiện ra bên ngoài dưới dạng tập bảng, tương tự như cơ sở dữ liệu quan hệ, nhưng bảng không phải là yêu cầu bắt buộc cho việc cài đặt bên trong của provider.
- Để lưu trữ dữ liệu dạng file, hệ điều hành Android có nhiều API hướng file (file-oriented data). Để tìm hiểu thêm về việc lưu trữ file, tham khảo mục "Lưu trữ dữ liệu". Nếu đang thiết kế content provider để chia sẻ dữ liệu liên quan đến đa phương tiện (media-related data) như âm nhạc hoặc video, bạn có thể xây dựng content provider kết hợp giữa dữ liệu bảng và file.
- Để làm việc với dữ liệu lưu trên mạng, sử dụng các lớp trong `java.net` và `android.net`. Bạn cũng có thể đồng bộ dữ liệu được lưu trên mạng với bộ lưu trữ dữ liệu cục bộ (local data store) như cơ sở dữ liệu, sau đó cung cấp dữ liệu dưới dạng bảng hoặc file. Ứng dụng mẫu Sample Sync Adapter (Ví dụ về adapter có chức năng đồng bộ) minh họa kiểu đồng bộ này.

Lời khuyên khi thiết kế dữ liệu

Sau đây là một số mẹo để thiết kế cấu trúc dữ liệu của provider:

- Dữ liệu bảng phải luôn có một cột "khóa chính" là trường mà content provider duy trì như một giá trị số duy nhất cho mỗi hàng. Bạn có thể dùng giá trị trên để liên kết hàng này với các hàng có liên quan trong bảng khác (lấy trường này làm "khóa ngoại" - "foreign key"). Mặc dù bạn có thể đặt một tên bất kỳ cho cột này, nhưng sử dụng tên `BaseColumns._ID` là lựa chọn tốt nhất, bởi vì việc liên kết kết quả của một truy vấn trên provider với `ListView` yêu cầu một trong các cột được truy xuất phải có tên là _ID.

- Nếu bạn muốn cung cấp ảnh bitmap hoặc các dữ liệu dung lượng lớn khác của dữ liệu dạng file, hãy lưu dữ liệu này trong một file, sau đó gián tiếp cung cấp chúng thay vì lưu trữ trực tiếp trong một bảng. Nếu làm theo cách này, bạn cần thông báo với người dùng content provider rằng họ phải sử dụng phương thức file `ContentResolver` để truy cập dữ liệu.
- Sử dụng kiểu dữ liệu BLOB để lưu dữ liệu có kích thước biến đổi hoặc có cấu trúc biến đổi. Ví dụ, bạn có thể sử dụng một cột BLOB để lưu JSON structure (cấu trúc JSON).

  Bạn cũng có thể sử dụng kiểu dữ liệu BLOB để cài đặt bảng *schema-independent (không phụ thuộc cấu trúc)*. Trong kiểu bảng này, bạn xác định một cột khóa chính, một cột có kiểu MIME và một hoặc nhiều cột thường kiểu BLOB. Ý nghĩa của dữ liệu trong các cột BLOB phụ thuộc vào giá trị trong cột kiểu MIME. Điều này cho phép bạn lưu các kiểu hàng khác nhau trong cùng một bảng. Bảng "dữ liệu" `ContactsContract.Data` của Provider Contacts là một ví dụ về bảng không phụ thuộc cấu trúc.

Thiết kế content URI

Content URI là một URI xác định dữ liệu trong content provider. Content URI bao gồm **chuỗi định danh** (tên có tính tượng trưng cho toàn bộ provider) và **một đường dẫn** (tên trỏ tới bảng hoặc file). Có thể có hoặc không có phần id trỏ tới một hàng cụ thể trong bảng. Mỗi phương thức truy cập dữ liệu của `ContentProvider` lấy content URI làm một đối số; việc này cho phép bạn xác định bảng, hàng hoặc file để truy cập.

Các khái niệm cơ bản của content URI được giới thiệu trong mục "Cơ bản về content provider".

Thiết kế chuỗi định danh

Mỗi content provider thường có chuỗi định danh riêng, đóng vai trò như tên sử dụng nội bộ trong Android. Để tránh xung đột với các content provider khác, bạn nên sử dụng quyền sở hữu miền Internet (Internet domain ownership) (theo thứ tự ngược lại) làm cơ sở cho chuỗi định danh của provider. Vì cách làm này cũng được áp dụng cho tên gói Android, nên bạn có thể định nghĩa chuỗi định danh của provider làm phần mở rộng của tên gói chứa trình này. Ví dụ, nếu gói Android có tên là `com.example.<tên ứng dụng>`, bạn nên để chuỗi định danh của provider là `com.example.<tên ứng dụng>.provider`.

Thiết kế cấu trúc đường dẫn

Các nhà phát triển thường tạo content URI từ chuỗi định danh bằng cách gắn thêm đường dẫn trỏ tới những bảng cụ thể. Chẳng hạn, nếu có hai bảng là *table1* và *table2*, bạn kết hợp với chuỗi định danh trong ví dụ trước để tạo thành content URI là `com.example.<tên ứng dụng>.provider/table1` và `com.example.<tên ứng dụng>.provider/table2`. Các đường dẫn này không hạn chế trong một phân đoạn (segment) đơn, không bắt buộc phải là bảng tương ứng với mỗi mức của đường dẫn.

Xử lý các ID của content URI

Theo quy ước, content provider cung cấp truy cập tới một hàng đơn trong bảng bằng cách chấp nhận một content URI với một giá trị ID tương ứng cho mỗi hàng nằm ở cuối URI. Cũng theo quy ước, content provider đối chiếu giá trị ID với cột `_ID` của bảng đó, đồng thời thực hiện truy cập được yêu cầu trên hàng tương ứng.

Các quy ước này tạo điều kiện thuận lợi cho kiểu mẫu thiết kế chung để truy cập vào content provider. Ứng dụng thực hiện truy vấn trên provider và hiển thị `Cursor` kết quả qua `ListView` bằng cách sử dụng `CursorAdapter`. Trong định nghĩa của lớp `CursorAdapter` yêu cầu một trong các cột của `Cursor` là `_ID`.

Sau đó, người dùng lấy ra một trong các hàng hiển thị từ giao diện người dùng theo thứ tự sắp xếp để xem hoặc chỉnh sửa dữ liệu. Ứng dụng lấy hàng tương ứng từ `Cursor` quay trở lại `ListView`, lấy ra giá trị `_ID` cho hàng đó, gắn giá trị này vào content URI, rồi gửi yêu cầu truy cập tới content provider. Từ đó, content provider có thể thực hiện truy vấn hoặc chỉnh sửa trên đúng hàng mà người dùng chọn.

Kiểu mẫu content URI

Để hỗ trợ bạn chọn action dành cho content URI tới, API của content provider chứa lớp tiện dụng `UriMatcher`, sẽ đối chiếu các "mẫu" content URI với giá trị số nguyên. Bạn có thể sử dụng những giá trị số nguyên này trong câu lệnh `switch` để chọn action mong muốn cho content URI hoặc các URI thỏa mãn một mẫu cụ thể.

Một mẫu content URI tương ứng với nhiều content URI bằng cách sử dụng các ký tự đại diện (wildcard character) sau:

- *: Tương ứng với một chuỗi có kích thước tùy ý chứa bất cứ ký tự nào hợp lệ.
- #: Tương ứng với một chuỗi có kích thước tùy ý chứa các ký tự số (numeric character).

Sau đây là ví dụ về thiết kế và viết mã xử lý content URI, sử dụng một Provider với chuỗi định danh `com.example.app.provider` nhận diện những content URI sau trỏ đến các bảng tương ứng:

- `content://com.example.app.provider/table1`: Một bảng có tên là `table1`.
- `content://com.example.app.provider/table2/dataset1`: Một bảng có tên là `dataset1`.
- `content://com.example.app.provider/table2/dataset2`: Một bảng có tên là `dataset2`.
- `content://com.example.app.provider/table3`: Một bảng có tên là `table3`.

Provider trên cũng nhận diện được các content URI này nếu chúng được gắn thêm ID hàng, chẳng hạn như `content://com.example.app.provider/table3/1` tương ứng với hàng 1 trong bảng `table3`.

Các mẫu content URI dưới đây có thể khả thi:

`content://com.example.app.provider/*:`

Tương ứng với bất kỳ content URI nào của provider.

`content://com.example.app.provider/table2/*:`

Tương ứng với content URI của các bảng `dataset1` và `dataset2`, nhưng không phù hợp với content URI của bảng `table1` hay `table3`.

`content://com.example.app.provider/table3/#`: Tương ứng với content URI của các hàng đơn trong bảng `table3`, chẳng hạn như `content://com.example.app.provider/table3/6` là hàng thứ `6`.

Đoạn mã nhỏ dưới đây mô tả cách các phương thức trong `UriMatcher` làm việc. Mã này sẽ xử lý các URI của toàn bộ bảng khác với các URI của một hàng đơn, sử dụng mẫu content URI `content://<chuỗi định danh>/<đường dẫn>` cho nhiều bảng và `content://<chuỗi định danh>/<đường dẫn>/<id>` cho từng hàng đơn.

Phương thức `addURI()` ánh xạ (map) giữa chuỗi định danh và đường dẫn với một giá trị số nguyên. Phương thức `match()` trả về giá trị số nguyên của URI. Câu lệnh switch chọn giữa truy vấn trên toàn bộ bảng và truy vấn trên bản ghi đơn:

```
public class ExampleProvider extends ContentProvider{
...
    // Tạo đối tượng UriMatcher.
    private static final UriMatcher sUriMatcher;
...
    /*
    * Đặt lời gọi phương thức addURI() ở đây, để xác định tất cả các
    * mẫu content URI mà provider sẽ nhận diện. Đối với đoạn mã
    * này, có thị lời gọi tới table (bảng) 3.
    */
...
    /*
    * Gán giá trị số nguyên 1 cho nhiều hàng của table 3. Lưu ý, đường
    * dẫn này không sử dụng ký tự thay thế
    */
    sUriMatcher.addURI("com.example.app.provider","table3",1);
     /*
    * Thiết lập mã cho một hàng đơn là 2. Trường hợp này có sử dụng ký
    * tự thay thế "#". Content URI  "content://com.example.app.provider/
    * table3/3" phù hợp, nhưng "content://com.example.app.provider/
    * table3" thì không.
    */
```

```
    sUriMatcher.addURI("com.example.app.provider","table3/#",2);
...
    // Cài đặt phương thức ContentProvider.query()
    publicCursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder){
...
        /*
        * Chọn bảng để truy vấn và thứ tự sắp xếp dựa trên mã mà URI
        trả về. Ở đây, chỉ giới thiệu các câu lệnh dành cho table 3.
        */
        switch(sUriMatcher.match(uri)){

            // Nếu URI tới là của toàn bộ table3
            case1:

                if(TextUtils.isEmpty(sortOrder)) sortOrder ="_ID ASC";
                break;

            // Nếu URI tới là của một hàng đơn
              case2:

                /*
                * Do đây là URI của một hàng đơn, nên phần giá trị _ID
                * tồn tại. Lấy phần cuối của đường dẫn từ URI; đây là
                * giá _ID. Sau đó, gắn giá trị với mệnh đề WHERE của
                * truy vấn
                */
                selection = selection + "_ID = " uri.getLastPathSegment();
                break;

            default:
            ...
                // Nếu không nhận diện được URI, bạn nên thực hiện một
                // số xử lý lỗi ở đây.
        }
        // gọi mã để thực hiện truy vấn
    }
```

Một lớp khác là `ContentUris` cung cấp các phương thức hữu dụng để làm việc với phần `id` của content URI. Các lớp `Uri` và `Uri.Builder` chứa những phương thức có thể dễ dàng phân tích đối tượng `Uri` hiện có và xây dựng các `Uri` mới.

Cài đặt lớp ContentProvider

Thể hiện của lớp `ContentProvider` quản lý việc truy cập tới bộ dữ liệu có cấu trúc bằng cách xử lý yêu cầu từ các ứng dụng khác. Cuối cùng thì tất cả các dạng truy cập đều gọi đến đối tượng `ContentResolver`, sau đó đối tượng này gọi tới một phương thức cụ thể của content provider để thực hiện truy cập.

Phương thức yêu cầu

Lớp trừu tượng (abstract class) `ContentProvider` định nghĩa sáu phương thức trừu tượng mà bạn phải cài đặt thành một phần trong các lớp con cụ thể của chính bạn. Tất cả các phương thức này, ngoại trừ phương thức `onCreate()`, được gọi bởi ứng dụng client đang truy cập vào content provider của bạn:

`query()`

Truy xuất dữ liệu từ content provider. Sử dụng các đối số để chọn bảng truy vấn, các hàng và cột trả về, cũng như thứ tự sắp xếp của kết quả. Trả về dữ liệu dưới dạng đối tượng `Cursor`.

`insert()`

Chèn một hàng mới vào provider. Sử dụng các đối số để chọn bảng đích và lấy giá trị cột để sử dụng. Trả về content URI của hàng mới chèn vào.

`update()`

Cập nhật các hàng đã có trong content provider. Sử dụng các đối số để chọn bảng cũng như hàng muốn cập nhật và lấy giá trị cột được cập nhật. Trả về số lượng hàng được cập nhật.

`delete()`

Xóa hàng trong content provider. Sử dụng các đối số để chọn bảng và hàng muốn xóa. Trả về số lượng hàng được xóa.

`getType()`

Trả về kiểu MIME tương ứng với content URI. Phương thức này được miêu tả chi tiết trong mục “Cài đặt kiểu MIME của content provider”.

`onCreate()`

Khởi tạo content provider. Hệ thống Android gọi phương thức này ngay sau khi Android tạo content provider của bạn. Lưu ý, content provider của bạn không được tạo ra cho đến khi đối tượng `ContentResolver` cố gắng truy cập vào trình này.

Lưu ý, các phương thức này có cùng chữ ký (signature) với những phương thức `ContentResolver` có tên tương tự.

Việc cài đặt các phương thức này nên tuân theo những phân tích dưới đây:

- Tất cả các phương thức, ngoại trừ `onCreate()`, có thể được nhiều luồng (thread) gọi cùng lúc, nên những phương thức này phải được cài đặt sao cho an toàn giữa nhiều luồng. Để tìm hiểu thêm về đa luồng, tham khảo mục “Processes and Threads” (“Tiến trình và luồng”).

- Tránh các thao tác dài tốn tài nguyên trong phương thức `onCreate()`. Không xử lý các tác vụ khởi tạo cho đến khi thực sự cần. Mục "Cài đặt phương thức `onCreate()`" bên dưới sẽ thảo luận chi tiết hơn về điều này.
- Mặc dù phải cài đặt các phương thức này, song bạn không cần viết mã để thực hiện thêm bất cứ điều gì, ngoại trừ trả về kiểu dữ liệu mong muốn. Ví dụ, có thể bạn muốn tránh không cho các ứng dụng khác chèn dữ liệu vào một số bảng. Để thực hiện điều này, bạn cần bỏ qua lời gọi hàm `insert()` và trả về 0.

Cài đặt phương thức query()

Phương thức `ContentProvider.query()` phải trả về đối tượng `Cursor`; hoặc nếu lỗi, phương thức này sẽ ném ra một ngoại lệ `Exception`. Nếu đang sử dụng cơ sở dữ liệu SQLite làm nơi lưu trữ dữ liệu, bạn chỉ cần trả về đối tượng `Cursor` từ một trong các phương thức `query()` của lớp `SQLiteDatabase`. Nếu truy vấn này không tìm được hàng phù hợp, bạn nên trả về một thể hiện của `Cursor` có phương thức `getCount()` trả về 0. Bạn chỉ nên trả về `null` nếu xảy ra lỗi bên trong trong quá trình thực hiện truy vấn.
Nếu bạn không sử dụng cơ sở dữ liệu SQLite làm nơi lưu trữ dữ liệu, hãy dùng một trong các lớp con cụ thể của `Cursor`. Ví dụ, lớp `MatrixCursor` cài đặt một con trỏ, trong đó mỗi hàng là một mảng của đối tượng `Object`. Với lớp này, hãy sử dụng phương thức `addRow()` để thêm một hàng mới.

Cần nhớ rằng, hệ thống Android phải có khả năng giao tiếp với ngoại lệ `Exception` thông qua các biên tiến trình. Android có thể làm điều này cho các ngoại lệ sau, cũng là những ngoại lệ rất có ích trong việc xử lý lỗi truy vấn:

- `IllegalArgumentException` (bạn có thể chọn để ném ra ngoại lệ này nếu provider của bạn nhận được content URI không hợp lệ).
- `NullPointerException`.

Cài đặt phương thức insert()

Phương thức `insert()` thêm một hàng mới vào bảng thích hợp, sử dụng các giá trị trong đối số `ContentValues`. Nếu tên cột không nằm trong đối số `ContentValues`, có thể bạn muốn cung cấp một giá trị mặc định cho phương thức này trong mã mã của content provider hoặc trong lược đồ (schema) cơ sở dữ liệu của bạn.

Phương thức này nên trả về content URI của hàng mới. Để thực hiện điều này, hãy thêm giá trị `_ID` của hàng mới (hoặc một khóa chính khác) vào content URI của bảng, sử dụng `withAppendedId()`.

Cài đặt phương thức delete()

Phương thức `delete()` không nhất thiết phải xóa hoàn toàn hàng về mặt vật lý khỏi nơi lưu trữ dữ liệu. Nếu đang sử dụng một adapter có chức năng đồng bộ (sync adapter) với content provider của mình, bạn phải tính đến việc tạo một cờ "xóa" cho hàng đã xóa thay vì xóa toàn bộ hàng đó. Adapter có chức năng đồng bộ có thể kiểm tra những hàng bị xóa và xóa chúng khỏi server trước khi xóa chúng khỏi content provider.

Cài đặt phương thức update()

Phương thức `update()` sử dụng các đối số `ContentValues` tương tự như phương thức `insert()`. Phương thức này cũng dùng các đối số `selection` và `selectionArgs` tương tự như phương thức `delete()` và `ContentProvider.query()`. Thông tin này cho phép bạn tái sử dụng mã giữa các phương thức.

Cài đặt phương thức onCreate()

Hệ thống Andorid gọi phương thức `onCreate()` khi khởi động content provider. Bạn chỉ nên thực hiện tác vụ khởi tạo nhanh trong phương thức này, đồng thời trì hoãn việc tạo cơ sở dữ liệu và nạp (load) dữ liệu cho đến khi provider thực sự nhận được yêu cầu về dữ liệu. Nếu thực hiện các tác vụ gây tốn thời gian trong phương thức `onCreate()`, bạn sẽ làm chậm việc khởi động của content provider; từ đó, làm chậm quá trình phản hồi của content provider với các ứng dụng khác.

Ví dụ, nếu sử dụng cơ sở dữ liệu SQLite, bạn có thể tạo một đối tượng `SQLiteOpenHelper` mới trong `ContentProvider.onCreate()`, sau đó tạo bảng SQL trong lần đầu mở cơ sở dữ liệu. Để thực hiện điều này theo cách đơn giản, trước tiên, bạn gọi phương thức `getWritableDatabase()`. Phương thức này sẽ tự động gọi phương thức `SQLiteOpenHelper.onCreate()`.

Hai đoạn mã nhỏ dưới đây minh họa sự tương tác giữa hai phương thức `ContentProvider.onCreate()` và `SQLiteOpenHelper.onCreate()`. Đoạn mã nhỏ đầu tiên cài đặt phương thức `ContentProvider.onCreate()`:

```
public class Example Provider extends ContentProvider

    /*
    * Định nghĩa một đối tượng hỗ trợ việc xử lý trên cơ sở dữ liệu.
    *  Lớp MainDatabaseHelper được định nghĩa trong đoạn mã nhỏ dưới đây.
    */
    private MainDatabaseHelper mOpenHelper;

    // Định nghĩa tên cơ sở dữ liệu
    private static final String DBNAME =”mydb”;

    // Biến chứa đối tượng cơ sở dữ liệu
    private SQLiteDatabase db;

    public boolean onCreate(){

        /*
        * Tạo một đối tượng trợ giúp mới. Phương thức này luôn được
        * xử lý nhanh. Lưu ý, cơ sở dữ liệu không được tạo và mở
        * cho đến khi phương thức SQLiteOpenHelper.getWritableDatabase
        * được gọi
        */
        mOpenHelper = new MainDatabaseHelper (
```

```
            getContext(),        // context của ứng dụng
            DBNAME,              // tên cơ sở dữ liệu
            null,                // sử dụng con trỏ SQLite mặc định
            1                    // số phiên bản (version number)
        );

        return true;
    }

    ...

    // Cài đặt phương thức chèn của content provider
    public Cursor insert(Uri uri,ContentValues values){
    // Chèn thêm mã vào đây để xác định bảng cần mở, xử lý việc kiểm
    // tra lỗi, và nhiều xử lý khác

        ...
         /*
        * Lấy cơ sở dữ liệu cho phép ghi. Lệnh này sẽ bắt đầu quá trình
        * tạo của cơ sở dữ liệu nếu cơ sở này chưa thực sự tồn tại.
        */
        db = mOpenHelper.getWritableDatabase();
    }
}
```

Đoạn mã nhỏ tiếp theo là phần cài đặt của phương thức [SQLiteOpenHelper.onCreate()](), bao gồm một lớp trợ giúp (helper class):

```
...
// Một chuỗi định nghĩa câu lệnh SQL để tạo bảng
private static final String SQL_CREATE_MAIN ="CREATE TABLE "+
    "main "+                        // Tên bảng
    "("+                            // Các cột của bảng
    " _ID INTEGER PRIMARY KEY, "+
    " WORD TEXT"
    " FREQUENCY INTEGER "+
    " LOCALE TEXT )";
...
/**
* Lớp trợ giúp thực sự tạo và quản lý kho lưu trữ dữ liệu bên dưới của
* content provider.
*/
protected static final class MainDatabaseHelper extends SQLiteOpenHelper{
```

```
    /*
    * Tạo một đối tượng hỗ trợ cho kho lưu trữ dữ liệu SQLite của
    * provider. Không tạo cơ sở dữ liệu và nâng cấp tại đây.
    */
    MainDatabaseHelper(Context context){
        super(context, DBNAME,null,1);
    }
     /*
    * Tạo kho lưu trữ dữ liệu. Phương thức này được gọi khi content
    * provider nội dung thử mở kho lưu trữ và SQLite báo cáo rằng kho
    * lưu trữ này chưa tồn tại.
    */
    public void onCreate(SQLiteDatabase db){
        // Tạo bảng chính
        db.execSQL(SQL_CREATE_MAIN);
    }
}
```

Cài đặt các kiểu MIME cho content provider

Lớp `ContentProvider` có hai phương thức trả về kiểu MIME:

`getType()`

Một trong các phương thức bắt buộc mà bạn phải cài đặt cho bất cứ content provider nào.

`getStreamTypes()`

Một phương thức bạn nên cài đặt nếu content provider của bạn cung cấp file.

Kiểu MIME của bảng

Phương thức `getType()` trả về một `String` trong định dạng kiểu MIME mô tả kiểu dữ liệu trả về bằng đối số content URI. Đối số Uri có thể là một mẫu chứ không phải một URI cụ thể nào; trong trường hợp này, bạn nên trả về kiểu dữ liệu có liên hệ với các content URI phù hợp với mẫu.

Đối với các kiểu dữ liệu phổ biến, chẳng hạn như văn bản (text), HTML hoặc JPEG, phương thức `getType()` sẽ trả về kiểu MIME chuẩn cho dữ liệu đó. Danh sách đầy đủ của các kiểu chuẩn này được liệt kê trên trang IANA MIME Media Types.

Với content URI trỏ tới một hoặc nhiều hàng của dữ liệu dạng bảng, `getType()` sẽ trả về một kiểu MIME theo kiểu MIME tùy chỉnh của Android:

- Phần type (kiểu): `vnd`
- Phần subtype (kiểu con):
  - Nếu mẫu URI cho một hàng đơn là: `android.cursor.item/`
  - Nếu mẫu URI cho nhiều hơn một hàng: `android.cursor.dir/`
- Phần xác định provider: `vnd.<tên>.<kiểu>`

  Bạn cung cấp `<tên>` và `<kiểu>`. Giá trị `<tên>` nên là toàn cục duy nhất, còn `<kiểu>` nên có giá trị duy nhất tương ứng với mẫu URI. Một lựa chọn tốt cho `<tên>` là tên công ty bạn hoặc một số phần trong tên gói Android của ứng dụng mà bạn phát triển. Một lựa chọn tốt cho `<kiểu>` là một chuỗi xác định bảng có liên hệ với URI.

Ví dụ, nếu chuỗi định danh của provider là `com.example.app.provider` và chứa một bảng có tên `table1`, kiểu MIME cho nhiều hàng trong `table1` là:

```
vnd.android.cursor.dir/vnd.com.example.provider.table1
```

Kiểu MIME cho một hàng đơn của bảng `table1` là:

```
vnd.android.cursor.item/vnd.com.example.provider.table1
```

Kiểu MIME cho file

Nếu provider của bạn cung cấp file, hãy cài đặt phương thức `getStreamTypes()`. Phương thức này trả về một mảng `String` bao gồm các kiểu MIME cho những file mà provider có thể trả về khi nhận được một content URI xác định. Bạn nên lọc các kiểu MIME bạn cung cấp bằng đối số bộ lọc kiểu MIME; từ đó, bạn chỉ cần trả về những kiểu MIME mà client muốn xử lý.

Ví dụ, giả sử một provider cung cấp các ảnh chụp có định dạng `.jpg`, `.png` và `.gif`. Nếu một ứng dụng gọi phương thức `ContentResolver.getStreamTypes()` với chuỗi lọc `image/*` (lọc lấy bất kỳ thứ gì là "ảnh"), thì phương thức `ContentProvider.getStreamTypes()` sẽ trả về mảng:

```
{"image/jpeg","image/png","image/gif"}
```

Nếu ứng dụng này chỉ quan tâm tới các file .jpg, nó có thể gọi phương thức `ContentResolver.getStreamTypes()` với chuỗi lọc `*\/jpeg`, còn phương thức `ContentProvider.getStreamTypes()` sẽ trả về:

```
{"image/jpeg"}
```

Nếu provider bạn phát triển không chia sẻ kiểu MIME nào do chuỗi lọc yêu cầu, phương thức `getStreamTypes()` nên trả về `null`.

Cài đặt lớp contract

Lớp contract là lớp được khai báo `public final` chứa các định nghĩa hằng cho URI, tên cột, kiểu MIME và siêu dữ liệu (meta-data) khác gắn liền với provider. Lớp này thiết lập một giao kết (contract) giữa provider và các ứng dụng khác bằng cách đảm bảo rằng provider có thể được truy cập đúng, ngay cả khi các giá trị thực sự của URI, tên cột,… thay đổi.

Lớp contract cũng giúp ích cho nhà phát triển ứng dụng, bởi chúng thường có các tên hằng dễ nhớ; do đó, nhà phát triển ít dùng tên cột hoặc URI không chính xác. Vì là một lớp, nên Lớp contract có thể cung cấp tài liệu Javadoc. Các môi trường phát triển tích hợp (*integrated development environment - IDE*) như Eclipse có thể đưa ra danh sách gợi ý cho phép người dùng chọn để tự động điền các tên hằng này từ Lớp contract và hiển thị Javadoc của hằng đó.

Nhà phát triển ứng dụng không cần truy cập vào file lớp của Lớp contract từ ứng dụng của bạn, nhưng họ có thể biên dịch tĩnh lớp này trong ứng dụng của mình từ file .jar bạn cung cấp.

Lớp `ContactsContract` và các lớp lồng (nested class) của nó là những ví dụ về Lớp contract.

Cài đặt quyền của content provider

Quyền và truy cập về mọi phương diện trong hệ thống Android được mô tả chi tiết trong chủ đề "Security and Permissions" ("Quyền và bảo mật"). Chủ đề "Data Storage" ("Bộ lưu trữ dữ liệu") cũng đã mô tả quyền và bảo mật có tác dụng với nhiều kiểu bộ nhớ khác nhau. Tóm lại, bạn cần lưu ý một số điểm quan trọng sau:

- Mặc định, file dữ liệu được lưu tại bộ nhớ trong của thiết bị là của riêng ứng dụng và provider do bạn phát triển.
- Cơ sở dữ liệu `SQLiteDatabase` bạn tạo là của riêng ứng dụng và provider do bạn phát triển.
- Mặc định, các file dữ liệu bạn lưu tại bộ nhớ ngoài là *công cộng (public) và bên ngoài có thể đọc được (world-readable)*. Bạn không thể dùng content provider để giới hạn truy cập tới các file lưu trong bộ nhớ ngoài, bởi các ứng dụng khác có thể dùng API khác để đọc và ghi chúng.
- Phương thức gọi để mở hay tạo file hoặc cơ sở dữ liệu SQLite trên bộ nhớ trong của thiết bị có thể trao cả quyền đọc lẫn ghi cho tất cả các ứng dụng khác. Nếu bạn sử dụng một file hoặc cơ sở dữ liệu bên trong làm nơi lưu trữ dữ liệu cho mình, đồng thời trao quyền truy cập "bên ngoài có thể đọc được" ("world-readable") và "bên ngoài có thể ghi được" ("world-writeable"), thì các quyền bạn thiết lập cho provider của mình trong file kê khai sẽ không còn tác dụng bảo vệ dữ liệu của bạn nữa. Quyền truy cập mặc định của file và cơ sở dữ liệu tại bộ nhớ trong là "riêng tư" ("private") và bạn không nên thay đổi quyền mặc định này đối với kho lưu trữ dữ liệu provider của mình.

Nếu muốn dùng quyền content provider để điều khiển truy cập tới dữ liệu của mình, bạn nên lưu dữ liệu trong các file bên trong, cơ sở dữ liệu SQLite hoặc "đám mây" (ví

dụ như trên một server ở xa), đồng thời nên giữ cho các file và cơ sở dữ liệu là dành riêng cho ứng dụng của bạn.

Cài đặt quyền

Mọi ứng dụng đều có thể đọc hoặc ghi vào provider của bạn, ngay cả khi dữ liệu bên dưới là riêng tư, bởi theo mặc định thì provider của bạn không có tập các quyền truy cập. Để thay đổi điều này, hãy thiết lập quyền cho provider của bạn trong file kê khai, sử dụng thuộc tính hoặc phần tử con của phần tử `<provider>`. Bạn có thể thiết lập quyền để áp dụng cho toàn bộ provider, cho bảng cụ thể, hay thậm chí cho một số bản ghi nhất định, hoặc cả ba.

Bạn định nghĩa các quyền cho content provider đang phát triển với một hoặc nhiều phần tử `<permission>` trong file kê khai. Để đảm bảo những quyền này là duy nhất đối với ứng dụng của bạn, sử dụng kiểu phạm vi theo định dạng Java cho thuộc tính `android:name`. Ví dụ, tên của quyền đọc là `com.example.app.provider.permission.READ_PROVIDER`.

Danh sách dưới đây mô tả phạm vi của các quyền của provider, bắt đầu với quyền áp dụng cho toàn bộ provider, sau đó đi vào các quyền chi tiết hơn. Quyền càng chi tiết càng có độ ưu tiên cao hơn quyền có phạm vi cao hơn:

Quyền đọc-ghi kết hợp mức provider

> Một quyền kiểm soát cả quyền đọc lẫn quyền ghi trên toàn provider, được xác định với thuộc tính `android:permission` của phần tử `<provider>`.

Quyền đọc và ghi riêng mức provider

> Một quyền đọc và một quyền ghi cho toàn provider. Bạn xác định những quyền này bằng các thuộc tính `android:readPermission` và `android:writePermission` của phần tử `<provider>`. Các quyền này có ưu tiên hơn quyền được thuộc tính `android:permission` xác định.

Quyền tại mức đường dẫn (path-level)

> Quyền đọc, ghi hoặc đọc/ghi cho một content URI trong provider mà bạn phát triển. Bạn chỉ rõ mỗi URI mình muốn quản lý với một phần tử con `<path-permission>` của phần tử `<provider>`. Với mỗi content URI bạn chỉ ra, bạn có thể xác định quyền đọc/ghi, một quyền đọc, hay một quyền ghi, hoặc cả ba. Quyền đọc và ghi này có quyền ưu tiên hơn quyền đọc/ghi. Quyền tại mức đường dẫn cũng có độ ưu tiên hơn quyền mức provider.

Quyền tạm (temporary permission)

> Một mức quyền mà sẽ trao quyền truy cập tạm thời cho một ứng dụng, ngay cả khi ứng dụng không yêu cầu các quyền thông thường. Đặc tính của quyền tạm là làm giảm số quyền mà một ứng dụng phải yêu cầu trong file kê khai của nó. Khi bạn bật quyền tạm, chỉ những ứng dụng cần quyền "lâu dài" trên provider mới có thể liên tục truy cập toàn bộ dữ liệu của bạn.
>
> Giả sử có các quyền bạn cần để cài đặt một ứng dụng và email provider (trình cung cấp e-mail), khi muốn cho phép ứng dụng hiển thị ảnh bên ngoài (outside image viewer) hiển thị các ảnh đính kèm trong provider. Để cấp cho ứng dụng hiển thị ảnh

các quyền truy cập cần thiết mà không phải yêu cầu quyền, hãy thiết lập quyền tạm cho các content URI của ảnh. Thiết kế ứng dụng e-mail của bạn để khi người dùng muốn hiển thị ảnh, ứng dụng sẽ gửi một intent chứa content URI của ảnh và các cờ quyền tới ứng dụng hiển thị ảnh. Sau đó, ứng dụng hiển thị ảnh có thể truy vấn email provider bạn phát triển để lấy ảnh ra, ngay cả khi ứng dụng hiển thị không có quyền đọc thông thường trên provider đó.

Để bật quyền tạm, hoặc là thiết lập thuộc tính `android:grantUriPermissions` của phần tử `<provider>`, hoặc thêm một hoặc nhiều phần tử con `<grant-uri-permission>` vào phần tử `<provider>`. Nếu dùng quyền tạm, bạn phải gọi phương thức `Context.revokeUriPermission()` mỗi khi xóa sự hỗ trợ trên một content URI khỏi provider và content URI được gắn với quyền tạm.

Giá trị của thuộc tính xác định mức độ truy cập được của provider do bạn phát triển. Nếu thuộc tính này được thiết lập là true, hệ thống của bạn sẽ gán quyền tạm cho toàn bộ provider, che khuất các quyền được yêu cầu tại mức đường dẫn và mức provider.

Nếu cờ này được thiết lập là false, bạn phải thêm phần tử con `<grant-uri-permission>` cho phần tử `<provider>`. Mỗi phần tử con xác định content URI hoặc các URI được gán quyền truy cập tạm.

Để gán quyền tạm cho một ứng dụng, một intent phải chứa cờ `FLAG_GRANT_READ_URI_PERMISSION` hoặc `FLAG_GRANT_WRITE_URI_PERMISSION`, hay cả hai. Những quyền này được thiết lập bằng phương thức setFlags().

Nếu không có thuộc tính `android:grantUriPermissions`, thuộc tính này được coi là false.

Phần tử <provider>

Tương tự phần tử `Activity` và `Service`, lớp con của `ContentProvider` phải được định nghĩa trong file kê khai của ứng dụng provider, bằng cách sử dụng phần tử `<provider>`. Hệ điều hành Android lấy các thông tin sau từ phần tử này:

Chuỗi định danh (`android:authorities`)

Tên có tính biểu tượng xác định toàn bộ provider bên trong hệ thống. Thuộc tính này được mô tả chi tiết ở mục "Thiết kế Content URI".

Tên lớp provider (`android:name`)

Là lớp cài đặt `ContentProvider`. Lớp này được mô tả chi tiết ở mục "Cài đặt lớp ContentProvider".

Quyền

Các thuộc tính xác định quyền mà những ứng dụng khác phải có để truy cập vào dữ liệu của provider:

- `android:grantUriPermssions:` Cờ quyền tạm.
- `android:permission`: Quyền đọc ghi kết hợp trên toàn bộ provider.

- `android:readPermission`: Quyền đọc trên toàn bộ provider.
- `android:writePermission`: Quyền ghi trên toàn bộ provider.

Quyền và các thuộc tính tương ứng được mô tả chi tiết hơn trong mục "Cài đặt quyền của content provider".

Các thuộc tính điều khiển và khởi động

Các thuộc tính này xác định cách thức và thời điểm hệ thống Android khởi động provider, những đặc tính xử lý của provider và các thiết lập trong thời gian chạy khác:

- `android:enabled`: Cờ cho phép hệ thống khởi động provider.
- `android:exported`: Cờ cho phép các ứng dụng khác sử dụng provider này.
- `android:initOrder`: Thứ tự khởi động của provider, liên quan đến các provider khác trong cùng tiến trình.
- `android:multiProcess`: Cờ cho phép hệ thống khởi động provider trong cùng tiến trình với client gọi.
- `android:process`: Tên của tiến trình mà tại đó, provider nên chạy.
- `android:syncable`: Cờ cho biết dữ liệu provider được đồng bộ với dữ liệu trên server hay không.

Các thuộc tính này được giới thiệu đầy đủ trong chủ đề hướng dẫn lập trình của phần tử `<provider>`.

Các thuộc tính thông tin

Nhãn và biểu tượng tùy chọn cho provider:

- `android:icon`: Tài nguyên drawable chứa một biểu tượng của provider. Biểu tượng này xuất hiện bên cạnh nhãn của provider trong danh sách các ứng dụng nằm ở *Settings* > *Apps* > *All*.
- `android:label`: Nhãn thông tin mô tả provider hay dữ liệu của provider này, hoặc cả hai. Nhãn này xuất hiện trong danh sách các ứng dụng nằm ở *Settings* > *Apps* > *All*.

Các thuộc tính này được giới thiệu đầy đủ trong phần hướng dẫn lập trình của phần tử `<provider>`.

Intent và truy cập dữ liệu

Các ứng dụng có thể gián tiếp truy cập content provider thông qua `Intent`. Ứng dụng không gọi phương thức bất cứ nào của `ContentResolver` hay `ContentProvider`. Thay vào đó, nó gửi một intent để khởi động một activity, đây là activity thường nằm trong chính ứng dụng của provider này. Activity đích chịu trách nhiệm truy xuất và hiển thị dữ liệu trong giao diện người dùng của activity này. Tùy vào hoạt động của intent, activity đích cũng có thể đưa ra nhắc nhở người dùng thực hiện thay đổi trên dữ liệu của provider. Một intent cũng có thể chứa dữ liệu "bổ sung" mà activity đích hiển thị trên giao diện người dùng; sau đó, người dùng có thể chọn thay đổi dữ liệu này trước khi sử dụng dữ liệu "bổ sung" để chỉnh sửa dữ liệu trong content provider.

Có thể bạn muốn sử dụng việc truy cập thông qua intent để đảm bảo tính toàn vẹn (integrity) của dữ liệu. Content provider của bạn có thể phụ thuộc vào việc dữ liệu được chèn thêm, cập nhật và xóa theo các lôgíc nghiệp vụ được định nghĩa chặt chẽ. Nếu gặp trường hợp như vậy, hãy cho phép ứng dụng khác được sửa trực tiếp trên dữ liệu của bạn, điều này có thể dẫn đến dữ liệu không hợp lệ. Nếu muốn các nhà phát triển sử dụng truy cập thông qua intent, bạn cần đảm bảo rằng việc này phải được hướng dẫn một cách rõ ràng. Hãy giải thích với các nhà phát triển ứng dụng rằng tại sao việc truy cập thông qua intent bằng cách dùng giao diện người dùng của ứng dụng bạn phát triển lại tốt hơn việc cố gắng chỉnh sửa dữ liệu bằng mã do họ phát triển.

Việc xử lý các intent đến, tức intent muốn chỉnh sửa dữ liệu của provider do bạn phát triển, không khác biệt so với việc xử lý những intent khác. Bạn có thể tìm hiểu thêm về việc sử dụng intent qua chủ đề "Intents and Intent Filters" ("Intent và bộ lọc intent").

10.3 Provider Calendar

Provider Calendar là kho lưu trữ các sự kiện lập lịch của người dùng. API của Provider Calendar cho phép bạn thực hiện các hoạt động truy vấn, chèn, cập nhật và xóa trên lịch ngày tháng (calendar), sự kiện (event), thành phần tham dự (attendee), lời nhắc nhở (reminder),…

API của Provider Calendar có thể được các ứng dụng và adapter có chức năng đồng bộ (sync adapter) sử dụng. Các quy tắc này thay đổi dựa vào loại chương trình thực hiện lời gọi. Mục này tập trung chủ yếu vào cách thức một ứng dụng dùng API của Provider Calendar. Phần thảo luận về cách sử dụng của adapter có chức năng đồng bộ sẽ được giới thiệu trong mục "Adapter có chức năng đồng bộ".

Thông thường, để đọc hoặc ghi dữ liệu lịch ngày tháng, file kê khai của ứng dụng phải có các quyền hợp lệ, như mô tả ở mục "Quyền người dùng". Để hỗ trợ việc thực hiện các thao tác được dễ dàng, Provider Calendar cung cấp tập các intent, như mô tả trong mục "Các intent của Calendar". Thông qua ứng dụng Calendar, các intent này giúp người dùng chèn, hiển thị và chỉnh sửa sự kiện. Người dùng tương tác với ứng dụng Calendar, sau đó trả về ứng dụng ban đầu. Do đó, ứng dụng của bạn không cần yêu cầu quyền, hoặc cần cung cấp thêm một giao diện người dùng để hiển thị hoặc tạo sự kiện nữa.

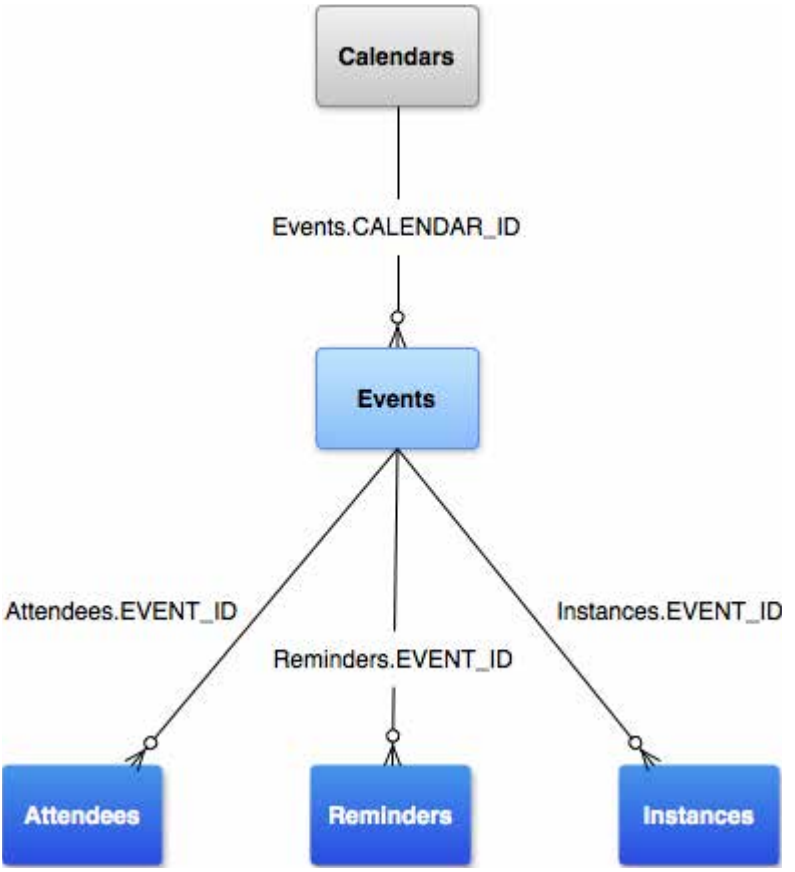
Các khái niệm cơ bản

Content provider lưu trữ dữ liệu và giúp các ứng dụng khác có thể truy cập được vào những dữ liệu này. Các content provider được nền tảng Android cung cấp (bao gồm Provider Calendar) thường xuất dữ liệu dưới dạng tập bảng dựa trên mô hình cơ sở dữ liệu quan hệ; trong đó, mỗi hàng là một bản ghi và mỗi cột là dữ liệu của một kiểu và ý nghĩa cụ thể. Thông qua API của Provider Calendar, ứng dụng và adapter có chức năng đồng bộ có thể đọc/ghi trên các bảng cơ sở dữ liệu lưu trữ dữ liệu lập lịch của người dùng.

Mỗi content provider cung cấp ra bên ngoài một URI công khai (nằm trong một đối tượng `Uri`), đây là định danh duy nhất cho mỗi tập dữ liệu của provider. Content provider quản lý nhiều tập dữ liệu (nhiều bảng) xuất ra một URI cho mỗi tập. Tất cả các URI của provider bắt đầu bằng chuỗi "`content://`". Định dạng này định danh

dữ liệu do provider quản lý. Provider Calendar định nghĩa hằng cho URI của mỗi lớp (bảng) trong trình này. Các URI này có định dạng là `<class>.CONTENT_URI`. Ví dụ, `Events.CONTENT_URI`.

Hình 1 minh họa một mô hình dữ liệu của Provider Calendar. Mô hình này chỉ biểu diễn các bảng chính và những trường có chức năng liên kết bảng.



Calendars	Lịch ngày tháng
Events	Sự kiện
Attendee	Thành phần tham dự
Reminder	Lời nhắc nhở
Instances	Thể hiện

Hình 1. Mô hình dữ liệu của Provider Calendar.

Một người dùng có thể có nhiều lịch (Calendar), các lịch khác nhau tương ứng với những loại tài khoản (account) khác nhau (Lịch trên Google, Exchange,…).

`CalendarContract` định nghĩa mô hình dữ liệu của lịch và các sự kiện chứa thông tin. Dữ liệu này được lưu vào một số bảng, như sau:

Bảng (Lớp)	Mô tả
CalendarContract.Calendars	Bảng này lưu thông tin về các lịch cụ thể. Mỗi hàng của bảng chứa thông tin chi tiết về từng lịch, chẳng hạn như tên, màu sắc, thông tin đồng bộ,…
CalendarContract.Events	Bảng này lưu thông tin về các sự kiện cụ thể. Mỗi hàng của bảng chứa thông tin về một sự kiện - ví dụ như tiêu đề sự kiện, vị trí, thời điểm bắt đầu, kết thúc,… Sự kiện có thể xảy ra một lần hoặc lặp lại nhiều lần. Thành phần tham dự, lời nhắc nhở và các thuộc tính mở rộng được lưu trong từng bảng riêng. Mỗi bảng đều có một trường EVENT_ID tham chiếu tới trường _ID của bảng Events.
CalendarContract.Instances	Bảng này lưu thời điểm bắt đầu và kết thúc của mỗi lần xảy ra sự kiện. Mỗi hàng của bảng đại diện cho một lần sự kiện xảy ra. Đối với các sự kiện chỉ xảy ra một lần, thể hiện (instance) và sự kiện là ánh xạ 1:1. Với các sự kiện xảy ra nhiều lần, nhiều hàng được tự động sinh ra tương ứng với nhiều lần xuất hiện của sự kiện đó.
CalendarContract.Attendees	Bảng này quản lý thông tin về thành phần tham dự (khách mời). Mỗi hàng đại diện cho một khách mời của một sự kiện. Bảng này xác định loại khách mời và phản hồi khả năng tham dự của khách mời đối với mỗi sự kiện.
CalendarContract.Reminders	Bảng này quản lý dữ liệu thông báo/nhắc nhở (alert/notification data). Mỗi hàng đại diện cho một lời nhắc nhở của một sự kiện. Mỗi sự kiện có thể có nhiều lời nhắc. Số lần nhắc tối đa cho mỗi sự kiện được xác định trong biến MAX_REMINDERS, được thiết lập bởi adapter có chức năng đồng bộ quản lý lịch đã xác định. Các lời nhắc được xác định theo số phút trước mỗi sự kiện và có một phương thức để xác định người dùng nào sẽ được nhắc.

API của Provider Calendar được thiết kế linh hoạt và mạnh mẽ. Cùng lúc, trình này có thể cung cấp những trải nghiệm tốt cho người dùng cuối cũng như đảm bảo tính toàn vẹn của lịch và dữ liệu bên trong. Phần cuối sẽ tóm tắt một số lưu ý khi sử dụng API này:

- **Chèn, cập nhật và hiển thị các sự kiện của lịch**. Để trực tiếp chèn, chỉnh sửa và đọc sự kiện trên Provider Calendar, bạn cần các quyền thích hợp. Tuy nhiên, nếu không cần xây dựng một ứng dụng lịch chính thức hoặc dùng đến adapter có chức năng đồng bộ, thì bạn không phải yêu cầu những quyền này. Thay vào đó, bạn có thể dùng các intent được ứng dụng Calendar của Android hỗ trợ để thực hiện các hoạt động đọc và ghi trên ứng dụng. Khi dùng intent, ứng dụng của bạn gửi người dùng tới ứng dụng Calendar để thực hiện các thao tác cần thiết dưới dạng form chứa dữ liệu điền sẵn. Sau khi các thao tác này được thực hiện, chúng được trả về ứng dụng mà bạn phát triển. Bằng cách thiết kế ứng dụng để thực hiện các thao tác chung thông qua Calendar, bạn có thể cung cấp giao diện người dùng thích hợp và thiết thực cho người dùng. Đây là cách tiếp cận được khuyến nghị. Để tham khảo thêm thông tin, xem mục "Các intent của Calendar".
- **Adapter có chức năng đồng bộ**. Adapter có chức năng đồng bộ tiến hành đồng bộ dữ liệu của lịch trên thiết bị người dùng với server hoặc nguồn dữ liệu khác. Bảng `CalendarContract.Calendars` và `CalendarContract.Events` chứa các cột để dành cho adapter có chức năng đồng bộ sử dụng. Provider này và các ứng dụng sẽ không chỉnh sửa những cột trên. Thực tế, đó cũng là các cột không được hiển thị, trừ phi chúng được adapter có chức năng đồng bộ truy cập. Để tìm hiểu thêm về adapter có chức năng đồng bộ, xem mục "Adapter có chức năng đồng bộ".

Quyền người dùng

Để đọc dữ liệu lịch, ứng dụng phải có quyền `READ_CALENDAR` trong file kê khai của nó. Ngoài ra, để xóa, chèn hoặc cập nhật dữ liệu của lịch, ứng dụng phải có quyền `WRITE_CALENDAR`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"...>
    <uses-sdkandroid:minSdkVersion="14"/>
    <uses-permissionandroid:name="android.permission.READ_CALENDAR"/>
    <uses-permissionandroid:name="android.permission.WRITE_CALENDAR"/>
    ...
</manifest>
```

Bảng Calendars

Bảng `CalendarContract.Calendars` chứa thông tin chi tiết về các lịch cá nhân. Các cột của bảng Calendars dưới đây cung cấp quyền ghi cho ứng dụng hoặc adapter có chức năng đồng bộ. Để xem danh sách đầy đủ các trường được hỗ trợ, tham khảo `CalendarContract.Calendars`.

Hằng	Mô tả
NAME	Tên lịch.
CALENDAR_DISPLAY_NAME	Tên lịch sẽ được hiển thị cho người dùng.
VISIBLE	Một giá trị kiểu boolean cho biết lịch có được chọn để hiển thị hay không. Giá trị 0 có nghĩa là các sự kiện tương ứng với lịch này không được hiển thị. Giá trị 1 cho biết các sự kiện tương ứng với lịch này được hiển thị. Đây là giá trị có tác động lên việc tạo các hàng trong bảng CalendarContract.Instances.
SYNC_EVENTS	Một giá trị kiểu boolean cho biết xem lịch có được đồng bộ và có các sự kiện được lưu trên thiết bị hay không. Giá trị 0 cho biết không đồng bộ lịch này, hoặc không lưu sự kiện của lịch trên thiết bị. Giá trị 1 cho biết có đồng bộ sự kiện của lịch này và lưu sự kiện của lịch trên thiết bị.

Truy vấn lịch

Sau đây là một ví dụ minh họa cách lấy lịch do một người dùng cụ thể quản lý. Để đơn giản, trong ví dụ này, hoạt động truy vấn được hiển thị trong luồng giao diện người dùng ("luồng chính" - "main thread"). Trong thực tế, các truy vấn này nên được thực hiện trong luồng không đồng bộ (asynchronous thread) thay vì trên luồng chính. Để thảo luận thêm về điều này, xem mục Loaders (Các trình loader). Nếu bạn chỉ đọc dữ liệu mà không chỉnh sửa, xem phần AsyncQueryHandler.

```
// Mảng đối số projection.  Tạo chỉ số cho mảng này thay vì thực hiện
// tìm kiếm động (dynamic lookup) để nâng cao hiệu suất.
public static final String [] EVENT_PROJECTION = newString[]{
    Calendars._ID,                           // 0
    Calendars.ACCOUNT_NAME,                  // 1
    Calendars.CALENDAR_DISPLAY_NAME,         // 2
    Calendars.OWNER_ACCOUNT                  // 3
};

// Chỉ số của mảng đối số projection bên trên.

privatestaticfinalint PROJECTION_ID_INDEX =0;
privatestaticfinalint PROJECTION_ACCOUNT_NAME_INDEX =1;
privatestaticfinalint PROJECTION_DISPLAY_NAME_INDEX =2;
privatestaticfinalint PROJECTION_OWNER_ACCOUNT_INDEX =3;
```

Tại sao phải đưa thêm trường ACCOUNT_TYPE vào mệnh đề chọn?

Nếu truy vấn trên Calendars.ACCOUNT_NAME, bạn phải đưa thêm trường Calendars.ACCOUNT_TYPE vào mệnh đề chọn. Đó là vì một tài khoản chỉ được xác định là duy nhất khi xét cả ACCOUNT_NAME lẫn ACCOUNT_TYPE.ACCOUNT_TYPE là một chuỗi tương ứng với trình xác thực tài khoản (account authenticator) được sử dụng khi đăng ký tài khoản với AccountManager (trình quản lý tài khoản). Một kiểu tài khoản đặc biệt gọi là ACCOUNT_TYPE_LOCAL của lịch không tương ứng với tài khoản thiết bị. Các tài khoản ACCOUNT_TYPE_LOCAL không được đồng bộ.

Ở phần tiếp theo của ví dụ, bạn sẽ tạo truy vấn của mình. Mệnh đề chọn xác định điều kiện thực hiện truy vấn. Trong ví dụ này, truy vấn tìm kiếm lịch có ACCOUNT_NAME là "sampleuser@google.com", ACCOUNT_TYPE là "com.google", và OWNER_ACCOUNT là sampleuser@google.com. Nếu bạn muốn xem toàn bộ lịch mà người dùng đã xem chứ không chỉ là các lịch của người dùng đó, hãy bỏ qua OWNER_ACCOUNT. Truy vấn trả về đối tượng Cursor mà bạn có thể dùng để duyệt qua tập kết quả mà truy vấn cơ sở dữ liệu trả về. Để xem thêm phần thảo luận về sử dụng truy vấn trong content provider, tham khảo "Content Providers" ("Content provider").

```
// Chạy truy vấn
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection ="((“+Calendars.ACCOUNT_NAME +” = ?) AND (“
                        +Calendars.ACCOUNT_TYPE +” = ?) AND (“
                        +Calendars.OWNER_ACCOUNT +” = ?))”;
String[] selectionArgs = new String[] {“sampleuser@gmail.com”,”com.google”,
        “sampleuser@gmail.com”};
// Gửi truy vấn và nhận đối tượng Cursor trả về.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs,null);
```

Phần tiếp theo sẽ dùng con trỏ để duyệt qua tập kết quả. Phần này sử dụng các hằng thiết lập ở đầu ví dụ để trả về giá trị của mỗi trường.

```
// Sử dụng con trỏ để duyệt qua các bản ghi kết quả được trả lại
while (cur.moveToNext()) {
    long calID =0;
    String displayName = null;
    String accountName = null;
    String ownerName = null;

    // Lấy ra giá trị của các trường
    calID = cur.getLong(PROJECTION_ID_INDEX);
    displayName = cur.getString(PROJECTION_DISPLAY_NAME_INDEX);
    accountName = cur.getString(PROJECTION_ACCOUNT_NAME_INDEX);
```

```
    ownerName = cur.getString(PROJECTION_OWNER_ACCOUNT_INDEX);
        // Thực hiện một số thao tác với các giá trị này...

    ...
}
```

Chỉnh sửa lịch

Để thực hiện cập nhật trên lịch, bạn có thể cung cấp _ID của lịch bằng cách gắn thêm ID vào Uri (với phương thức withAppendedId()), hoặc lấy mục chọn đầu tiên. Mệnh đề chọn sẽ bắt đầu với "_id=?", và selectionArg (đối số chọn) đầu tiên sẽ là _ID của lịch. Bạn cũng có thể cập nhật bằng cách mã hóa (encode) ID trong URI. Ví dụ này sẽ thay đổi tên hiển thị của lịch, sử dụng phương thức withAppendedId():

```
private static final String DEBUG_TAG ="MyActivity";
...
long calID =2;
ContentValues values = new ContentValues();
// Tên hiển thị mới của lịch
values.put(Calendars.CALENDAR_DISPLAY_NAME,"Trevor's Calendar");
Uri updateUri =ContentUris.withAppendedId(Calendars.CONTENT_URI, calID);
int rows = getContentResolver().update(updateUri, values,null,null);
Log.i(DEBUG_TAG,"Rows updated: "+ rows);
```

Chèn lịch

Lịch được thiết kế để được quản lý chủ yếu bởi adapter có chức năng đồng bộ. Do đó, bạn chỉ nên thêm các lịch mới dưới dạng adapter có chức năng đồng bộ. Điểm quan trọng nhất là ứng dụng chỉ có thể thực hiện các thay đổi bề ngoài, chẳng hạn như thay đổi tên hiển thị. Nếu một ứng dụng cần tạo một lịch cục bộ (local calendar), bạn có thể chèn lịch dưới dạng một adapter đồng bộ, sử dụng ACCOUNT_TYPE là ACCOUNT_TYPE_LOCAL.ACCOUNT_TYPE_LOCAL là một loại tài khoản đặc biệt của lịch, không có mối liên kết với tài khoản trên thiết bị. Lịch của tài khoản này không được đồng bộ với server. Để thảo luận thêm về adapter có chức năng đồng bộ, xem mục "Adapter có chức năng đồng bộ".

Bảng Events

Bảng CalendarContract.Events bao gồm thông tin chi tiết cho từng sự kiện riêng. Để thêm, cập nhật hoặc xóa sự kiện, một ứng dụng phải thêm quyền WRITE_CALENDAR vào file file kê khai của nó.

Các cột dưới đây của bảng Events có thể được ứng dụng và adapter có chức năng đồng bộ chỉnh sửa. Để có danh sách đầy đủ các trường được hỗ trợ này, xem phần tham khảo của CalendarContract.Events.

Hằng	Mô tả
CALENDAR_ID	_ID của lịch chứa sự kiện.
ORGANIZER	Địa chỉ e-mail của tổ chức (hoặc chủ) của sự kiện.
TITLE	Tiêu đề của sự kiện.
EVENT_LOCATION	Địa điểm diễn ra sự kiện.
DESCRIPTION	Phần mô tả sự kiện.
DTSTART	Thời gian bắt đầu sự kiện, tính bằng đơn vị ms (mili giây), theo chuẩn UTC, tính từ epoch (kỷ nguyên Android, tức thời điểm 1/1/1970).
DTEND	Thời gian kết thúc sự kiện, tính bằng ms, theo chuẩn UTC, tính từ epoch.
EVENT_TIMEZONE	Múi giờ của sự kiện.
EVENT_END_TIMEZONE	Múi giờ của thời gian kết thúc sự kiện.
DURATION	Khoảng thời gian diễn ra sự kiện theo định dạng RFC5545. Ví dụ, giá trị của trạng thái "PT1H" cho biết sự kiện sẽ kéo dài trong một giờ, và giá trị "P2W" tương ứng với khoảng thời gian là 2 tuần.
ALL_DAY	Giá trị 1 cho biết sự kiện diễn ra cả ngày, tại múi giờ khu vực đó. Giá trị 0 cho biết đây là sự kiện định kỳ có thể bắt đầu và kết thúc vào bất cứ thời điểm nào trong một ngày.
RRULE	Quy định về tần suất của dạng sự kiện này. Ví dụ, "FREQ=WEEKLY;COUNT=10;WKST=SU". Bạn có thể tham khảo thêm ví dụ tại đây.
RDATE	Các ngày lặp lại sự kiện. Bạn thường sử dụng RDATE kết hợp với RRULE để xác định tập hợp các sự kiện lặp lại. Để thảo luận thêm về điều này, xem mục "RFC5545 spec" ("Giải thích các quy ước RFC5545").
AVAILABILITY	Sự kiện này được tính là diễn ra trong thời gian bận rộn hay rảnh rỗi. Đây là đặc điểm có thể lên lịch trước.
GUESTS_CAN_MODIFY	Xác định xem liệu người tham dự có được chỉnh sửa thông tin sự kiện hay không.
GUESTS_CAN_INVITE_OTHERS	Xác định xem liệu người tham dự có được mời thêm người khác không.

Hằng	Mô tả
GUESTS_CAN_SEE_GUESTS	Xác định xem liệu người tham dự có thể xem danh sách toàn bộ khách mời hay không.

Thêm sự kiện

Khi ứng dụng của bạn chèn thêm sự kiện mới, chúng tôi khuyến nghị bạn nên sử dụng Intent INSERT, như mô tả trong mục "Sử dụng intent để chèn thêm sự kiện". Tuy nhiên, nếu cần, bạn có thể trực tiếp chèn sự kiện. Phần này sẽ hướng dẫn cách làm.

Sau đây là các quy tắc chèn thêm sự kiện mới:

- Bạn phải chèn thêm trường CALENDAR_ID và DTSTART.
- Bạn cũng phải thêm trường EVENT_TIMEZONE. Để lấy ra danh sách các ID cho múi giờ (time zone) được cài đặt của hệ thống, sử dụng phương thức getAvailableIDs(). Lưu ý, luật này không được áp dụng khi bạn chèn một sự kiện bằng intent INSERT, như mô tả trong mục "Sử dụng intent để chèn thêm sự kiện" - theo cách này, sự kiện của bạn sẽ được cung cấp múi giờ mặc định.
- Đối với các sự kiện không lặp lại, bạn phải thêm trường DTEND.
- Với sự kiện lặp lại, bạn phải thêm trường DURATION, đồng thời bổ sung thêm RRULE hay RDATE. Lưu ý, không áp dụng quy tắc này khi chèn sự kiện bằng intent INSERT, như mô tả trong mục "Sử dụng intent để chèn thêm sự kiện" - theo cách này, bạn có thể sử dụng một RRULE kết hợp với DTSTART và DTEND, sau đó ứng dụng Calendar sẽ tự động chuyển đổi chúng thành khoảng thời gian.

Dưới đây là ví dụ chèn thêm một sự kiện. Để đơn giản, các câu lệnh sau được thực hiện trong luồng giao diện người dùng (luồng chính). Thực tế, thao tác chèn và cập nhật nên được thực hiện trong một luồng không đồng bộ để chuyển hoạt động vào luồng chạy ngầm. Để tìm hiểu thêm, xem phần AsyncQueryHandler.

```
long calID =3;
long startMillis =0;
long endMillis =0;
Calendar beginTime =Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
startMillis = beginTime.getTimeInMillis();
Calendar endTime =Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);
endMillis = endTime.getTimeInMillis();
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, startMillis);
values.put(Events.DTEND, endMillis);
```

```
values.put(Events.TITLE,”Jazzercise”);
values.put(Events.DESCRIPTION,”Group workout”);
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE,”America/Los_Angeles”);
Uri uri = cr.insert(Events.CONTENT_URI, values);

// lấy ra ID của sự kiện là phần tử cuối cùng trong Uri
long eventID =Long.parseLong(uri.getLastPathSegment());
//
// ... thực hiện một số thao tác với ID của sự kiện này tại đây
//
//
```

> **Ghi chú:** Hãy xem xét cách ví dụ này lấy ID của sự kiện sau khi đã tạo xong sự kiện. Đây là cách dễ nhất để lấy ra ID của một sự kiện. Bạn thường cần ID của sự kiện để thực hiện các thao tác khác trên lịch - ví dụ, thêm khách mời hoặc lời nhắc nhở vào sự kiện.

Cập nhật sự kiện

Khi ứng dụng của bạn muốn cho phép người dùng chỉnh sửa một sự kiện, chúng tôi khuyến nghị bạn nên dùng intent EDIT, như mô tả trong mục “Sử dụng intent để chỉnh sửa sự kiện”. Tuy nhiên, nếu cần, bạn có thể trực tiếp chỉnh sửa sự kiện. Để thực hiện cập nhật trên một sự kiện, bạn có thể cung cấp `_ID` của sự kiện thông qua việc ghép thêm ID vào Uri (bằng phương thức `withAppendedId()`), hoặc lấy mục chọn đầu tiên. Mệnh đề chọn phải bắt đầu với `“_id=?”`, còn đối số chọn `selectionArg` đầu tiên phải là `_ID` của sự kiện. Bạn cũng có thể cập nhật bằng cách sử dụng mệnh đề chọn không có ID. Sau đây là một ví dụ về cập nhật sự kiện. Ví dụ này sẽ thay đổi tiêu đề (title) của sự kiện, sử dụng `withAppendedId()`:

```
private static final String DEBUG_TAG = ”MyActivity”;
...
long eventID =188;
...
ContentResolver cr = getContentResolver();
ContentValues values =newContentValues();
Uri updateUri = null;
// Tiêu đề mới của sự kiện
values.put(Events.TITLE,”Kickboxing”);
updateUri =ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG,”Rows updated: “+ rows);
```

Xóa sự kiện

Bạn có thể xóa sự kiện bằng _ID của sự kiện này dưới dạng ghép thêm ID vào URI, hoặc bằng cách sử dụng mệnh đề chọn chuẩn. Nếu sử dụng cách gắn thêm ID, bạn không phải thực hiện phép chọn. Có hai kiểu xóa: Dùng ứng dụng và dùng adapter có chức năng cập nhật. Ứng dụng xóa các bản ghi có cột deleted bằng 1. Cờ này báo với adapter có chức năng đồng bộ là hàng đã bị xóa và việc xóa này phải được áp dụng vào server. Adapter có chức năng đồng bộ loại bỏ hoàn toàn sự kiện khỏi cơ sở dữ liệu cùng với những dữ liệu liên quan tới sự kiện này. Sau đây là ví dụ dùng ứng dụng để xóa một sự kiện thông qua _ID của sự kiện đó:

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 201;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri deleteUri = null;
deleteUri =ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().delete(deleteUri,null,null);
Log.i(DEBUG_TAG,"Rows deleted: "+ rows);
```

Bảng Attendees

Mỗi hàng của bảng CalendarContract.Attendees đại diện cho một người tham dự hoặc khách mời của sự kiện. Gọi phương thức query() để trả về danh sách khách tham dự sự kiện với EVENT_ID cho trước. EVENT_ID này phải phù hợp với _ID của một sự kiện cụ thể.

Bảng dưới đây liệt kê các trường có thể ghi được. Khi chèn thêm khách mời mới, bạn phải cung cấp giá trị cho tất cả các trường này, ngoại trừ ATTENDEE_NAME.

Hằng	Mô tả
EVENT_ID	ID sự kiện.
ATTENDEE_NAME	Tên người tham dự.
ATTENDEE_EMAIL	Địa chỉ e-mail của người tham dự.

Hằng	Mô tả
ATTENDEE_RELATIONSHIP	Mối quan hệ của người tham dự với sự kiện. Nhận một trong các mối quan hệ sau: • RELATIONSHIP_ATTENDEE (Người tham dự) • RELATIONSHIP_NONE (Chưa xác định) • RELATIONSHIP_ORGANIZER Người trong tổ chức) • RELATIONSHIP_PERFORMER (Người biểu diễn) • RELATIONSHIP_SPEAKER (Người diễn thuyết)
ATTENDEE_TYPE	Kiểu người tham dự. Thuộc một trong các kiểu sau: • TYPE_REQUIRED (Bắt buộc) • TYPE_OPTIONAL (Có thể tham dự hoặc không)
ATTENDEE_STATUS	Trạng thái tham gia của người tham dự. Nhận một trong các trạng thái sau: • ATTENDEE_STATUS_ACCEPTED (Chấp nhận) • ATTENDEE_STATUS_DECLINED (Từ chối) • ATTENDEE_STATUS_INVITED (Được mời) • ATTENDEE_STATUS_NONE (Không xác định) • ATTENDEE_STATUS_TENTATIVE (Chưa quyết định)

Thêm người tham dự

Sau đây là ví dụ về việc thêm một người tham dự vào sự kiện. Lưu ý, giá trị EVENT_ID là bắt buộc:

```
long eventID = 202;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME,”Trevor”);
```

```
values.put(Attendees.ATTENDEE_EMAIL,"trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP,Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE,Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS,Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = cr.insert(Attendees.CONTENT_URI, values);
```

Bảng Reminders

Mỗi hàng của bảng `CalendarContract.Reminders` tương ứng với một lời nhắc nhở của sự kiện. Lời gọi phương thức `query()` trả về danh sách các lời nhắc nhở của sự kiện có `EVENT_ID` cho trước.

Bảng dưới đây liệt kê các trường có thể ghi được của bảng nhắc nhở. Khi thêm một lời nhắc mới, bạn phải cung cấp giá trị cho tất cả các trường này. Lưu ý, các adapter có chức năng đồng bộ xác định kiểu nhắc nhở mà chúng hỗ trợ trong bảng `CalendarContract.Calendars`. Xem mục "ALLOWED REMINDERS" ("Các kiểu nhắc nhở được cho phép") để biết thêm chi tiết.

Hằng	Mô tả
`EVENT_ID`	ID của sự kiện.
`MINUTES`	Số phút khởi động lời nhắc nhở trước khi sự kiện xảy ra.
`METHOD`	Phương thức cảnh báo, khi cài đặt trên server. Nhận một trong các giá trị sau: • `METHOD_ALERT` (Cảnh báo) • `METHOD_DEFAULT` (Mặc định) • `METHOD_EMAIL` (E-mail) • `METHOD_SMS` (Tin nhắn SMS)

Thêm lời nhắc nhở

Ví dụ sau thêm một lời nhắc nhở vào sự kiện. Lời nhắc nhở được khởi động trước sự kiện 15 phút.

```
long eventID = 221;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES,15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = cr.insert(Reminders.CONTENT_URI, values);
```

Bảng Instances

Bảng `CalendarContract.Instances` quản lý thời điểm bắt đầu và kết thúc mỗi lần diễn ra một sự kiện. Mỗi hàng trong bảng đại diện cho một lần sự kiện diễn ra. Bảng Instances không cho quyền ghi và chỉ cung cấp một cách để truy vấn các lần diễn ra sự kiện.

Bảng sau liệt kê một số trường bạn có thể dùng để truy vấn cho một thể hiện. Lưu ý, múi giờ là do `KEY_TIMEZONE_TYPE` và `KEY_TIMEZONE_INSTANCES` xác định.

Hằng	Mô tả
`BEGIN`	Thời điểm bắt đầu của một thể hiện, tính đến mili giây theo chuẩn UTC.
`END`	Thời điểm kết thúc của một thể hiện, tính đến mili giây theo chuẩn UTC.
`END_DAY`	Ngày kết thúc theo lịch Julian của thể hiện, có liên hệ với múi giờ của Calendar.
`END_MINUTE`	Thời điểm kết thúc theo phút của thể hiện, được tính từ giữa đêm theo múi giờ của Calendar.
`EVENT_ID`	`_ID` của sự kiện tương ứng với thể hiện này.
`START_DAY`	Ngày bắt đầu theo lịch Julian của thể hiện, có liên hệ với múi giờ của Calendar.
`START_MINUTE`	Thời điểm bắt đầu theo phút của thể hiện, được tính từ giữa đêm, có liên hệ với múi giờ của Calendar.

Thực hiện truy vấn trên bảng Instances

Để truy vấn trên bảng Instances, bạn cần xác định khoảng thời gian cho truy vấn trong URI. Trong ví dụ này, `CalendarContract.Instances` lấy truy cập tới trường `TITLE` thông qua bản cài đặt giao diện `CalendarContract.EventsColumns`. Nói cách khác, trường `TITLE` được trả về thông qua một view của cơ sở dữ liệu, chứ không qua truy vấn trên bảng `CalendarContract.Instances` thô.

```
private static final String DEBUG_TAG = ”MyActivity”;
public static final String[] INSTANCE_PROJECTION = new String[] {
       Instances.EVENT_ID,      // 0
       Instances.BEGIN,         // 1
       Instances.TITLE          // 2
    };

// Các chỉ số của mảng đối số pojection trên.
private static final int PROJECTION_ID_INDEX =0;
private static final int PROJECTION_BEGIN_INDEX =1;
private static final int PROJECTION_TITLE_INDEX =2;
...

// Xác định phạm vi ngày tháng bạn muốn thực hiện tìm kiếm các thể
// hiện của sự kiện
Calendar beginTime = Calendar.getInstance();
beginTime.set(2011, 9, 23, 8, 0);
long startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2011, 10, 24, 8, 0);
long endMillis = endTime.getTimeInMillis();

Cursor cur = null;
ContentResolver cr = getContentResolver();

// ID của sự kiện mà bạn đang tìm kiếm các thể hiện của nó trên bảng
// Instances
String selection = Instances.EVENT_ID + ” = ?”;
String[] selectionArgs = new String[] {“207”};

// Xây dựng truy vấn với phạm vi ngày tháng muốn thực hiện tìm kiếm.
Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(builder, startMillis);
ContentUris.appendId(builder, endMillis);

// Thực hiện truy vấn
cur = cr.query(builder.build(),
    INSTANCE_PROJECTION,
    selection,
    selectionArgs,
    null);

while(cur.moveToNext()){
    String title =null;
    long eventID =0;
    long beginVal =0;
```

```
    // Lấy giá trị của các trường
    eventID = cur.getLong(PROJECTION_ID_INDEX);
     beginVal = cur.getLong(PROJECTION_BEGIN_INDEX);
     title = cur.getString(PROJECTION_TITLE_INDEX);

    // Thực hiện một số thao tác với các giá trị này.
    Log.i(DEBUG_TAG,"Event:  "+ title);
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(beginVal);
    DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
    Log.i(DEBUG_TAG,"Date: "+ formatter.format(calendar.getTime()));
    }
}
```

Các intent của Calendar

Ứng dụng của bạn không cần phải có quyền để đọc và ghi dữ liệu lịch. Thay vào đó, ứng dụng do bạn phát triển có thể dùng các intent được ứng dụng Calendar của Android hỗ trợ để thực hiện những thao tác đọc và ghi với ứng dụng này. Bảng sau đây liệt kê các intent được Provider Calendar hỗ trợ:

Action	URI	Mô tả	Các trường phụ hỗ trợ (extra)
VIEW	content://com.android.calendar/time/<ms_since_epoch> Bạn cũng có thể trỏ tới URI này thông qua hằng CalendarContract.CONTENT_URI. Về ví dụ sử dụng intent, xem mục "Sử dụng intent để hiển thị dữ liệu lịch".	Mở lịch với thời gian được xác định bởi <ms_since_epoch>.	Không có.
VIEW	content://com.android.calendar/events/<event_id> Bạn cũng có thể trỏ tới URI này bằng hằng Events.CONTENT_URI. Về ví dụ sử dụng intent này, xem mục "Sử dụng intent để chỉnh sửa sự kiện".	Hiển thị sự kiện được xác định bởi <event_id>.	CalendarContract.EXTRA_EVENT_BEGIN_TIME CalendarContract.EXTRA_EVENT_END_TIME

Action	URI	Mô tả	Các trường phụ hỗ trợ (extra)
EDIT	content://com.android.calendar/events/<event_id> Bạn cũng có thể trỏ tới URI này bằng hằng Events.CONTENT_URI. Về ví dụ sử dụng intent này, xem mục "Sử dụng intent để chỉnh sửa sự kiện".	Chỉnh sửa sự kiện được xác định bởi <event_id>.	CalendarContract.EXTRA_EVENT_BEGIN_TIME CalendarContract.EXTRA_EVENT_END_TIME
EDIT INSERT	content://com.android.calendar/events Bạn có thể trỏ tới URI này bằng hằng Events.CONTENT_URI. Về ví dụ sử dụng intent này, xem mục "Sử dụng intent để chèn thêm sự kiện".	Tạo một sự kiện.	Bất cứ trường phụ hỗ trợ nào được liệt kê ở bảng bên dưới.

Bảng sau liệt kê một số trường phụ hỗ trợ intent (extra) được Provider Calendar cung cấp:

Trường phụ hỗ trợ intent (extra)	Mô tả
Events.TITLE	Tên sự kiện.
CalendarContract.EXTRA_EVENT_BEGIN_TIME	Thời điểm bắt đầu của sự kiện, tính theo mili giây từ epoch.
CalendarContract.EXTRA_EVENT_END_TIME	Thời điểm kết thúc sự kiện, tính theo mili giây từ epoch.
CalendarContract.EXTRA_EVENT_ALL_DAY	Một giá trị kiểu boolean cho biết sự kiện có diễn ra cả ngày hay không. Giá trị có thể là true hoặc false.
Events.EVENT_LOCATION	Địa điểm tổ chức sự kiện.
Events.DESCRIPTION	Mô tả về sự kiện.
Intent.EXTRA_EMAIL	Các địa chỉ e-mail của khách mời dưới dạng một danh sách phân cách nhau bởi dấu phẩy.
Events.RRULE	Quy tắc lặp sự kiện.
Events.ACCESS_LEVEL	Xác định xem sự kiện là riêng tư (private) hay công cộng (public).

Trường phụ hỗ trợ intent (extra)	Mô tả
Events.AVAILABILITY	Sự kiện này được tính là diễn ra trong thời gian bận rộn hay rảnh rỗi. Đây là đặc điểm có thể lên lịch trước.

Các mục dưới đây mô tả cách sử dụng những intent này.

Sử dụng intent để chèn thêm sự kiện

Sử dụng intent INSERT cho phép ứng dụng mà bạn phát triển chuyển tác vụ chèn sự kiện sang Calendar. Với cách tiếp cận này, ứng dụng của bạn không cần phải có quyền WRITE_CALENDAR trong file kê khai.

Khi người dùng chạy ứng dụng theo cách tiếp cận này, ứng dụng sẽ gửi họ tới Calendar để hoàn thành thao tác thêm sự kiện. Intent INSERT sử dụng các trường phụ để lưu trước một form chứa những thông tin chi tiết của sự kiện trong Calendar. Sau đó, người dùng có thể hủy sự kiện, chỉnh sửa form đó (nếu cần), hoặc lưu sự kiện vào lịch của họ.

Sau đây là đoạn mã nhỏ lên lịch cho một sự kiện được tổ chức vào ngày 19 tháng 1 năm 2012, diễn ra từ 7 giờ 30 phút sáng đến 8 giờ 30 phút sáng. Hãy lưu ý các đặc điểm dưới đây của đoạn mã nhỏ này:

- Đoạn mã xác định hằng Events.CONTENT_URI làm Uri.
- Đoạn mã sử dụng các trường phụ CalendarContract.EXTRA_EVENT_BEGIN_TIME và CalendarContract.EXTRA_EVENT_END_TIME để lưu trước một form chứa thời gian của sự kiện. Các giá trị thời gian này được tính theo mili giây, theo chuẩn UTC, tính từ epoch.
- Đoạn mã sử dụng trường phụ Intent.EXTRA_EMAIL để cung cấp một danh sách khách mời phân cách nhau bởi dấu phẩy. Khách mời được xác định bằng địa chỉ e-mail.

```
Calendar beginTime =Calendar.getInstance();
beginTime.set(2012,0,19,7,30);
Calendar endTime =Calendar.getInstance();
endTime.set(2012,0,19,8,30);
Intent intent =new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,
           beginTime.getTimeInMillis())
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
           endTime.getTimeInMillis())
        .putExtra(Events.TITLE,"Yoga")
        .putExtra(Events.DESCRIPTION,"Group class")
        .putExtra(Events.EVENT_LOCATION,"The gym")
        .putExtra(Events.AVAILABILITY,Events.AVAILABILITY_BUSY)
```

```
        .putExtra(Intent.EXTRA_EMAIL,"rowan@example.com,
           trevor@example.com");
startActivity(intent);
```

Sử dụng intent để chỉnh sửa sự kiện

Bạn có thể trực tiếp cập nhật sự kiện, như mô tả trong mục "Cập nhật sự kiện". Tuy nhiên, khi bạn sử dụng intent EDIT sẽ cho phép một ứng dụng không được cấp quyền có thể chuyển thao tác chỉnh sửa tới ứng dụng Calendar. Khi người dùng hoàn thành việc chỉnh sửa sự kiện trên Calendar, họ sẽ được đưa trở lại ứng dụng ban đầu.

Sau đây là ví dụ về một intent thiết lập tiêu đề mới cho một sự kiện xác định, cho phép người dùng chỉnh sửa sự kiện trong Calendar.

```
long eventID =208;
Uri uri =ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent =new Intent(Intent.ACTION_EDIT)
    .setData(uri)
    .putExtra(Events.TITLE,"My New Title");
startActivity(intent);
```

Sử dụng intent để hiển thị dữ liệu lịch

Provider Calendar đem lại hai cách khác nhau để sử dụng intent VIEW:

- Mở lịch của một ngày cụ thể.
- Xem một sự kiện.

Sau đây là ví dụ minh họa cách mở lịch của một ngày cụ thể:

```
// Ngày giờ tính theo mili giây từ epoch.
long startMillis;
...
Uri.Builder builder =CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, startMillis);
Intent intent =new Intent(Intent.ACTION_VIEW)
    .setData(builder.build());
startActivity(intent);
```

Sau đây là ví dụ minh họa cách mở một sự kiện để xem:

```
long eventID =208;
...
Uri uri =ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent =new Intent(Intent.ACTION_VIEW)
    .setData(uri);
startActivity(intent);
```

Adapter có chức năng đồng bộ

Có một số khác biệt nhỏ trong cách một ứng dụng và một adapter có chức năng đồng bộ truy cập Provider Calendar:

- Adapter có chức năng đồng bộ cần xác định rằng nó chính là adapter có chức năng đồng bộ bằng việc thiết lập `CALLER_IS_SYNCADAPTER` là true.
- Adapter có chức năng đồng bộ cần cung cấp `ACCOUNT_NAME` và `ACCOUNT_TYPE` làm tham số truy vấn trong URI.
- Adapter có chức năng đồng bộ có quyền truy cập ghi vào nhiều cột hơn so với ứng dụng hoặc widget. Ví dụ, một ứng dụng chỉ có thể chỉnh sửa một số đặc điểm của lịch, chẳng hạn như tên, tên hiển thị, cài đặt hiển thị và xác định xem lịch có được đồng bộ hay không. Khi so sánh, adapter có chức năng đồng bộ không chỉ được truy cập vào những cột này mà còn nhiều cột khác, như màu của lịch, múi giờ, mức truy cập, nơi diễn ra,… Tuy nhiên, adapter có chức năng đồng bộ bị giới hạn quyền tương ứng với `ACCOUNT_NAME` và `ACCOUNT_TYPE` mà nó xác định.

Sau đây là phương thức trợ giúp mà bạn có thể dùng để trả về URI sử dụng với một adapter có chức năng đồng bộ:

```
static Uri asSyncAdapter(Uri uri,String account,String accountType){
    return uri.buildUpon()
        .appendQueryParameter(android.provider.CalendarContract.
CALLER_IS_SYNCADAPTER,”true”)
    .appendQueryParameter(Calendars.ACCOUNT_NAME, account)
    .appendQueryParameter(Calendars.ACCOUNT_TYPE, accountType).build();
 }
```

Để tham khảo ví dụ cài đặt mẫu của adapter có chức năng đồng bộ (không chỉ liên quan tới Calendar), xem SampleSyncAdapter.

10.4 Provider Contacts

Provider Contacts nằm trong hệ điều hành Android, là một thành phần linh hoạt, mạnh mẽ, quản lý kho lưu trữ trung tâm chứa dữ liệu thông tin về con người trên thiết bị. Provider Contacts là nguồn của dữ liệu bạn thấy trong ứng dụng contacts (thông tin liên lạc); trong ứng dụng của mình, bạn còn có thể truy cập dữ liệu này, cũng như chuyển

dữ liệu giữa thiết bị và các service trực tuyến. Provider này lấy thông tin từ nhiều nguồn dữ liệu và cố gắng quản lý nhiều dữ liệu nhất có thể cho từng cá nhân (contact), nên kết quả khá phức tạp. Do vậy, API của provider này chứa một tập lớp contract lớn, cùng với đó là những giao diện hỗ trợ đắc lực cho việc truy xuất và chỉnh sửa dữ liệu.

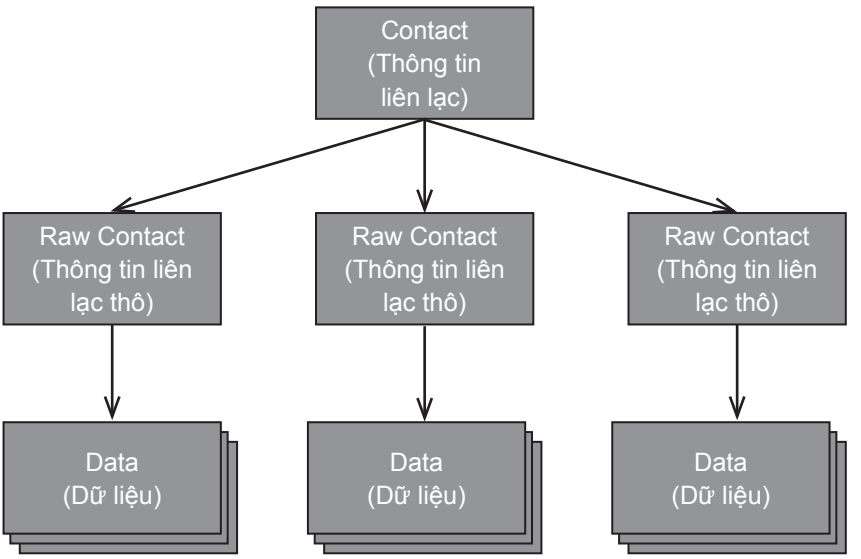
Phần này sẽ giới thiệu các nội dung sau:

- Cấu trúc cơ bản của provider.
- Hướng dẫn cách truy xuất dữ liệu từ provider.
- Hướng dẫn cách chỉnh sửa dữ liệu trong provider.
- Hướng dẫn cách viết một adapter có chức năng đồng bộ để đồng bộ hóa dữ liệu từ server tới Provider Contacts.

Phần này giả sử bạn đã nắm được các khái niệm cơ bản về content provider Android. Để tìm hiểu thêm về provider này, tham khảo mục "Cơ bản về content provider". Ứng dụng mẫu Sample Sync Adapter (Adapter có chức năng đồng bộ mẫu) là một ví dụ về sử dụng adapter có chức năng đồng bộ (sync adapter) để chuyển dữ liệu giữa Provider Contacts và ứng dụng mẫu nằm trên Google Web Services (Các service Web của Google).

Tổ chức bên trong Provider Contacts

Provider Contacts là một thành phần cung cấp nội dung của Android. Trình này quản lý ba kiểu dữ liệu về một người, mỗi kiểu tương ứng với một bảng trong provider, như minh họa ở Hình 1:



Hình 1. Cấu trúc bảng của Provider Contacts.

Ba bảng này thường được tham chiếu bằng tên các lớp contract của chúng. Những lớp này định nghĩa các hằng cho các content URI, tên cột và giá trị của cột được sử dụng trong bảng:

Bảng `ContactsContract.Contacts`

Mỗi hàng đại diện cho một người, dựa trên sự kết hợp của các hàng dữ liệu liên lạc thô.

Bảng `ContactsContract.RawContacts`

Các hàng này lưu bản tóm tắt dữ liệu của một người, đặc trưng cho loại và tài khoản của người dùng.

Bảng `ContactsContract.Data`

Hàng chứa thông tin chi tiết về dữ liệu liên lạc thô, chẳng hạn như địa chỉ e-mail hoặc số điện thoại.

Những bảng còn lại được đại diện bằng các lớp contract trong `ContactsContract` là những bảng bổ trợ mà Provider Contacts dùng để quản lý các thao tác và hỗ trợ những chức năng cụ thể trong thông tin liên lạc của thiết bị hoặc ứng dụng điện thoại.

Thông tin liên lạc thô

Một thông tin liên lạc thô (raw contact) đại diện cho dữ liệu của mỗi người tương ứng với một tên tài khoản (account name) và một loại tài khoản (account type). Do Provider Contacts cho phép nhiều hơn một service trực tuyến làm nguồn dữ liệu cho một người dùng, nên provider này cho phép tồn tại nhiều dữ liệu liên lạc thô của cùng một người. Nhiều dữ liệu liên lạc thô cũng cho phép người dùng kết hợp dữ liệu của một người từ nhiều tài khoản thuộc cùng một loại tài khoản.

Hầu hết dữ liệu của thông tin liên lạc thô không được lưu trong bảng `ContactsContract.RawContacts`. Thay vào đó, chúng được lưu trong một hoặc nhiều hàng của bảng `ContactsContract.Data`. Mỗi hàng dữ liệu có một cột là `Data.RAW_CONTACT_ID` chứa giá trị `RawContacts._ID` của hàng `ContactsContract.RawContacts` cha.

Các cột dữ liệu liên lạc thô quan trọng

Các cột quan trọng trong bảng `ContactsContract.RawContacts` được liệt kê ở Bảng 1. Hãy đọc kỹ phần Ghi chú bên dưới bảng này:

Bảng 1. Các cột dữ liệu thô quan trọng.

Tên cột	Mục đích sử dụng	Chú thích
ACCOUNT_NAME	Tên tài khoản của loại tài khoản làm nguồn cho thông tin liên lạc thô này. Ví dụ, tên tài khoản của tài khoản Google là một trong các địa chỉ Gmail của chủ thiết bị. Xem mục tiếp theo về ACCOUNT_TYPE để tìm hiểu thêm.	Định dạng của tên này đặc trưng cho loại tài khoản của nó. Không nhất thiết phải là địa chỉ e-mail.
ACCOUNT_TYPE	Loại tài khoản làm nguồn cho thông tin liên lạc thô này. Ví dụ, loại tài khoản của tài khoản Google là `com.google`. Luôn luôn xác định loại tài khoản của bạn bằng một định danh miền (domain identifier) của miền bạn làm chủ hoặc quản lý. Điều này đảm bảo rằng loại tài khoản của bạn là duy nhất.	Loại tài khoản cung cấp thông tin liên lạc thường có một adapter có chức năng đồng bộ tương ứng để đồng bộ với Provider Contacts.
DELETED	Cờ "xóa" của một liên lạc thô.	Cờ này cho phép Provider Contacts vẫn giữ hàng ở trong bảng cho đến khi adapter có chức năng đồng bộ có thể xóa hàng từ server, sau đó hoàn thành việc xóa hàng tại kho lưu trữ.

Ghi chú

Sau đây là một số lưu ý quan trọng về bảng `ContactsContract.RawContacts`:

- Tên thông tin liên lạc thô không lưu trong hàng của bảng `ContactsContract.RawContacts`. Thay vào đó, các thông tin này được lưu trong bảng `ContactsContract.Data`, trong hàng `ContactsContract.CommonDataKinds.StructuredName`. Mỗi thông tin liên lạc thô chỉ có một hàng kiểu này trong bảng `ContactsContract.Data`.
- **Lưu ý:** Để sử dụng dữ liệu tài khoản của chính bạn trong một hàng thông tin liên lạc thô, trước tiên, dữ liệu này phải được đăng ký với `AccountManager` (trình quản lý tài khoản). Để thực hiện điều này, hãy nhắc nhở người dùng thêm loại tài khoản và tên tài khoản của họ vào danh sách các tài khoản. Nếu bạn không thực hiện, Provider Contacts sẽ tự động xóa hàng thông tin liên lạc thô của bạn.

  Ví dụ, nếu bạn muốn ứng dụng của mình lưu lại dữ liệu liên lạc của một service trên nền Web có tên miền là `com.example.dataservice`, và tài khoản người dùng trên service của bạn là `becky.sharp@dataservice.example.com`, trước tiên, người dùng phải thêm "loại" tài khoản `(com.example.dataservice)` và

"tên" tài khoản (becky.sharp@dataservice.example.com) để ứng dụng của bạn có thể thêm các hàng thông tin liên lạc thô. Bạn có thể giải thích yêu cầu này với người dùng trong tài liệu hướng dẫn, hoặc có thể nhắc nhở họ thêm loại và tên, hay cả hai. Mục tiếp theo mô tả chi tiết hơn về loại tài khoản và tên tài khoản.

Nguồn của dữ liệu liên lạc thô

Để hiểu về cách làm việc của dữ liệu liên lạc thô, giả sử rằng người dùng "Emily Dickinson" có ba tài khoản người dùng sau được xác định trên thiết bị của cô ấy:

- emily.dickinson@gmail.com.
- emilyd@gmail.com.
- Tài khoản Twitter "belle_of_amherst".

Người dùng này bật chức năng *Sync Contacts (đồng bộ thông tin liên lạc)* cho cả ba tài khoản trong phần cài đặt *Accounts (tài khoản)*.

Giả sử Emily Dickinson mở một cửa sổ trình duyệt, đăng nhập vào Gmail bằng tài khoản emily.dickinson@gmail.com, mở phần Contacts và thêm vào liên lạc "Thomas Higginson". Sau đó, cô ấy đăng nhập vào Gmail bằng tài khoản emilyd@gmail.com và gửi e-mail cho "Thomas Higginson", việc này sẽ tự động thêm Thomas thành một liên lạc. Emily cũng đưa tài khoản "colonel_tom" (ID Twitter của Thomas Higginson) vào Twitter.

Sau các thao tác trên, Provider Contacts tạo ra ba dữ liệu liên lạc dạng thô:

1. Một dữ liệu liên lạc dạng thô của "Thomas Higginson" liên kết với tài khoản emily.dickinson@gmail.com. Loại tài khoản của người dùng là Google.
2. Dữ liệu liên lạc thô thứ hai của "Thomas Higginson" liên kết với tài khoản emilyd@gmail.com. Loại tài khoản của người dùng này cũng là Google. Thậm chí, có dữ liệu liên lạc thô thứ hai giống hệt tên trong dữ liệu liên lạc thô thứ nhất, bởi vì người này được thêm vào trong một tài khoản người dùng khác.
3. Dữ liệu liên lạc thô thứ ba của "Thomas Higginson" liên kết với tài khoản "belle_of_amherst". Tài khoản người dùng loại này là Twitter.

Dữ liệu

Như đã ghi chú ở mục trước, dữ liệu của liên lạc thô ban đầu được lưu trong hàng ContactsContract.Data liên kết với giá trị _ID của dữ liệu liên lạc thô. Điều này cho phép một dữ liệu thô có nhiều thể hiện thuộc cùng một loại dữ liệu như địa chỉ e-mail hoặc số điện thoại. Ví dụ, nếu "Thomas Higginson" của tài khoản emilyd@gmail.com (hàng dữ liệu liên lạc thô của Thomas Higginson liên kết với tài khoản Google emilyd@gmail.com) có một địa chỉ e-mail ở nhà thigg@gmail.com và một e-mail công việc thomas.higginson@gmail.com, Provider Contacts sẽ lưu hai hàng địa chỉ e-mail và liên kết cả hai với dữ liệu liên lạc thô.

Lưu ý, các loại dữ liệu khác nhau đều được lưu trong bảng này. Tên hiển thị, số điện thoại, e-mail, địa chỉ bưu điện, ảnh và các hàng chi tiết về Website đều được lưu trong bảng ContactsContract.Data. Để giúp quản lý các thông tin này, bảng ContactsContract.Data chứa vài cột có tên mô tả (descriptive name) và những

cột khác có tên chung chung (generic name). Nội dung của cột tên mô tả có cùng ý nghĩa với kiểu dữ liệu trong hàng, trong khi nội dung của cột tên chung chung có các ý nghĩa khác nhau tùy theo kiểu dữ liệu.

Tên cột mô tả

Một số ví dụ về tên cột mô tả là:

`RAW_CONTACT_ID`

Là giá trị cột _ID cho thông tin liên lạc thô của dữ liệu này.

`MIMETYPE`

Loại dữ liệu được lưu trong hàng này là kiểu MIME tùy chỉnh. Provider Contacts sử dụng những kiểu MIME được định nghĩa trong các lớp con của `ContactsContract.CommonDataKinds`. Những kiểu MIME này là mã nguồn mở (open source), có thể được bất cứ ứng dụng hay adapter có chức năng đồng bộ nào làm việc với Provider Contacts sử dụng.

`IS_PRIMARY`

Nếu có nhiều kiểu hàng dữ liệu này trong bảng dữ liệu cho một thông tin liên lạc thô, cột `IS_PRIMARY` đặt cờ vào hàng dữ liệu chứa dữ liệu chính của loại này. Ví dụ, nếu người dùng nhập số điện thoại cho một liên lạc và chọn **Set default (thiết lập mặc định)**, hàng `ContactsContract.Data` chứa số điện thoại này có cột `IS_PRIMARY` được đặt một giá trị khác 0 (non-zero value).

Tên cột chung chung

Có 15 cột chung chung có tên từ `DATA1` đến `DATA15` luôn sẵn sàng để sử dụng và bốn cột chung chung bổ sung từ `SYNC1` đến `SYNC4` sẽ được các adapter có chức năng đồng bộ sử dụng. Các hằng tên cột chung chung luôn có tác dụng, bất kể kiểu dữ liệu trong hàng là gì.

Cột `DATA1` là cột được đánh chỉ mục. Provider Contacts luôn sử dụng cột `DATA1` cho dữ liệu mà trình này mong muốn các truy vấn sẽ thường xuyên hướng tới nhất. Ví dụ, trong hàng e-mail, cột này sẽ chứa địa chỉ e-mail thực.

Theo quy ước, cột `DATA15` được dành để lưu trữ dữ liệu BLOB, chẳng hạn như hình phác thảo nhỏ của ảnh chụp.

Tên cột có kiểu xác định

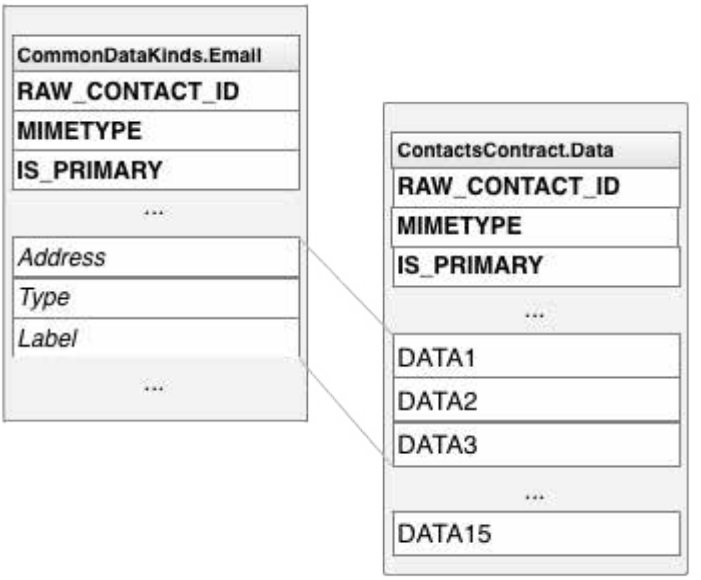
Để dễ làm việc với các cột có kiểu dữ liệu cụ thể trên hàng, Provider Contacts cũng cung cấp hằng tên cột có kiểu xác định (type-specific name), được định nghĩa trong lớp con `ContactsContract.CommonDataKinds`. Những hằng này chỉ cung cấp các tên hằng khác nhau cho cùng tên cột, giúp bạn truy cập dữ liệu trong một hàng của một kiểu cụ thể.

Ví dụ, lớp `ContactsContract.CommonDataKinds.Email` định nghĩa các hằng tên cột có kiểu xác định cho một hàng `ContactsContract.Data` có kiểu MIME là

Email.CONTENT_ITEM_TYPE. Lớp này có hằng ADDRESS tương ứng với cột địa chỉ e-mail. Giá trị thực sự của ADDRESS là "data1", chính là tên chung chung của cột.

> **Lưu ý**: Không đưa thêm dữ liệu tùy chỉnh của bạn vào bảng ContactsContract.Data bằng cách sử dụng một hàng có một trong các kiểu MIME định nghĩa trước của content provider. Nếu thực hiện điều này, bạn có thể làm mất dữ liệu hoặc khiến content provider bị lỗi. Ví dụ, bạn không nên thêm hàng với kiểu MIME là Email.CONTENT_ITEM_TYPE chứa tên người dùng thay vì địa chỉ e-mail trong cột DATA1. Nếu sử dụng kiểu MIME tùy chỉnh của mình cho hàng này, bạn có thể thoải mái định nghĩa tên cột có kiểu xác định và sử dụng các cột này theo ý muốn.

Hình 2 minh họa cách bố trí các cột miêu tả và cột dữ liệu trong hàng ContactsContract.Data, cùng với đó là cách tên cột có kiểu xác định "phủ lên" tên cột chung chung.



Address	Địa chỉ
Type	Loại
Lable	Nhãn

Hình 2. tên cột có kiểu xác định và tên cột chung chung.

Lớp tên cột có kiểu xác định

Bảng 2 dưới đây liệt kê các lớp tên cột có kiểu xác định thường dùng:

Bảng 2. Các lớp tên cột có kiểu xác định.

Lớp ánh xạ	Kiểu dữ liệu	Chú thích
ContactsContract.CommonDataKinds.StructuredName	Dữ liệu tên của liên lạc thô có liên kết với hàng dữ liệu này.	Mỗi dữ liệu liên lạc thô chỉ có duy nhất một hàng này.
ContactsContract.CommonDataKinds.Photo	Ảnh chính của dữ liệu liên lạc thô có liên kết với hàng dữ liệu này.	Mỗi dữ liệu liên lạc thô chỉ có duy nhất một hàng này.
ContactsContract.CommonDataKinds.Email	Địa chỉ e-mail của dữ liệu liên lạc thô có liên kết với hàng dữ liệu này.	Một dữ liệu thô có thể có nhiều địa chỉ e-mail.
ContactsContract.CommonDataKinds.StructuredPostal	Địa chỉ bưu điện của dữ liệu thô có liên kết với hàng dữ liệu này.	Một dữ liệu thô có thể có nhiều địa chỉ bưu điện.
ContactsContract.CommonDataKinds.GroupMembership	Một định danh liên kết dữ liệu liên lạc thô với một trong các nhóm của Provider Contacts.	Nhóm là một tính năng tùy chọn của một loại tài khoản và tên tài khoản. Khái niệm về nhóm được miêu tả chi tiết trong mục "Nhóm liên lạc".

Liên lạc

Provider Contacts kết hợp các hàng dữ liệu liên lạc thô thông qua tất cả các loại tài khoản và tên tài khoản để tạo thành một **liên lạc (contact)**. Điều này giúp hiển thị và chỉnh sửa dễ dàng mọi dữ liệu mà người dùng có thể thu thập cho một người. Provider Contacts quản lý việc tạo các hàng liên lạc mới, đồng thời tổng hợp những dữ liệu liên lạc thô với hàng liên lạc đã có. Cả ứng dụng lẫn adapter có chức năng đồng bộ đều không được phép thêm liên lạc, và một số cột chỉ được đọc (read-only) trong hàng liên lạc.

> **Ghi chú:** Nếu cố gắng thêm một liên lạc vào Provider Contacts bằng phương thức insert(), bạn sẽ nhận được ngoại lệ UnsupportedOperationException. Nếu bạn cố gắng cập nhật một cột được thiết lập là "chỉ đọc", thao tác cập nhật sẽ bị bỏ qua.

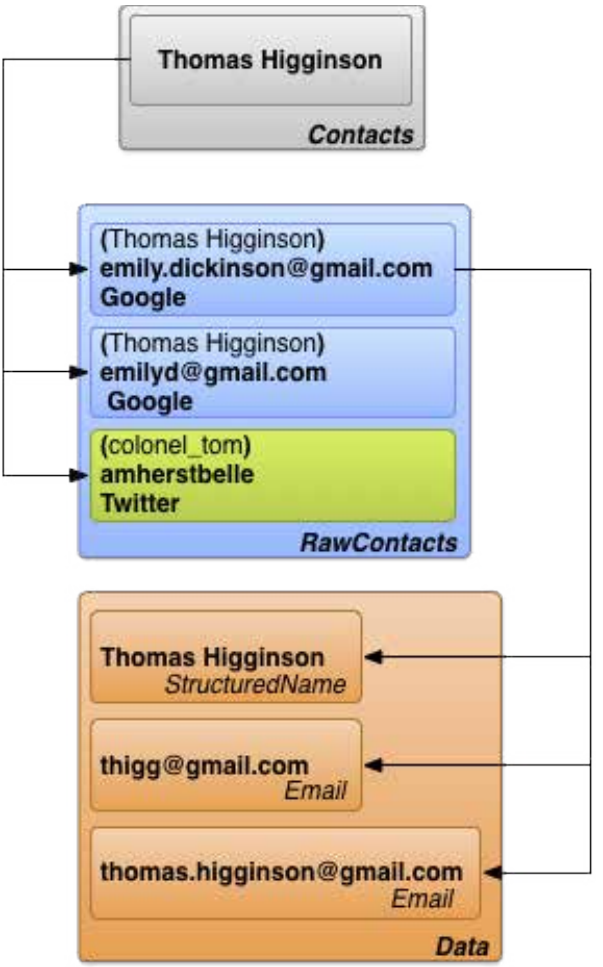
Provider Contacts tạo một liên lạc mới để phản hồi việc thêm một dữ liệu liên lạc thô mới không trùng với bất kỳ liên lạc nào hiện có. Provider cũng thực hiện điều này nếu dữ liệu liên lạc thô đã có thay đổi dựa trên việc dữ liệu này không còn khớp với liên lạc đã được thêm vào trước đó. Nếu một ứng dụng hoặc adapter có chức năng đồng bộ tạo một dữ liệu liên lạc thô mới trùng với liên lạc hiện có, dữ liệu liên lạc mới sẽ được kết hợp với liên lạc hiện có.

Provider Contacts liên kết một hàng liên lạc với các hàng dữ liệu liên lạc thô của nó bằng cột _ID của hàng liên lạc trong bảng Contacts. Cột CONTACT_ID của bảng

dữ liệu thô `ContactsContract.RawContacts` chứa giá trị `_ID` của hàng liên lạc có liên kết với mỗi hàng dữ liệu liên lạc thô.

Bảng `ContactsContract.Contacts` cũng có cột `LOOKUP_KEY` là cột liên kết "lâu dài" với hàng liên lạc. Do Provider Contacts tự động quản lý các liên lạc, nên trình này có thể thay đổi giá trị _ID của một hàng liên lạc để phản hồi một sự kết hợp hoặc đồng bộ. Thậm chí, nếu những thay đổi này xảy ra, content URI `CONTENT_LOOKUP_URI` kết hợp với `LOOKUP_KEY` của liên lạc vẫn trỏ tới hàng liên lạc này, nên bạn có thể sử dụng giá trị `LOOKUP_KEY` để duy trì liên kết tới những liên lạc "ưa thích",… Cột này có định dạng riêng không liên quan tới định dạng của cột `_ID`.

Hình 3 minh họa cách ba bảng chính liên kết với nhau.



Contacts	Liên lạc
Rawcontacts	Liên lạc thô
StructuredName	Tên có cấu trúc
Data	Dữ liệu

Hình 3. Mối quan hệ giữa các bảng Contacts, Raw Contacts và Details.

Dữ liệu từ Adapter có chức năng đồng bộ

Người dùng nhập dữ liệu liên lạc trực tiếp vào thiết bị, nhưng trong Provider Contacts, dữ liệu vẫn đi theo luồng từ Web service thông qua **adapter có chức năng đồng bộ (sync adapter)**, những adapter này sẽ tự động chuyển dữ liệu giữa thiết bị và các service. Adapter có chức năng đồng bộ chạy ngầm dưới sự điều khiển của hệ thống, và các phần này gọi phương thức `ContentResolver` để quản lý dữ liệu.

Trong Android, Web service mà adapter có chức năng đồng bộ làm việc cùng sẽ được một loại tài khoản xác định. Mỗi adapter có chức năng đồng bộ làm việc với một loại tài khoản, nhưng adapter này có thể hỗ trợ nhiều tên tài khoản thuộc loại trên. Loại tài khoản và tên tài khoản được mô tả tóm lược trong mục "Nguồn dữ liệu liên lạc thô". Các định nghĩa dưới đây sẽ cung cấp thêm thông tin chi tiết, đồng thời mô tả cách thức mà loại tài khoản cũng như tên tài khoản liên hệ với adapter có chức năng đồng bộ và service.

Loại tài khoản

> Xác định service mà người dùng lưu trữ dữ liệu vào đó. Trong hầu hết trường hợp, người dùng phải xác thực (authenticate) bản thân với service. Ví dụ, Google Contacts (Các tài khoản liên lạc của Google) là một loại tài khoản, so mã `google.com` xác định. Giá trị này tương ứng với loại tài khoản mà `AccountManager` sử dụng.

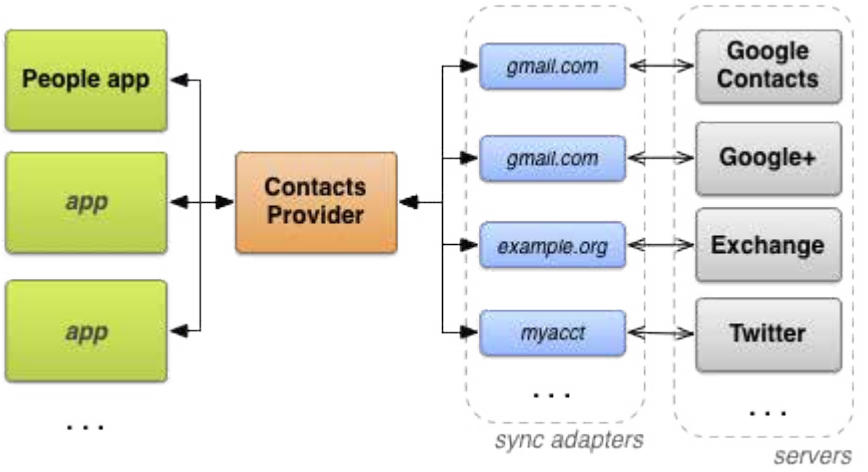
Tên tài khoản

> Xác định tài khoản cụ thể hoặc tài khoản đăng nhập (login) cho một loại tài khoản. Tài khoản Google Contacts giống với tài khoản Google, lấy địa chỉ e-mail làm tên tài khoản. Các service khác có thể sử dụng tên người dùng ở dạng từ đơn hoặc định danh bằng số không chứa dấu cách.

Loại tài khoản không phải là duy nhất. Người dùng có thể cấu hình nhiều tài khoản Google Contacts và tải (download) dữ liệu của họ về Provider Contacts; điều này có thể xảy ra nếu người dùng có một tập liên lạc của một tài khoản cá nhân, cùng với một tập liên lạc khác cho công việc. Các tên tài khoản thường là duy nhất. Cùng lúc, tên tài khoản và loại tài khoản nhận diện một luồng dữ liệu xác định giữa Provider Contacts và một service bên ngoài.

Nếu muốn chuyển dữ liệu trên service bạn sử dụng về Provider Contacts, bạn cần viết adapter có chức năng đồng bộ của chính mình. Điều này sẽ được mô tả chi tiết trong mục "Adapter có chức năng đồng bộ của Provider Contacts".

Hình 4 minh họa cách bố trí Provider Contacts trong luồng dữ liệu về con người. Trong các khung chữ nhật được đánh dấu là "sync adapters" (adapter có chức năng đồng bộ), mỗi adapter được gán nhãn bằng loại tài khoản của nó.



People app	Ứng dụng của mọi người
App	Ứng dụng
Contacts provider	Provider Contacts
Sync adapters	adapter có chức năng đồng bộ

Hình 4. Luồng dữ liệu của Provider Contacts.

Các quyền bắt buộc

Ứng dụng muốn truy cập Provider Contacts phải yêu cầu các quyền sau:

Quyền đọc tới một hoặc nhiều bảng

> Quyền `READ_CONTACTS`, được xác định trong file `AndroidManifest.xml` với phần tử `<use-permission>` ví dụ `<uses-permission android:name="android.permission.READ_CONTACTS">`.

Quyền ghi tới một hoặc nhiều bảng

> Quyền `WRITE_CONTACTS`, được xác định trong file `AndroidManifest.xml` bằng phần tử `<uses-permission>` như là `<uses-permission android:name="android.permission.WRITE_CONTACTS">`.

Những quyền này không áp dụng cho user profile (hồ sơ thông tin người dùng). User profile và các quyền bắt buộc của nó được miêu tả chi tiết ở mục bên dưới, "User Profile".

Hãy nhớ rằng, dữ liệu liên lạc của người dùng là thông tin cá nhân, có tính chất nhạy cảm. Người dùng lo ngại về tính riêng tư, nên họ không muốn các ứng dụng thu thập dữ liệu về những thông tin này hay các liên lạc của họ. Nếu không đưa ra lý do rõ ràng về việc tại sao bạn cần quyền truy cập vào dữ liệu liên lạc của người dùng, họ sẽ đánh giá thấp ứng dụng của bạn hoặc đơn giản là từ chối cài đặt ứng dụng này.

User Profile

Bảng ContactsContract.Contacts có một hàng chứa dữ liệu về profile cho người dùng thiết bị. Dữ liệu này mô tả người dùng thiết bị, chứ không phải là một trong những liên lạc của họ. Hàng profile được liên kết với hàng dữ liệu thô tương ứng của từng hệ thống sử dụng profile. Mỗi profile thô có thể chứa nhiều hàng dữ liệu. Các hằng để truy cập user profile nằm trong lớp ContactsContract.Profile.

Truy cập vào user profile cần các quyền đặc biệt. Ngoài quyền READ_CONTACTS và WRITE_CONTACTS để đọc và ghi, bạn còn cần thêm quyền READ_PROFILE và WRITE_PROFILE để truy cập đọc và ghi vào user profile.

Hãy nhớ rằng, bạn phải tính đến thực tế rằng user profile là dữ liệu nhạy cảm. Quyền READ_PROFILE cho phép bạn truy cập dữ liệu cá nhân của người dùng thiết bị. Hãy đảm bảo bạn đã thông báo đầy đủ tới người dùng về lý do bạn cần quyền truy cập vào user profile trong phần mô tả của ứng dụng mà bạn phát triển.

Để truy xuất vào hàng chứa user profile trong bảng ContactsContract.Contacts, gọi phương thức ContentResolver.query(). Thiết lập content URI là CONTENT_URI và không cung cấp bất cứ điều kiện chọn nào. Bạn cũng có thể sử dụng content URI làm URI cơ sở để truy xuất dữ liệu liên lạc thô hoặc dữ liệu cho user profile. Ví dụ, đoạn mã nhỏ dưới đây truy xuất dữ liệu profile:

```
// Thiết lập các cột truy xuất để lấy user profile
mProjection =newString[]
    {
        Profile._ID,
        Profile.DISPLAY_NAME_PRIMARY,
        Profile.LOOKUP_KEY,
        Profile.PHOTO_THUMBNAIL_URI
    };

// Truy xuất profile trong Provider Contacts
mProfileCursor =
        getContentResolver().query(
                Profile.CONTENT_URI,
                mProjection ,
                null,
                null,
                null);
```

> **Ghi chú:** Nếu bạn truy xuất nhiều hàng từ bảng ContactsContract.Contacts, đồng thời muốn xác định xem một trong những hàng này có chứa user profile hay không, hãy kiểm thử cột IS_USER_PROFILE của hàng. Nếu cột này được thiết lập là "1", liên lạc này chính là user profile.

Siêu dữ liệu của Provider Contacts

Provider Contacts quản lý dữ liệu lần theo vết (track) trạng thái dữ liệu liên lạc ở kho lưu trữ. Siêu dữ liệu về kho lưu trữ được lưu ở nhiều vị trí khác nhau, bao gồm các hàng của bảng `Raw Contacts`, `Data`, `Contacts`, bảng `ContactsContract.Settings` và bảng `ContactsContract.SyncState`. Bảng dưới đây minh họa tác dụng của mỗi phần siêu dữ liệu này:

Bảng 3. Siêu dữ liệu trong Provider Contacts.

Bảng	Cột	Giá trị	Ý nghĩa
`ContactsContract.RawContacts`	`DIRTY`	"0" - không thay đổi kể từ lần đồng bộ cuối.	Đánh dấu dữ liệu liên lạc thô đã có thay đổi trên thiết bị và cần được đồng bộ trở lại server. Giá trị này được Provider Contacts thiết lập tự động khi ứng dụng Android cập nhật một hàng. Adapter có chức năng đồng bộ chỉnh sửa dữ liệu liên lạc thô hoặc bảng dữ liệu luôn phải gắn chuỗi `CALLER_IS_SYNCADAPTER` vào content URI mà adapter này sử dụng. Điều này tránh cho provider khỏi việc đánh dấu hàng là dirty (chưa được cập nhật). Nếu không, việc chỉnh sửa của adapter có chức năng đồng bộ chỉ mang nghĩa là thay đổi cục bộ và được gửi tới server, ngay cả khi server là nguồn chỉnh sửa.
		"1" - đã thay đổi kể từ lần đồng bộ cuối, cần được đồng bộ trở lại server.	
`ContactsContract.RawContacts`	`VERSION`	Số chỉ phiên bản của hàng này.	Provider Contacts tự động tăng giá trị này mỗi khi có thay đổi trên hàng, hoặc dữ liệu liên quan đến hàng này thay đổi.
`ContactsContract.Data`	`DATA_VERSION`	Số chỉ phiên bản của hàng này.	Provider Contacts tự động tăng giá trị này vào bất cứ thời điểm nào có thay đổi trên hàng dữ liệu.

Bảng	Cột	Giá trị	Ý nghĩa
ContactsContract.RawContacts	SOURCE_ID	Giá trị chuỗi định danh duy nhất cho một dữ liệu liên lạc thô với tài khoản tạo ra dữ liệu này.	Khi một adapter có chức năng đồng bộ tạo ra một hàng liên lạc thô mới, cột này sẽ được thiết lập ID duy nhất của server đó cho hàng này. Khi một ứng dụng Android tạo một dữ liệu liên lạc thô mới, ứng dụng sẽ để trống cột này. Đây chính là dấu hiệu báo cho adapter có chức năng đồng bộ biết rằng adapter này nên tạo một dữ liệu liên lạc thô mới trên server, và lấy một giá trị cho SOURCE_ID. Cụ thể, id nguồn phải là **duy nhất** đối với mỗi loại tài khoản và nên cố định qua các lần đồng bộ: • Duy nhất (unique): Mỗi dữ liệu liên lạc thô của một tài khoản phải có id nguồn cho dữ liệu đó. Nếu không tuân theo tính chất này, bạn sẽ để lại vấn đề trong ứng dụng liên lạc. Lưu ý, hai dữ liệu liên lạc thô của cùng *loại* tài khoản có thể có cùng id nguồn. Ví dụ, dữ liệu liên lạc thô "Thomas Higginson" của tài khoản `emily.dickinson@gmail.com` được phép có cùng id nguồn với dữ liệu liên lạc thô "Thomas Higginson" của tài khoản `emilyd@gmail.com`. • Ổn định (stable): Các id nguồn là phần bền vững của dữ liệu service trực tuyến cho dữ liệu liên lạc thô. Ví dụ, nếu người dùng xóa bộ nhớ lưu trữ thông tin liên lạc (Contacts Storage) từ nơi cài đặt ứng dụng, sau đó tái đồng bộ, dữ liệu liên lạc thô được khôi phục lại sẽ có cùng id nguồn như trước đó. Nếu bạn không tuân theo tính chất này, biểu tượng shortcut sẽ ngừng hoạt động.

Bảng	Cột	Giá trị	Ý nghĩa
ContactsContract.Groups	GROUP_VISIBLE	Giá trị "0" - Các liên lạc trong nhóm này không được hiển thị (visible) trong giao diện người dùng của ứng dụng Android.	Cột này là để tương thích với server, cho phép người dùng ẩn các liên lạc trong nhóm cụ thể.
		"1" - Các liên lạc trong nhóm này được hiển thị trong giao diện người dùng của ứng dụng.	
ContactsContract.Settings	UNGROUPED_VISIBLE	"0" - Với tài khoản và loại tài khoản này, các liên lạc không nằm trong nhóm sẽ ở chế độ ẩn (invisible) với giao diện người dùng ứng dụng Android.	Mặc định, các liên lạc bị ẩn nếu không có liên lạc thô nào của chúng thuộc về một nhóm (thông tin thành viên nhóm của một dữ liệu liên lạc thô được một hoặc nhiều hàng ContactsContract.CommonDataKinds.GroupMembership trong bảng ContactsContract.Data chỉ ra). Bằng cách thiết lập cờ này trong hàng của bảng ContactsContract.Settings cho một tài khoản và một loại tài khoản, bạn có thể quy ước những liên lạc không cần nhóm được hiển thị. Một tác dụng của cờ này là hiển thị liên lạc từ server mà không cần sử dụng nhóm.
		"1" - Với tài khoản và loại tài khoản này, các liên lạc không nằm trong nhóm được hiển thị trong giao diện người dùng ứng dụng.	

Bảng	Cột	Giá trị	Ý nghĩa
ContactsContract.SyncState	(tất cả các cột)	Sử dụng bảng này để lưu siêu dữ liệu cho adapter có chức năng đồng bộ.	Với bảng này, bạn có thể lưu lại trạng thái đồng bộ và các dữ liệu liên quan đến đồng bộ khác một cách lâu dài trên thiết bị.

Truy cập Provider Contacts

Mục này giới thiệu các bước truy cập dữ liệu từ Provider Contacts, như sau:

- Truy vấn thực thể.
- Chỉnh sửa hàng loạt.
- Tìm kiếm và chỉnh sửa bằng intent.
- Thực hiện toàn vẹn dữ liệu.

Việc thực hiện chỉnh sửa từ adapter có chức năng đồng bộ sẽ được giới thiệu chi tiết hơn trong mục "Adapter có chức năng đồng bộ của Provider Contacts".

Các thực thể truy vấn

Do các bảng của Provider Contacts được tổ chức trong một hệ thống phân cấp, nên việc truy vấn một hàng và tất cả các hàng "con" có liên kết với nó rất thuận lợi. Ví dụ, để hiển thị toàn bộ thông tin về một người, bạn có thể muốn truy xuất tất cả các hàng của bảng `ContactsContract.RawContacts` có liên quan đến một hàng trong bảng ContactsContract.Contacts, hoặc tất cả các hàng trong bảng `ContactsContract.CommonDataKinds.Email` có liên quan đến một hàng trong bảng `ContactsContract.RawContacts`. Để thực hiện điều này một cách dễ dàng, Provider Contacts cung cấp cấu trúc **thực thể (entity)**, đóng vai trò tương tự những liên kết cơ sở dữ liệu giữa các bảng.

Một thực thể giống như một bảng chứa các cột được chọn từ bảng cha và bảng con của nó. Khi truy vấn một thực thể, bạn cung cấp một phép chiếu và điều kiện chọn dựa trên những cột có sẵn trong thực thể. Kết quả sẽ là một `Cursor` chứa một hàng cho mỗi hàng của bảng con được truy xuất. Ví dụ, nếu truy vấn `ContactsContract.Contacts.Entity` để lấy một tên liên lạc và tất cả hàng `ContactsContract.CommonDataKinds.Email` của toàn bộ dữ liệu liên lạc thô có tên này, bạn sẽ nhận về một `Cursor` chứa một hàng cho mỗi hàng trong bảng `ContactsContract.CommonDataKinds.Email`.

Các thực thể giúp đơn giản hóa truy vấn. Sử dụng một thực thể, bạn có thể cùng lúc truy xuất tất cả dữ liệu liên lạc của một liên lạc hoặc dữ liệu liên lạc thô, thay vì phải truy vấn trên bảng cha trước để lấy ID, sau đó thực hiện truy vấn bảng con bằng ID này. Ngoài ra, Provider Contacts thực hiện truy vấn trên một thực thể trong một giao dịch (transaction) đơn, điều này đảm bảo dữ liệu truy xuất có cấu trúc ổn định.

> **Ghi chú:** Một thực thể thường không chứa tất cả các cột của bảng cha và bảng con. Nếu thử làm việc với một tên cột không nằm trong danh sách hằng tên cột của thực thể, bạn sẽ nhận về một `Exception` (ngoại lệ).

Đoạn mã nhỏ sau minh họa cách truy xuất tất cả các hàng dữ liệu thô của một liên lạc. Đoạn mã nhỏ này là một phần trong một ứng dụng lớn hơn, ứng dụng gồm hai activity là "activity chính" và "activity chi tiết". Activity chính hiển thị danh sách các hàng liên lạc; khi người dùng chọn một hàng, activity này sẽ gửi lại ID của hàng đó tới activity chi tiết. Activity chi tiết sử dụng `ContactsContract.Contacts.Entity` để hiển thị tất cả các hàng dữ liệu từ những liên lạc thô tương ứng với liên lạc đã chọn.

Đoạn mã nhỏ dưới đây minh họa "activity chi tiết":

```
...
    /*
    * Thêm đường dẫn thực thể vào URI. Trong trường hợp của
    * Provider Contacts, URI được khuyên dùng là
    * content://com.google.contacts/#/entity (# là giá trị ID).
    */
    mContactUri =Uri.withAppendedPath(
           mContactUri,
           ContactsContract.Contacts.Entity.CONTENT_DIRECTORY);

    // Khởi tạo loader có LOADER_ID xác định.
    getLoaderManager().initLoader(
           LOADER_ID,     // ID của loader dùng để khởi tạo
           null,          // Các đối số của loader (trong trường hợp
                          // này là không có đối số)
           this);         // Ngữ cảnh (context) của activity

    // Tạo một con trỏ adapter mới để gắn vào list view
    mCursorAdapter == new SimpleCursorAdapter(
            this,                       // ngữ cảnh của activity
            R.layout.detail_list_item,  // mục hiển thị chứa widget
             mCursor,                   // chi tiết con trỏ được trả về
             mFromColumns,              // các cột trong con trỏ sẽ
                                        // cung cấp dữ liệu
             mToViews,                  // các view của mục view
                                        // hiển thị dữ liệu
             0);                        // cờ

    // Thiết lập adapter trả về cho ListView.
    mRawContactList.setAdapter(mCursorAdapter);
```

```
...
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    /*
    * Thiết lập cột lấy dữ liệu.
    * RAW_CONTACT_ID được đưa vào để xác định dữ liệu
    * liên lạc thô liên kết với hàng dữ liệu.
    * DATA1 chứa cột đầu tiền của hàng dữ liệu
    * (thường là cột quan trọng nhất).
    * MIMETYPE chỉ ra loại dữ liệu của hàng dữ liệu.
    */
    String[] projection =
        {
            ContactsContract.Contacts.Entity.RAW_CONTACT_ID,
            ContactsContract.Contacts.Entity.DATA1,
            ContactsContract.Contacts.Entity.MIMETYPE
        };
     /*
    * Sắp xếp trên con trỏ truy xuất được theo id của dữ liệu liên lạc
    * thô sao cho tất cả các hàng dữ liệu của một dữ liệu liên lạc thô
    * được  xếp lại cùng nhau.
    */
    String sortOrder =
            ContactsContract.Contacts.Entity.RAW_CONTACT_ID +
            " ASC";
    /*
    * Trả về CursorLoader mới. Đối số tương tự đối số của phương thức
    * ContentResolver.query(), ngoại trừ đối số Context (ngữ cảnh),
    * cung cấp vị trí sử dụng của ContentResolver.
    */
    return new CursorLoader(
            getApplicationContext(), // Ngữ cảnh của activity
            mContactUri,             // Content URI thực thể của một
                                     // liên lạc đơn
            projection,              // Các cột lấy giá trị
            null,                    // Truy xuất tất cả các liên lạc
                                     // thô và các hàng dữ liệu của chúng.
            null,                    //
            sortOrder);              // Sắp xếp theo ID của dữ liệu
                                     // liên lạc thô.
}
```

Khi đã hoàn thành việc nạp dữ liệu, `LoaderManager` (trình quản lý việc nạp) gọi phương thức callback `onLoadFinished()`. Một trong những đối số của phương thức này là một `Cursor` kết quả của truy vấn. Trong ứng dụng của mình, bạn có thể lấy dữ liệu từ `Cursor` này để hiển thị nó hoặc thực hiện các thao tác khác.

Chỉnh sửa hàng loạt

Bất cứ khi nào có thể, bạn nên chèn, cập nhật và xóa dữ liệu của Provider Contacts trong "chế độ chỉnh sửa hàng loạt", bằng cách tạo một `ArrayList` cho các đối tượng `ContentProviderOperation` và gọi `applyBatch()`. Do Provider Contacts thực hiện toàn bộ các thao tác của một `applyBatch()` trong một giao dịch, nên việc chỉnh sửa sẽ không bao giờ làm kho lưu trữ liên lạc rơi vào tình trạng thiếu đồng nhất. Chỉnh sửa hàng loạt cũng giúp chèn cùng lúc dữ liệu thô và dữ liệu chi tiết một cách dễ dàng.

> **Ghi chú:** Để chỉnh sửa dữ liệu liên lạc thô dạng đơn, cần tính đến việc gửi một intent tới ứng dụng Contacts (liên lạc) của thiết bị thay vì xử lý chỉnh sửa trong ứng dụng của bạn. Cách làm này được miêu tả chi tiết trong mục "Truy vấn và chỉnh sửa bằng intent".

Các điểm trung chuyển

Chỉnh sửa hàng loạt chứa số lượng thao tác lớn có thể chặn các tiến trình khác, về toàn cục sẽ khiến người dùng không thoải mái. Để tổ chức tất cả các chỉnh sửa bạn muốn thành một vài danh sách tách biệt, đồng thời tránh làm chậm hệ thống, bạn nên đặt một vài **điểm trung chuyển (yield point)** cho một hoặc nhiều thao tác. Điểm trung chuyển là đối tượng `ContentProviderOperation` có giá trị `isYieldAllowed()` thiết lập là `true`. Khi Provider Contacts gặp một điểm trung chuyển, nó dừng thao tác để các tiến trình khác chạy và đóng giao dịch hiện tại lại. Khi content provider khởi động trở lại, provider này sẽ tiếp tục với thao tác tiếp theo trong `ArrayList` và bắt đầu tiến hành một giao dịch mới.

Điểm trung chuyển tạo ra nhiều giao dịch trên mỗi lời gọi phương thức `applyBatch()`. Do vậy, bạn nên thiết lập điểm trung chuyển cho thao tác cuối của tập các hàng liên kết. Ví dụ, bạn nên thiết lập cho thao tác cuối một điểm trung chuyển cho tập lệnh thêm các hàng dữ liệu liên lạc thô và những hàng dữ liệu liên quan của nó, hoặc thao tác cuối cho tập các hàng liên hệ với một liên lạc đơn.

Điểm trung chuyển cũng là một đơn vị thao tác đơn. Tất cả các truy cập giữa hai điểm trung chuyển sẽ thành công hoặc thất bại giống như một đơn vị đơn. Nếu bạn không thiết lập bất cứ điểm trung chuyển nào, thao tác đơn nhỏ nhất sẽ là toàn bộ thao tác của nhóm lệnh (batch). Nếu sử dụng điểm trung chuyển, bạn sẽ tránh được các thao tác làm giảm hiệu suất của hệ thống, trong khi vẫn đảm bảo được rằng tập con của các thao tác là đơn.

Tham chiếu lùi cho thao tác chỉnh sửa

Khi đang chèn một hàng liên lạc thô mới và những hàng dữ liệu liên quan dưới dạng một tập đối tượng `ContentProviderOperation`, bạn phải liên kết các hàng dữ liệu với hàng dữ liệu liên lạc thô bằng cách chèn giá trị `_ID` của dữ liệu liên lạc thô làm giá trị `RAW_CONTACT_ID`. Tuy nhiên, giá trị này không sẵn có khi bạn tạo đối tượng `ContentProviderOperation` cho hàng dữ liệu, do bạn vẫn chưa áp dụng `ContentProviderOperation` cho hàng dữ liệu thô. Để thực hiện

điều này, lớp ContentProviderOperation.Builder cung cấp phương thức withValueBackReference(). Phương thức này cho phép bạn chèn hoặc chỉnh sửa một cột trên kết quả của thao tác trước đó.

Phương thức withValueBackReference() có hai đối số:

key

Khóa của cặp khóa-giá trị. Giá trị của đối số này phải là tên một cột trong bảng bạn đang chỉnh sửa.

previousResult

Chỉ số bắt đầu từ 0 của một giá trị trong mảng các đối tượng ContentProviderResult trả về từ phương thức applyBatch(). Khi các thao tác của nhóm được áp dụng, kết quả của mỗi thao tác được lưu trong một mảng kết quả trung gian. Giá trị previousResult là chỉ số của một trong những kết quả này, được truy xuất và lưu trữ bằng giá trị key. Điều này cho phép bạn chèn một bản ghi liên lạc thô mới và lấy về giá trị _ID của bản ghi này, sau đó thực hiện một "tham chiếu lùi" ("back reference") giá trị khi bạn thêm hàng ContactsContract.Data.

Mảng chứa toàn bộ kết quả được tạo ra khi bạn gọi phương thức applyBatch() lần đầu, với kích thước bằng kích thước ArrayList của các đối tượng ContentProviderOperation mà bạn cung cấp. Tuy nhiên, tất cả các phần tử của mảng kết quả được thiết lập là null, và nếu bạn cố gắng làm một tham chiếu lùi tới kết quả của một thao tác chưa được áp dụng, phương thức withValueBackReference() ném ra một ngoại lệ Exception.

Đoạn mã nhỏ sau minh họa cách chèn một dữ liệu liên lạc thô mới và dữ liệu trong nhóm. Đoạn mã này bao gồm mã thiết lập một điểm trung chuyển và sử dụng một tham chiếu lùi. Đoạn mã sau là phiên bản mở rộng của phương thức createContactEntry(), là một phần của lớp ContactAdder trong ứng dụng mẫu Contact Manager.

Đoạn mã đầu truy xuất dữ liệu liên lạc từ giao diện người dùng. Lúc này, người dùng đã chọn một tài khoản để thêm dữ liệu thô mới vào.

```
// Tạo một mục liên lạc từ các giá trị trên giao diện người dùng
// hiện tại, sử dụng tài khoản đang được chọn.
protected void createContactEntry() {
    /*
    * Lấy các giá trị từ giao diện người dùng
    */
    String name = mContactNameEditText.getText().toString();
    String phone = mContactPhoneEditText.getText().toString();
    String email = mContactEmailEditText.getText().toString();

    int phoneType = mContactPhoneTypes.get(
            mContactPhoneTypeSpinner.getSelectedItemPosition());
    int emailType = mContactEmailTypes.get(
            mContactEmailTypeSpinner.getSelectedItemPosition());
```

Đoạn mã nhỏ tiếp theo sẽ tạo một thao tác chèn hàng dữ liệu liên lạc thô vào bảng ContactsContract.RawContacts:

```
    /*
    * Chuẩn bị nhóm lệnh (batch) để thực hiện thao tác chèn một dữ
    * liệu liên lạc thô mới và các dữ liệu của nó. Ngay cả khi Provider
    * Contacts không có bất cứ dữ liệu nào của người này, bạn cũng
    * không thể thêm vào một Contact, mà chỉ có thể thêm vào một dữ
    * liệu liên lạc thô. Sau đó, Provider Contacts sẽ tự động
    * thêm Contact tương ứng.
    */

    // Tạo một mảng các đối tượng ContentProviderOperation mới.
    ArrayList<ContentProviderOperation> ops =
        new ArrayList<ContentProviderOperation>();

    /*
    * Tạo một dữ liệu liên lạc thô mới với loại tài khoản (loại server)
    * của dữ liệu thô đó và tên tài khoản (tài khoản của người dùng).
    * Hãy nhớ rằng tên hiển thị không được lưu trong hàng này, mà là  trong
    * hàng dữ liệu StructuredName. Không yêu cầu thêm dữ liệu khác.
    */
    ContentProviderOperation.Builder op =
            ContentProviderOperation.newInsert(ContactsContract.
RawContacts.CONTENT_URI)
            .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE,
mSelectedAccount.getType())
            .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
mSelectedAccount.getName());

    // Xây dựng thao tác và thêm thao tác đó vào mảng các thao tác
    ops.add(op.build());
```

Tiếp theo, đoạn mã tạo ra các hàng dữ liệu cho tên hiển thị, số điện thoại và địa chỉ e-mail.

Mỗi đối tượng trình xây dựng thao tác (operation builder) sử dụng phương thức withValueBackReference() để lấy về RAW_CONTACT_ID. Các điểm tham chiếu quay về đối tượng ContentProviderResult từ thao tác đầu tiên, là thao tác thêm hàng dữ liệu thô và trả về giá trị _ID mới của hàng này. Kết quả là, mỗi hàng dữ liệu được tự động liên kết bởi giá trị RAW_CONTACT_ID với hàng ContactsContract.RawContacts mới của dữ liệu này.

Đối tượng ContentProviderOperation.Builder thêm hàng địa chỉ e-mail được đánh dấu bằng cờ thông qua phương thức withYieldAllowed(), là phương thức thiết lập điểm trung chuyển:

```
    // Tạo tên hiển thị cho liên lạc thô mới,
    // là một hàng dữ liệu StructuredName.
    op = ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
            /*
            * Phương thức withValueBackReference thiết lập giá trị của
            * đối số đầu tiên là giá trị của ContentProviderResult được
            * đánh chỉ mục bởi đối số thứ hai. Trong lời gọi cụ thể này,
            * cột ID dữ liệu liên lạc thô của hàng dữ liệu StructuredName
            * được thiết lập là giá trị trả về của thao tác đầu * tiên,
            cũng chính là thao tác thực sự thêm hàng dữ liệu * thô mới.
            */
            .withValueBackReference(ContactsContract.Data.RAW_
CONTACT_ID,0)
            // Thiết lập kiểu MIME của hàng dữ liệu là StructuredName
            .withValue(ContactsContract.Data.MIMETYPE,
                ContactsContract.CommonDataKinds.StructuredName.
CONTENT_ITEM_TYPE)
            // Thiết lập tên hiển thị của hàng dữ liệu là tên trong giao
            // diện người dùng.
            .withValue(ContactsContract.CommonDataKinds.
StructuredName.DISPLAY_NAME, name);
    // Xây dựng thao tác và thêm thao tác đó vào mảng các thao tác
     ops.add(op.build());
    // Chèn số điện thoại cụ thể và loại làm hàng dữ liệu Phone
    // (điện thoại)
    op = ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
            /*
            * Thiết lập giá trị của cột id liên lạc thô là giá trị ID
            liên lạc thô được thao tác đầu tiên của nhóm trả về.
            */
            .withValueBackReference(ContactsContract.Data.RAW_
CONTACT_ID,0)
            // Thiết lập kiểu MIME của hàng dữ liệu là Phone
            .withValue(ContactsContract.Data.MIMETYPE,
                ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
              // Thiết lập số điện thoại và kiểu
            .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phone)
            .withValue(ContactsContract.CommonDataKinds.Phone.TYPE, phoneType);
    // Xây dựng thao tác và thêm thao tác đó vào mảng các thao tác
    ops.add(op.build());
```

```
    // Chèn địa chỉ e-mail cụ thể và kiểu làm một hàng dữ liệu Email
    op = ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
           /*
           * Thiết lập giá trị của cột id dữ liệu thô là ID dữ liệu
           thô mới được trả về từ thao tác đầu tiên của nhóm.
             */
           .withValueBackReference(ContactsContract.Data.RAW_
CONTACT_ID,0)
           // Thiết lập kiểu MIME cho hàng dữ liệu là Email
           .withValue(ContactsContract.Data.MIMETYPE,
           ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
           // Thiết lập địa chỉ e-mail và kiểu
           .withValue(ContactsContract.CommonDataKinds.Email.
ADDRESS, email)
           .withValue(ContactsContract.CommonDataKinds.Email.TYPE,
emailType);
    /*
    * Minh họa cách sử dụng điểm trung chuyển. Ở cuối thao tác chèn
    * này, luồng nhóm thao tác sẽ chuyển ưu tiên cho các luồng khác.
    * Sử dụng điểm trung chuyển này sau mỗi tập thao tác liên quan đến
    * một liên lạc đơn, để tránh làm giảm hiệu suất hệ thống.
    */
    op.withYieldAllowed(true);
    // Xây dựng thao tác và thêm thao tác đó vào mảng các thao tác
    ops.add(op.build());
```

Đoạn mã nhỏ cuối cùng minh họa lời gọi phương thức `applyBatch()` để chèn liên lạc thô mới và các hàng dữ liệu.

```
    // Yêu cầu Provider Contacts tạo một liên lạc mới
    Log.d(TAG,"Selected account: "+ mSelectedAccount.getName()+" ("+
           mSelectedAccount.getType()+")");
    Log.d(TAG,"Creating contact: "+ name);
    /*
    * Đưa một mảng các đối tượng ContentProviderOperation vào nhóm lệnh.
    * Kết quả là bị bỏ qua.
    */
    try {
           getContentResolver().applyBatch(ContactsContract.
AUTHORITY, ops);
    }catch(Exception e){
```

```
            // Hiển thị lời cảnh báo (warning)
            Context ctx = getApplicationContext();

            CharSequence txt =
getString(R.string.contactCreationFailure);
            int duration =Toast.LENGTH_SHORT;
            Toast toast =Toast.makeText(ctx, txt, duration);
            toast.show();

            // Ghi ngoại lệ
            Log.e(TAG,"Exception encountered while inserting contact: "+ e);
    }
}
```

Các nhóm thao tác cũng cho phép bạn cài đặt **kiểm soát truy cập đồng thời "lạc quan" (optimistic concurrency control)**, một phương thức áp dụng cho những giao dịch chỉnh sửa mà không phải chặn kho lưu trữ bên dưới. Để sử dụng phương thức này, bạn áp dụng giao dịch, sau đó kiểm tra xem các chỉnh sửa khác có diễn ra cùng thời điểm đó không. Nếu tìm thấy có chỉnh sửa không nhất quán, bạn roll back giao dịch (hủy các thao tác của giao dịch đã thực hiện) của mình lại và thử lại sau.

Kiểm soát truy cập đồng thời "lạc quan" rất hữu dụng đối với thiết bị di động, thường chỉ có một người dùng tại một thời điểm, và hiếm khi xảy ra các truy cập đồng thời tới kho lưu trữ dữ liệu. Bởi vậy, việc chặn (locking) không xảy ra thường xuyên, do đó không tốn thời gian để thiết lập khóa chặn và chờ các giao dịch khác giải phóng khóa của chúng.

Để sử dụng Kiểm soát truy cập đồng thời "lạc quan" trong khi đang cập nhật một hàng `ContactsContract.RawContracts`, bạn thực hiện theo các bước sau:

1. Truy xuất cột `VERSION` của dữ liệu liên lạc thô cùng với những dữ liệu khác mà bạn muốn truy xuất.
2. Tạo một đối tượng `ContentProviderOperation.Builder` phù hợp cho việc thực hiện một ràng buộc (constraint), bằng cách dùng phương thức `newAssertQuery(Uri)`. Đối với content URI, sử dụng `RawContacts.CONTENT_URI` được gắn thêm `_ID` của dữ liệu liên lạc thô.
3. Với đối tượng `ContentProviderOperation.Builder`, gọi phương thức `withValue()` để so sánh cột `VERSION` với số hiệu phiên bản bạn vừa truy xuất.
4. Trên cùng đối tượng `ContentProviderOperation.Builder`, gọi phương thức `withExpectedCount()` để đảm bảo chỉ có duy nhất một hàng được xác nhận (assertion) này.
5. Gọi phương thức `build()` để tạo đối tượng `ContentProviderOperation`, sau đó thêm đối tượng này làm đối tượng đầu tiên của `ArrayList`, là mảng sẽ chuyển đến phương thức `applyBatch()`.
6. Áp dụng giao dịch hàng loạt.

Nếu hàng dữ liệu liên lạc thô được một thao tác khác cập nhật trong thời gian bạn đọc hàng này và thời điểm bạn thử chỉnh sửa hàng, việc "xác nhận" ContentProviderOperation sẽ bị lỗi, toàn bộ nhóm thao tác sẽ bị lùi lại. Sau đó, bạn có thể chọn thử lại nhóm hoặc thực hiện một số thao tác khác.

Đoạn mã nhỏ dưới đây minh họa cách tạo "xác nhận" ContentProviderOperation sau khi truy vấn một dữ liệu liên lạc thô đơn bằng cách sử dụng CursorLoader:

```
/*
 * Ứng dụng dùng CursorLoader để truy vấn bảng dữ liệu liên lạc thô.
 * Hệ thống gọi phương thức này khi đã hoàn thành việc nạp dữ liệu.
 */
public void onLoadFinished(Loader<Cursor> loader,Cursor cursor){

    // Lấy giá trị VERSION và _ID của dữ liệu liên lạc thô
    mRawContactID = cursor.getLong(cursor.getColumnIndex(BaseColumns._ID));
        mVersion = cursor.getInt(cursor.getColumnIndex(SyncColumns.
VERSION));
}

...

// Thiết lập Uri cho thao tác xác nhận
Uri rawContactUri = ContentUris.withAppendedId(RawContacts.CONTENT_URI,
mRawContactID);

// Tạo một trình xây dựng (builder) cho thao tác xác nhận
ContentProviderOperation.Builder assertOp =ContentProviderOperation.
netAssertQuery(rawContactUri);

// Thêm xác nhận cho thao tác xác nhận: Kiểm tra phiên bản và đếm số
// hàng được kiểm tra
assertOp.withValue(SyncColumns.VERSION, mVersion);
assertOp.withExpectedCount(1);

// Tạo một đối tượng ArrayList để lưu các đối tượng
// ContentProviderOperation
ArrayList ops == new ArrayList<ContentProviderOperationg>;

ops.add(assertOp.build());

// Bạn sẽ thêm các thao tác còn lại của nhóm thao tác vào "ops" tại đây

...

// Áp dụng nhóm. Nếu việc xác nhận lỗi, hệ thống sẽ ném ra một ngoại lệ
// Exception
```

```
try
    {
        ContentProviderResult[] results =
                getContentResolver().applyBatch(AUTHORITY, ops);

    }catch(OperationApplicationException e){

        // Hoạt động bạn muốn thực hiện trong trường hợp xảy ra lỗi tại
        // ứng dụng của ContentProviderOperation vì một số ràng buộc cụ thể
    }
```

Truy xuất và chỉnh sửa bằng intent

Gửi một intent tới ứng dụng liên lạc của thiết bị cho phép bạn gián tiếp truy cập vào tProvider Contacts. Intent khởi tạo tại giao diện người dùng ứng dụng liên lạc của thiết bị; tại đây, người dùng có thể thực hiện các thao tác liên quan tới dữ liệu liên lạc. Với kiểu truy cập này, người dùng có thể:

- Lấy ra một liên lạc từ danh sách và trả liên lạc này lại ứng dụng của bạn để sử dụng cho các thao tác sau.
- Chỉnh sửa dữ liệu liên lạc hiện có.
- Chèn một liên lạc thô mới cho bất kỳ tài khoản nào của người dùng.
- Xóa một liên lạc hoặc dữ liệu liên lạc.

Nếu người dùng đang chèn hoặc cập nhật dữ liệu, bạn có thể thu thập dữ liệu này trước, sau đó gửi chúng đi như một phần của intent.

Khi sử dụng intent để truy cập Provider Contactsthông qua ứng dụng liên lạc của thiết bị, bạn không phải xây dựng giao diện người dùng của mình và mã để truy cập provider này. Bạn cũng không phải yêu cầu quyền để đọc và ghi vào provider. Ứng dụng liên lạc của thiết bị có thể cấp quyền đọc một liên lạc cho bạn, và do đang thực hiện chỉnh sửa provider thông qua ứng dụng khác, nên bạn không cần tới quyền ghi.

Tiến trình chung để gửi một intent nhằm truy cập provider được mô tả chi tiết trong mục hướng dẫn "Cơ bản về content provider", phần "Truy cập dữ liệu thông qua intent". Action, kiểu MIME và giá trị dữ liệu bạn sử dụng cho các tác vụ cơ bản được tổng kết trong Bảng 4. Ngoài ra, các giá trị phụ trợ mà bạn có thể sử dụng với phương thức `putExtra()` được liệt kê trong tài liệu tham khảo của `ContactsContract.Intents.Insert`:

Bảng 4. Các intent của Provider Contacts.

Thao tác	Hành động	Dữ liệu	Kiểu MIME
Lấy ra một liên lạc từ danh sách	ACTION_PICK	Một trong các dữ liệu sau: • Contacts.CONTENT_URI, hiển thị danh sách các liên lạc. • Phone.CONTENT_URI, hiển thị danh sách các số điện thoại của một dữ liệu liên lạc thô. • StructuredPostal.CONTENT_URI, hiển thị danh sách các địa chỉ bưu điện của một dữ liệu liên lạc thô. • Email.CONTENT_URI, hiển thị danh sách các địa chỉ e-mail của một dữ liệu liên lạc thô.	Không sử dụng.
Chèn một dữ liệu liên lạc thô mới	Insert.ACTION	Không có.	RawContacts.CONTENT_TYPE, kiểu MIME của tập dữ liệu liên lạc thô.
Chỉnh sửa một liên lạc	ACTION_EDIT	CONTENT_LOOKUP_URI của liên lạc này. Activity chỉnh sửa sẽ cho phép người dùng sửa bất kỳ dữ liệu nào liên kết với liên lạc này.	Contacts.CONTENT_ITEM_TYPE, một liên lạc đơn.
Hiển thị một View cho phép người dùng chọn thông tin (có thể thêm thông tin vào View)	ACTION_INSERT_OR_EDIT	Không có.	CONTENT_ITEM_TYPE

Ứng dụng liên lạc của thiết bị không cho phép bạn xóa một dữ liệu liên lạc thô hay bất kỳ dữ liệu nào của liên lạc này bằng intent. Thay vào đó, ứng dụng cho phép bạn xóa dữ liệu liên lạc thô bằng cách sử dụng `ContentResolver.delete()` hoặc `ContentProviderOperation.newDelete()`.

Đoạn mã nhỏ dưới đây minh họa cách xây dựng và gửi một intent có chức năng chèn dữ liệu liên lạc thô và dữ liệu mới:

```
// Lấy giá trị từ giao diện người dùng
String name = mContactNameEditText.getText().toString();
String phone = mContactPhoneEditText.getText().toString();
String email = mContactEmailEditText.getText().toString();

String company = mCompanyName.getText().toString();
String jobtitle = mJobTitle.getText().toString();

// Tạo một intent mới để gửi tới ứng dụng liên lạc của thiết bị
Intent insertIntent = new Intent(ContactsContract.Intents.Insert.ACTION);

// Thiết lập kiểu MIME cho intent bạn muốn bằng activity chèn
insertIntent.setType(ContactsContract.RawContacts.CONTENT_TYPE);

// Thiết lập tên liên lạc mới
insertIntent.putExtra(ContactsContract.Intents.Insert.NAME, name);

// Thiết lập tên công ty và nghề nghiệp mới
insertIntent.putExtra(ContactsContract.Intents.Insert.COMPANY,
company);
insertIntent.putExtra(ContactsContract.Intents.Insert.JOB_TITLE,
jobtitle);
/*

 * Minh họa cách thêm các hàng dữ liệu dưới dạng một array list
 * liên kết với khóa DATA
 */

// Định nghĩa một array list chứa các đối tượng
// ContentValues cho mỗi hàng
ArrayList<ContentValues> contactData = new ArrayListt<ContentValues>();

/*
 * Định nghĩa hàng dữ liệu liên lạc thô
 */

// Thiết lập hàng là một đối tượng ContentValues
ContentValues rawContactRow = new ContentValues();

// Thêm loại tài khoản và tên tài khoản cho hàng
```

```
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_TYPE,
mSelectedAccount.getType());
rawContactRow.put(ContactsContract.RawContacts.ACCOUNT_NAME,
mSelectedAccount.getName());

// Đưa hàng vào mảng
contactData.add(rawContactRow);

/*
 * Thiết lập hàng dữ liệu điện thoại
 */

// Thiết lập hàng dưới dạng một đối tượng ContentValues
ContentValues phoneRow = new ContentValues();

// Xác định kiểu MIME cho hàng dữ liệu này (tất cả các hàng dữ liệu
// phải được đánh dấu bằng kiểu của chúng)
phoneRow.put(
        ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
);

// Thêm số điện thoại và kiểu vào hàng
phoneRow.put(ContactsContract.CommonDataKinds.Phone.NUMBER, phone);

// Đưa hàng này vào mảng
contactData.add(phoneRow);

/*
 * Thiết lập hàng dữ liệu e-mail
 */

// Thiết lập hàng e-mail là một đối tượng ContentValues
ContentValues emailRow =newContentValues();

// Xác định kiểu MIME cho hàng dữ liệu này (tất cả các hàng dữ liệu
// phải được đánh dấu bằng kiểu của chúng)
emailRow.put(
        ContactsContract.Data.MIMETYPE,
        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
);

// Thêm địa chỉ e-mail và kiểu của địa chỉ vào hàng
emailRow.put(ContactsContract.CommonDataKinds.Email.ADDRESS, email);

// Đưa hàng vào mảng
contactData.add(emailRow);
```

```
/*
 * Thêm mảng vào phần phụ trợ của intent. Mảng này phải là đối tượng có thể
 * chia nhỏ được (parcelable object) để có thể di chuyển giữa các tiến
 * trình. Ứng dụng liên lạc của thiết bị mong muốn khóa của mảng này phải
 * là Intents.Insert.DATA
 */
insertIntent.putParcelableArrayListExtra(ContactsContract.Intents.
Insert.DATA, contactData);

// Gửi intent đi để khởi động ứng dụng liên lạc của
// thiết bị trong activity thêm liên lạc của ứng dụng này.
startActivity(insertIntent);
```

Tính toàn vẹn của dữ liệu

Do kho lưu trữ dữ liệu liên lạc chứa các dữ liệu quan trọng và nhạy cảm mà người dùng muốn phải chính xác và được cập nhật mới nhất, nên Provider Contactschứa các quy tắc được định nghĩa rõ ràng để thực hiện toàn vẹn dữ liệu. Nhiệm vụ của bạn là tuân theo những quy tắc này khi tiến hành chỉnh sửa dữ liệu liên lạc. Các quy tắc quan trọng được liệt kê như dưới đây:

Luôn luôn thêm một hàng `ContactsContract.CommonDataKinds.StructuredName` cho mỗi hàng `ContactsContract.RawContacts` bạn thêm vào.

> Một hàng `ContactsContract.RawContacts` không có hàng `ContactsContract.CommonDataKinds.StructuredName` trong bảng `ContactsContract.Data` có thể gây ra vấn đề trong quá trình tổng hợp.

Luôn luôn liên kết các hàng `ContactsContract.Data` mới với hàng cha `ContactsContract.RawContacts`.

> Một hàng `ContactsContract.Data` không được liên kết với hàng `ContactsContract.RawContacts` sẽ không hiển thị trong ứng dụng liên lạc của thiết bị và có thể gây ra vấn đề với các adapter có chức năng đồng bộ.

Chỉ thay đổi dữ liệu của những dữ liệu liên lạc thô mà bạn quản lý.

> Hãy nhớ rằng, Provider Contactsthường quản lý dữ liệu của một vài service trực tuyến/loại tài khoản khác nhau. Bạn cần đảm bảo ứng dụng của mình chỉ chỉnh sửa và xóa dữ liệu cho các hàng của mình, cũng như chỉ chèn dữ liệu chứa loại tài khoản và tên mà bạn quản lý.

Luôn luôn sử dụng hằng được định nghĩa trong `ContactsContract` và lớp con của nó để thao tác với các ID, content URI, đường dẫn URI, tên cột, kiểu MIME và các giá trị `TYPE`.

Sử dụng những hằng này giúp bạn tránh được lỗi. Bạn cũng nhận được các cảnh báo của trình biên dịch nếu các hằng đã bị loại bỏ (không còn sử dụng được nữa).

Các hàng dữ liệu tùy chỉnh

Bằng cách tạo và sử dụng kiểu MIME tùy chỉnh tự định nghĩa, bạn có thể chèn, chỉnh sửa, xóa và truy xuất các hàng dữ liệu của mình trong bảng `ContactsContract.Data`. Các hàng của bạn bị giới hạn chỉ được phép dùng cột định nghĩa trong `ContactsContract.DataColumns`, mặc dù bạn có thể ánh xạ các Tên cột có kiểu xác định của mình với tên cột mặc định. Trong ứng dụng liên lạc của thiết bị, dữ liệu cho các hàng của bạn được hiển thị nhưng không thể chỉnh sửa hoặc xóa, và người dùng không thể bổ sung thêm dữ liệu. Để cho phép người dùng chỉnh sửa hàng dữ liệu tùy chỉnh (custom data row) do bạn tạo ra, bạn phải cung cấp activity chỉnh sửa trong ứng dụng do mình phát triển.

Để hiển thị dữ liệu tùy chỉnh của bạn, cung cấp file `contacts.xml` chứa một phần tử `<ContactsAccountType>` và một hoặc nhiều phần tử con `<ContactsDataKind>` của phần tử này. Mục "Phần tử `<ContactsDataKind>`" mô tả chi tiết hơn về điều này.

Để tìm hiểu thêm về kiểu MIME tùy chỉnh, đọc mục hướng dẫn "Tạo content provider".

Adapter có chức năng đồng bộ của Provider Contacts

Provider Contactsđược thiết kế chủ yếu để quản lý **việc đồng bộ (synchronization)** dữ liệu liên lạc giữa thiết bị và service trực tuyến. Điều này cho phép người dùng tải dữ liệu hiện có về một thiết bị mới và đẩy (upload) dữ liệu hiện có lên một tài khoản mới. Việc đồng bộ cũng đảm bảo người dùng có được dữ liệu mới nhất, mà không phải quan tâm đến việc nguồn dữ liệu có bổ sung hay chỉnh sửa. Lợi ích khác của việc đồng bộ là dữ liệu liên lạc luôn sẵn sàng ngay cả khi thiết bị không được kết nối mạng.

Mặc dù bạn có thể cài đặt việc đồng bộ theo nhiều cách khác nhau, nhưng hệ thống Android cung cấp một plug-in framework đồng bộ sẽ tự động thực hiện các tác vụ sau:

- Kiểm tra xem mạng có sẵn sàng hay không.
- Lên lịch và thực hiện đồng bộ, dựa trên yêu cầu của người dùng.
- Khởi động lại việc đồng bộ đã bị dừng trước đó.

Để sử dụng framework này, bạn cung cấp một plug-in adapter có chức năng đồng bộ. Mỗi adapter trên chỉ dành cho một service và một content provider, nhưng adapter này có thể xử lý nhiều tên tài khoản của cùng một service. Framework này cũng cho phép nhiều adapter có chức năng đồng bộ thuộc cùng một service và provider.

Các file và lớp của adapter có chức năng đồng bộ

Cài đặt một adapter có chức năng đồng bộ dưới dạng một lớp con của lớp `AbstractThreadedSyncAdapter`, đồng thời cài đặt adapter làm một phần của như là một phần của ứng dụng Android. Hệ thống sẽ tìm hiểu thêm về adapter có chức năng đồng bộ vừa cài đặt từ các phần tử trong file kê khai của ứng dụng, và từ một file XML đặc biệt mà file kê khai trỏ tới. File XML định nghĩa loại tài khoản cho service

trực tuyến và chuỗi định danh cho content provider này, hai thông tin trên cùng xác định duy nhất một adapter. Adapter có chức năng đồng bộ chỉ được kích hoạt cho đến khi người dùng thêm một tài khoản vào loại tài khoản của adapter này và cho phép đồng bộ content provider. Tại điểm này, hệ thống khởi động việc quản lý adapter, gọi phần này khi cần thiết để đồng bộ giữa providervà server.

> **Ghi chú:** Sử dụng loại tài khoản làm một phần để nhận diện adapter có chức năng đồng bộ cho phép hệ thống phát hiện và nhóm những adapter truy cập các service khác nhau từ cùng một tổ chức. Ví dụ, các tài khoản của các adapterr có chức năng đồng bộ của service trực tuyến Google có cùng loại tài khoản là `com.google`. Khi người dùng thêm một tài khoản Google vào thiết bị của họ, tất cả các adapter có chức năng đồng bộ đã cài đặt của service Google được liệt kê cùng nhau; trong đó, mỗi adapter có chức năng đồng bộ được liệt kê cùng với một content provider trên thiết bị.

Do hầu hết service đều yêu cầu người dùng phải xác thực nhận dạng của họ trước khi truy cập dữ liệu, nên hệ thống Android cung cấp framework xác thực tương tự như vậy và cũng thường được sử dụng kết hợp cùng framework adapter có chức năng đồng bộ. Framework xác thực sử dụng trình xác thực plug-in, cũng là lớp con của lớp `AbstractAccountAuthenticator`. Trình xác thực kiểm tra nhận dạng người dùng theo các bước sau:

1. Thu thập tên và mật khẩu người dùng hoặc thông tin tương tự thông tin cá nhân của người dùng.
2. Gửi thông tin cá nhân tới service.
3. Kiểm tra hồi đáp của service.

Nếu service chấp nhận thông tin này, trình xác thực có thể lưu thông tin này lại để sử dụng về sau. Nhờ plug-in framework xác thực, `AccountManager` có thể cung cấp truy cập tới bất kỳ mã thông báo xác thực (authtoken) nào mà trình xác thực hỗ trợ và chọn để cung cấp, chẳng hạn như mã thông báo xác thực `OAuth2`.

Mặc dù việc xác thực không bắt buộc, nhưng hầu hết service liên lạc đều sử dụng yêu cầu này. Tuy nhiên, bạn không bắt buộc phải sử dụng framework xác thực của Android để tiến hành xác thực.

Cài đặt adapter có chức năng đồng bộ

Để cài đặt một adapter có chức năng đồng bộ cho Provider Contacts, bạn bắt đầu bằng việc tạo một ứng dụng Android chứa các thành phần sau:

Một thành phần `Service` phản hồi yêu cầu của hệ thống để gắn vào adapter có chức năng đồng bộ.

Khi hệ thống muốn thực hiện một quá trình đồng bộ, nó gọi phương thức `onBind()` của service để lấy một đối tượng `IBinder` cho adapter có chức năng đồng bộ. Điều này cho phép hệ thống thực hiện lời gọi tiến trình chéo (cross-process call) tới phương thức của adapter.

Trong ứng dụng mẫu Sample Sync Adapter, tên lớp cho service này là `com.example.android.samplesync.syncadapter.SyncService`.

Adapter có chức năng đồng bộ thực sự, được cài đặt như một lớp con cụ thể của lớp AbstractThreadedSyncAdapter.

Lớp này thực hiện tải dữ liệu từ server, đẩy dữ liệu từ thiết bị lên server và giải quyết xung đột. Công việc chính của adapter được thực hiện trong phương thức onPerformSync(). Lớp này phải được khởi tạo dưới dạng một Singleton (lớp chỉ có duy nhất một đối tượng).

Trong ứng dụng mẫu Sample Sync Adapter, adapter có chức năng đồng bộ được định nghĩa tại lớp com.example.android.samplesync.syncadapter.SyncAdapter.

Một lớp con của lớp Application.

Lớp này hoạt động giống nhưnhà máy cho chỉ một adapter duy nhất có chức năng đồng bộ. Sử dụng phương thức onCreate() để khởi tạo adapter, đồng thời cung cấp phương thức truy cập get tĩnh (phương thức được cài đặt để trả về thuộc tính của đối tượng) trả về singleton cho phương thức onBind() của service adapter có chức năng đồng bộ.

Tùy chọn: Thành phần Service phản hồi yêu cầu từ hệ thống để xác thực người dùng.

AccountManager khởi động service này để bắt đầu quá trình xác thực. Phương thức onCreate() của service khởi tạo một đối tượng trình xác thực. Khi hệ thống muốn xác thực một tài khoản người dùng của adapter có chức năng đồng bộ của ứng dụng, hệ thống gọi phương thức onBind() của service này để lấy một IBinder cho trình xác thực. Điều này cho phép hệ thống thực hiện lời gọi tiến trình chéo tới các phương thức của trình xác thực.

Trong ứng dụng mẫu Sample Sync Adapter, tên lớp của service này là com.example.android.samplesync.authenticator.AuthenticationService.

Tùy chọn: Một lớp con cụ thể của lớp AbstractAccountAuthenticator xử lý yêu cầu xác thực.

Lớp này cung cấp các phương thức mà AccountManager sử dụng để xác thực thông tin cá nhân của người dùng với server. Chi tiết về tiến trình xử lý xác thực mỗi nơi một khác, phụ thuộc vào kỹ thuật mà server sử dụng. Bạn nên tham khảo tài liệu về phần mềm server của mình để tìm hiểu thêm về xác thực.

Trong ứng dụng mẫu Sample Sync Adapter, trình xác thực được định nghĩa trong lớp com.example.android.samplesync.authenticator.Authenticator.

File XML định nghĩa adapter có chức năng đồng bộ và trình xác thực của hệ thống.

Adapter có chức năng đồng bộ và các thành phần service của trình xác thực mô tả ở trên được định nghĩa trong phần tử <service> của file kê khai thuộc ứng dụng. Những phần tử này chứa các phần tử con <meta-data> cung cấp dữ liệu cụ thể cho hệ thống:

- Phần tử <meta-data> của service adapter có chức năng đồng bộ trỏ tới file XML res/xml/syncadapter.xml. Nói cách khác, file này xác định một URI cho Web service sẽ được đồng bộ với Provider Contacts, đồng thời xác định một loại tài khoản cho Web service này.

- **Tùy chọn**: Phần tử `<meta-data>` của trình xác thực trỏ tới file XML `res/xml/authenticator.xml`. Nói cách khác, file XML trên xác định loại tài khoản mà trình xác thực này hỗ trợ, cũng như nguồn tài nguyên giao diện người dùng xuất hiện trong suốt tiến trình xử lý xác thực. Loại tài khoản được xác định trong phần tử này phải giống với loại tài khoản được xác định cho adapter có chức năng đồng bộ.

Dữ liệu thu thập từ mạng xã hội

Các bảng `ContactsContract.StreamItems` và `ContactsContract.StreamItemPhotos` quản lý dữ liệu đến từ các mạng xã hội (social network). Bạn có thể viết một adapter có chức năng đồng bộ thể thêm Dữ liệu thu thập từ mạng xã hội vào hai bảng trên, hoặc có thể đọc dữ liệu luồng từ hai bảng này và hiển thị dữ liệu luồng trong chính ứng dụng của mình, hay cả hai. Với những tính năng này, các ứng dụng và service kết nối mạng xã hội của bạn có thể được tính hợp vào trải nghiệm mạng xã hội của Android.

Dữ liệu văn bản thu thập từ mạng xã hội

Các hạng mục của luồng luôn được liên kết với dữ liệu liên lạc thô. Cột `RAW_CONTACT_ID` liên kết với giá trị `_ID` của dữ liệu thô. Loại tài khoản và tên tài khoản của dữ liệu thô cũng được lưu trong hàng các hạng mục luồng.

Hãy lưu dữ liệu từ luồng của bạn vào các cột sau:

`ACCOUNT_TYPE`

Bắt buộc. Loại tài khoản người dùng của dữ liệu liên lạc thô liên kết với hạng mục luồng này. Hãy nhớ thiết lập giá trị này khi bạn chèn một hạng mục luồng.

`ACCOUNT_NAME`

Bắt buộc. Tên tài khoản người dùng của liên lạc thô liên kết với hạng mục luồng này. Bạn nhớ thiết lập giá trị này khi chèn một hạng mục luồng.

Các cột định danh

Bắt buộc. Bạn phải chèn các cột định danh (identifier column) khi chèn một hạng mục luồng mới:

- `CONTACT_ID`: Giá trị `_ID` của liên lạc mà hạng mục luồng này có liên kết.
- `CONTACT_LOOKUP_KEY`: Giá trị `LOOKUP_KEY` của liên lạc mà hạng mục luồng này có liên kết.
- `RAW_CONTACT_ID`: Giá trị `_ID` của dữ liệu liên lạc thô mà hạng mục luồng này có liên kết.

`COMMENTS`

Tùy chọn (không bắt buộc). Lưu thông tin tổng kết mà bạn có thể hiển thị ở đầu một hạng mục luồng.

TEXT

Văn bản của hạng mục luồng, hoặc là nội dung được nguồn hạng mục đăng, hoặc mô tả một vài hoạt động đã tạo ra hạng mục luồng này. Đây là cột có thể chứa bất cứ định dạng và các hình ảnh nguồn được nhúng vào nào, sao cho phương thức fromHtml() có thể hiển thị được. Provider có thể cắt bớt (truncate) hoặc ellipsize (cắt bớt nội dung và để dấu chấm lửng) nội dung dài, nhưng sẽ cố gắng tránh làm vỡ các thẻ (tag).

TIMESTAMP

Chuỗi văn bản chứa thời gian mà hạng mục luồng được chèn hoặc cập nhật, thời gian được biểu diễn dưới dạng *mili giây* tính từ epoch. Ứng dụng chèn hoặc cập nhật các hạng mục luồng có trách nhiệm tạo và cập nhật cột này; đây cũng là cột không được Provider Contactsduy trì tự động.

Để hiển thị thông tin nhận dạng cho các hạng mục luồng của bạn, sử dụng RES_ICON, RES_LABEL và RES_PACKAGE để liên kết các nguồn dữ liệu trong ứng dụng của bạn.

Bảng ContactsContract.StreamItems cũng chứa các cột từ SYNC1 đến SYNC4 để dành riêng cho những adapter có chức năng đồng bộ.

Dữ liệu ảnh trên luồng thu thập từ mạng xã hội

Bảng ContactsContract.StreamItemPhotos lưu các ảnh liên kết với một hạng mục luồng. Cột STREAM_ITEM_ID của bảng này liên kết với các giá trị trong cột _ID của bảng ContactsContract.StreamItems. Các tham chiếu ảnh trong bảng này nằm trên những cột sau:

Cột PHOTO (dữ liệu kiểu BLOB)

Dạng biểu diễn nhị phân của ảnh, được provider thay đổi kích thước để lưu trữ và hiển thị. Cột này luôn sẵn sàng để tương thích ngược trở lại với các phiên bản trước của Provider Contactsdùng cột này để lưu trữ ảnh. Tuy nhiên, trong phiên bản hiện tại, bạn không nên dùng cột này để lưu trữ ảnh. Thay vào đó, hãy sử dụng cột PHOTO_FILE_ID hoặc PHOTO_URI (cả hai cột này sẽ được mô tả trong các mục sau đây) để lưu trữ ảnh dưới dạng file. Hiện tại, cột này lưu ảnh đại diện (thumbnail) của bức ảnh là dữ liệu có thể xem nhanh được.

PHOTO_FILE_ID

Là một con số, định danh ảnh của một dữ liệu liên lạc thô. Gắn giá trị này vào hằng DisplayPhoto.CONTENT_URI để được content URI trỏ tới một file ảnh đơn, sau đó gọi phương thức openAssetFileDescriptor() để lấy xử lý trên file ảnh.

PHOTO_URI

Một content URI trực tiếp trỏ tới file ảnh của ảnh mà hàng này đại diện. Gọi phương thức openAssetFileDescriptor() với URI này để lấy xử lý trên file ảnh.

Sử dụng các bảng luồng thu thập từ mạng xã hội

Những bảng này hoạt động giống như các bảng chính khác trong Provider Contacts, ngoại trừ:

- Các bảng này yêu cầu phải có thêm quyền truy cập. Để đọc, ứng dụng của bạn phải có quyền `READ_SOCIAL_STREAM`. Để chỉnh sửa, ứng dụng của bạn phải có quyền `WRITE_SOCIAL_STREAM`.
- Đối với bảng `ContactsContract.StreamItems`, số lượng hàng được lưu cho mỗi dữ liệu liên lạc thô bị giới hạn. Khi đạt tới giới hạn này, Provider Contactssẽ tạo không gian cho hạng mục luồng mới bằng cách tự động xóa những hàng có `TIMESTAMP` cũ nhất. Để biết giới hạn này, thực hiện một truy vấn tới content URI `CONTENT_LIMIT_URI`. Bạn có thể bỏ qua tất cả các đối số khác, ngoại trừ content URI, bằng cách thiết lập những giá trị đó là `null`. Truy vấn này trả về một Cursor chứa một hàng đơn, có cột đơn `MAX_ITEMS`.

Lớp `ContactsContract.StreamItems.StreamItemPhotos` định nghĩa một bảng con của `ContactsContract.StreamItemPhotos` chứa các hàng ảnh cho một hạng mục luồng đơn.

Tương tác với luồng từ mạng xã hội

Dữ liệu luồng thu thập từ mạng xã hội mà Provider Contactsquản lý, kết hợp với ứng dụng liên lạc của thiết bị, trở thành một công cụ hiệu quả giúp kết nối hệ thống mạng xã hội với các liên lạc hiện có. Dưới đây là những tính năng được cung cấp:

- Bằng cách đồng bộ service mạng xã hội của bạn với Provider Contactsnhờ một adapter có chức năng đồng bộ, bạn có thể truy xuất activity gần đây cho các liên lạc của người dùng, sau đó lưu chúng vào trong các bảng `ContactsContract.StreamItems` và `ContactsContract.StreamItemPhotos` để sử dụng sau.
- Bên cạnh việc đồng bộ định kỳ, bạn có thể đặt một trigger (chứa tập các lệnh truy vấn kích hoạt dưới một sự kiện nào đó) cho adapter có chức năng đồng bộ để truy xuất dữ liệu bổ sung khi người dùng chọn hiển thị một liên lạc. Điều này cho phép adapter có chức năng đồng bộ của bạn truy xuất ảnh có độ phân giải cao và các hạng mục luồng mới nhất cho liên lạc.
- Bằng cách đăng ký một thông báo với ứng dụng liên lạc của thiết bị và Provider Contacts, bạn có thể *nhận* một intent khi hiển thị liên lạc và tại thời điểm service của bạn cập nhật trạng thái của liên lạc. Cách tiếp cận này có thể nhanh hơn, tốn ít băng thông hơn so với sử dụng đồng bộ toàn phần với adapter có chức năng đồng bộ.

  Người dùng có thể thêm một liên lạc vào service mạng xã hội của bạn trong khi đang xem liên lạc đó trong ứng dụng liên lạc của thiết bị. Bạn có thể thiết lập điều này bằng tính năng “invite contact” (“mời liên lạc”). Tính năng trên được kích hoạt bằng một tổ hợp bao gồm một activity thêm một liên lạc hiện có vào mạng đang sử dụng, kèm theo đó là một file XML cung cấp ứng dụng liên lạc của thiết bị và Provider Contactsvới những thông tin chi tiết về ứng dụng của bạn.

Việc đồng bộ định kỳ các hạng mục luồng với Provider Contactscũng giống như những đồng bộ khác. Để tìm hiểu thêm về đồng bộ, xem mục "Adapter có chức năng đồng bộ của Provider Contacts". Cách đăng ký thông báo và tính năng invite contact được giới thiệu trong hai mục tiếp theo.

Đăng ký để xử lý hiển thị trên mạng xã hội

Để đăng ký adapter có chức năng đồng bộ nhận thông báo mỗi khi người dùng hiển thị một liên lạc do adapter này quản lý, bạn cần:

1. Tạo một file có tên là `contacts.xml` trong thư mục `res/xml/` của dự án. Nếu đã có file trên, hãy bỏ qua bước này.
2. Trong file này, thêm phần tử `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`. Nếu đã có phần tử trên, hãy bỏ qua bước này.
3. Để đăng ký service nhận thông báo khi người dùng mở trang thông tin chi tiết của liên lạc trên ứng dụng liên lạc của thiết bị, thêm thuộc tính `viewContactNotifyService="serviceclass"` vào phần tử trên, với `serviceclass` là tên lớp đầy đủ của service nhận intent từ ứng dụng liên lạc của thiết bị. Với service thông báo này, hãy sử dụng lớp mở rộng `IntentService`, nhằm cho phép service nhận intent. Dữ liệu trong intent tới chứa content URI của dữ liệu liên lạc thô mà người dùng nhấn vào. Nhờ service thông báo, bạn có thể liên kết với adapter có chức năng đồng bộ, sau đó gọi adapter này để cập nhật dữ liệu cho dữ liệu liên lạc thô.

Để đăng ký activity được gọi khi người dùng nhấn vào hạng mục luồng hay ảnh, hoặc cả hai:

1. Tạo một file có tên `contacts.xml` trong thư mục `res/xml/` của dự án. Nếu đã có file trên, hãy bỏ qua bước này.
2. Trong file này, thêm vào phần tử `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`. Nếu đã có phần tử trên, hãy bỏ qua bước này.
3. Để đăng ký một trong các acitvity của ứng dụng nhận xử lý sự kiện người dùng nhấn vào hạng mục luồng trong ứng dụng liên lạc của thiết bị, thêm thuộc tính `viewStreamItemActivity="activityclass"` vào phần tử trên, trong đó `activityclass` là tên lớp đầy đủ của activity sẽ nhận intent từ ứng dụng liên lạc của thiết bị.
4. Để đăng ký một trong các activity nhận xử lý sự kiện người dùng nhấn vào một ảnh thuộc hạng mục luồng trong ứng dụng liên lạc của thiết bị, thêm thuộc tính `viewStreamItemPhotoActivity="activityclass"` vào phần tử trên, trong đó `activityclass` là tên lớp đầy đủ của activity nhận intent từ ứng dụng liên lạc của thiết bị.

Mục "Phần tử `<ContactsAccountType>`" mô tả chi tiết hơn về phần tử `<ContactsAccountType>`.

Intent tới chứa content URI của hạng mục hoặc ảnh mà người dùng đã nhấn vào. Để có các activity riêng biệt cho hạng mục văn bản và ảnh, hãy sử dụng các thuộc tính trong cùng một file.

Tương tác với service mạng xã hội của bạn

Người dùng không phải rời ứng dụng liên lạc của thiết bị để mời một liên lạc vào trang mạng xã hội của bạn. Thay vào đó, bạn có thể để ứng dụng liên lạc của thiết bị gửi một intent mời liên lạc đó tới một trong các activity của mình. Để thiết lập điều này, bạn cần:

1. Tạo một file có tên `contacts.xml` trong thư mục `res/xml/` của dự án bạn phát triển. Nếu đã có file trên, hãy bỏ qua bước này.
2. Trong file này, thêm phần tử `<ContactsAccountType xmlns:android="http://schemas.android.com/apk/res/android">`. Nếu đã có phần tử trên, hãy bỏ qua bước này.
3. Thêm các thuộc tính sau:
   - `inviteContactActivity="activityclass"`
   - `inviteContactActionLabel="@string/invite_action_label"`

Giá trị `activityclass` là tên lớp đầy đủ của activity nhận intent. Giá trị `invite_action_label` là chuỗi văn bản được hiển thị trong menu **Add Connection** (thêm kết nối) ở ứng dụng liên lạc của thiết bị.

Ghi chú: `ContactsSource` là tên thẻ cũ của `ContactsAccountType`.

Tham khảo hướng dẫn sử dụng file contacts.xml

File `contacts.xml` chứa các phần tử XML điều khiển việc tương tác giữa adapter có chức năng đồng bộ và ứng dụng với ứng dụng liên lạc và Provider Contacts. Các phần tử của file này sẽ được giới thiệu ở những mục dưới đây.

Phần tử <ContactsAccountType>

Phần tử `<ContactsAccountType>` điều khiển việc tương tác của ứng dụng bạn phát triển với ứng dụng liên lạc. Cú pháp của phần tử này như sau:

```
<ContactsAccountType
       xmlns:android="http://schemas.android.com/apk/res/android"
       inviteContactActivity="activity_name"
       inviteContactActionLabel="invite_command_text"
       viewContactNotifyService="view_notify_service"
       viewGroupActivity="group_view_activity"
       viewGroupActionLabel="group_action_text"
       viewStreamItemActivity="viewstream_activity_name"
       viewStreamItemPhotoActivity="viewphotostream_activity_name">
```

được lưu trong:

`res/xml/contacts.xml`

Có thể bao gồm phần tử:

`<ContactsDataKind>`

Mô tả:

Khai báo các thành phần của Android và các nhãn trên giao diện người dùng cho phép người dùng mời một trong các liên lạc của họ vào mạng xã hội, thông báo với người dùng khi một trong những luồng mạng xã hội của họ được cập nhật,…

Lưu ý, không cần sử dụng tiền tố thuộc tính `android:` cho thuộc tính `<ContactsAccountType>`.

Thuộc tính:

`inviteContactActivity`

Tên lớp đầy đủ của activity trong ứng dụng của bạn, là activity bạn muốn kích hoạt khi người dùng chọn mục **Add connection** từ ứng dụng liên lạc của thiết bị.

`inviteContactActionLabel`

Chuỗi văn bản hiển thị cho activity, được xác định trong thuộc tính `inviteContactActivity`, trên menu **Add connection**. Ví dụ, bạn có thể sử dụng chuỗi "Follow in my network" ("Theo dõi trong mạng của tôi"). Bạn có thể sử dụng một định danh tài nguyên dạng chuỗi cho nhãn này.

`viewContactNotifyService`

Tên lớp đầy đủ của một service trong ứng dụng của bạn, là service sẽ nhận thông báo khi người dùng hiển thị liên lạc. Thông báo này được ứng dụng liên lạc của thiết bị gửi đi; việc này cho phép ứng dụng của bạn có thể hoãn các thao tác cần nhiều dữ liệu cho đến khi cần thiết. Ví dụ, ứng dụng của bạn có thể phản hồi thông báo này bằng việc đọc cũng như hiển thị ảnh có độ phân giải cao và hạng mục luồng mạng xã hội gần đây nhất của liên lạc. Mục "Tương tác với luồng mạng xã hội" mô tả chi tiết hơn về tính năng này. Bạn có thể xem một ví dụ về service thông báo trong file `NotifierService.java` của ứng dụng mẫu SampleSyncAdapter.

`viewGroupActivity`

Tên lớp đầy đủ của activity có thể hiển thị thông tin về nhóm trong ứng dụng của bạn. Khi người dùng nhấn vào nhãn của nhóm trong ứng dụng liên lạc của thiết bị, giao diện người dùng cho activity này hiển thị.

`viewGroupActionLabel`

Nhãn mà ứng dụng liên lạc hiển thị cho điều khiển giao diện người dùng, là điều khiển cho phép người dùng xem các nhóm trong ứng dụng của bạn.

Ví dụ, nếu cài đặt ứng dụng Google+ trên thiết bị và đồng bộ Google+ với ứng dụng liên lạc, bạn sẽ thấy những vòng kết nối (circle) Google+ được liệt kê thành các nhóm trong tab **Groups** của ứng dụng liên lạc. Nếu nhấn vào nhóm Google+, bạn sẽ thấy mọi người trong vòng kết nối này được liệt kê như một "nhóm". Ở đầu

vùng hiển thị các liên lạc Google+, sẽ xuất hiện một biểu tượng của Google+; nếu bạn nhấn vào biểu tượng này, điều khiển sẽ được chuyển hướng sang ứng dụng Google+. Ứng dụng liên lạc làm được điều này thông qua `viewGroupActivity`, sử dụng biểu tượng Google+ làm giá trị cho `viewGroupActionLabel`.

Thuộc tính này cũng có thể sử dụng một định danh tài nguyên dạng chuỗi.

`viewStreamItemActivity`

Tên lớp đầy đủ của activity trong ứng dụng của bạn mà ứng dụng liên lạc khởi động khi người dùng nhấn vào một hạng mục luồng của dữ liệu liên lạc thô.

`viewStreamItemPhotoActivity`

Tên lớp đầy đủ của activity trong ứng dụng mà ứng dụng liên lạc khởi động khi người dùng nhấn vào một ảnh trong hạng mục luồng của dữ liệu liên lạc thô.

Phần tử <ContactsDataKind>

Phần tử `<ContactsDataKind>` điều khiển việc hiển thị các hàng dữ liệu tùy chỉnh cho ứng dụng bạn phát triển trong giao diện người dùng của ứng dụng liên lạc. Phần tử này có cú pháp như sau:

```
<ContactsDataKind
        android:mimeType="MIMEtype"
        android:icon="icon_resources"
        android:summaryColumn="column_name"
        android:detailColumn="column_name">
```

nằm trong:

`<ContactsAccountType>`

Mô tả:

Sử dụng phần tử này để ứng dụng liên lạc hiển thị nội dung của hàng dữ liệu tùy chỉnh làm một phần trong các thông tin chi tiết của dữ liệu liên lạc thô. Mỗi phần tử con `<ContactsDataKind>` của `<ContactsAccountType>` đại diện cho một kiểu hàng dữ liệu tùy chỉnh mà adapter có chức năng đồng bộ của bạn thêm vào bảng `ContactsContract.Data`. Thêm một phần tử `<ContactsDataKind>` vào mỗi kiểu MIME tùy chỉnh bạn sử dụng. Bạn không phải thêm phần tử này nếu đây là hàng dữ liệu tùy chỉnh mà bạn không muốn hiển thị dữ liệu.

Thuộc tính:

`android:mimeType`

Kiểu MIME tùy chỉnh mà bạn định nghĩa cho một trong những kiểu hàng dữ liệu tùy chỉnh của mình tại bảng `ContactsContract.Data`. Ví dụ, giá trị `vnd.android.cursor.item/vnd.example.locationstatus` có thể là kiểu MIME tùy chỉnh cho hàng dữ liệu ghi lại vị trí mới nhất của liên lạc.

`android:icon`

Một tài nguyên drawable (drawable resource) của Android mà ứng dụng liên lạc hiển thị bên cạnh dữ liệu của bạn. Sử dụng thuộc tính này để chỉ cho người dùng biết rằng dữ liệu đến từ service của bạn.

`android:summaryColumn`

Tên cột của giá trị đầu tiên trong hai giá trị được truy xuất từ hàng dữ liệu. Giá trị này khi được hiển thị là dòng đầu tiên của mục cho hàng dữ liệu này. Hàng đầu tiên có mục đích sử dụng là thông tin tổng kết của dữ liệu, nhưng là dữ liệu tùy chọn (có thể có hoặc không). Xem thêm thuộc tính `android:detailColumn.`

`Android:detailColumn`

Tên cột của giá trị thứ hai trong hai giá trị được truy xuất từ hàng dữ liệu. Giá trị này được hiển thị là dòng thứ hai của mục cho hàng dữ liệu này. Xem thêm thuộc tính `android:summaryColumn`.

Các tính năng bổ sung của Provider Contacts

Bên cạnh những tính năng chính mô tả trong các mục trước, Provider Contactshỗ trợ những tính năng hữu dụng dưới đây để làm việc với dữ liệu liên lạc:

- Nhóm liên lạc.
- Các tính năng liên quan đến ảnh.

Nhóm liên lạc

Provider Contactscó thể gán nhãn tùy chọn cho tập các liên lạc liên quan thành dữ liệu **nhóm (group)**. Nếu server liên kết với tài khoản người dùng muốn duy trì các nhóm, adapter có chức năng đồng bộ của loại tài khoản này phải truyền dữ liệu nhóm giữa Provider Contactsvà server. Khi người dùng thêm một liên lạc mới vào server, sau đó đưa liên lạc này vào một nhóm mới, adapter có chức năng đồng bộ phải thêm nhóm mới vào bảng `ContactsContract.Groups`. Nhóm hoặc các nhóm chứa một dữ liệu liên lạc thô được lưu trong bảng `ContactsContract.Data`, sử dụng kiểu MIME `ContactsContract.CommonDataKinds.GroupMembership`.

Nếu bạn đang thiết kế một adapter có chức năng đồng bộ để thêm dữ liệu liên lạc thô từ server vào Provider Contacts, đồng thời đang không sử dụng nhóm, bạn cần yêu cầu provider hiển thị dữ liệu này. Trong mã thực thi khi người dùng thêm tài khoản vào thiết bị, cập nhật hàng `ContactsContract.Setting` mà Provider Contacts thêm vào tài khoản này. Trong hàng này, thiết lập giá trị của cột `Settings.UNGROUPED_VISIBLE` là `1`. Khi thực hiện điều này, Provider Contactsluôn hiển thị dữ liệu liên lạc đó, ngay cả khi bạn không sử dụng nhóm.

Ảnh của liên lạc

Bảng `ContactsContract.Data` lưu trữ ảnh dưới dạng các hàng có kiểu MIME là `Photo.CONTENT_ITEM_TYPE`. Cột `CONTACT_ID` của hàng này được liên kết với cột `_ID` của dữ liệu liên lạc thô chứa ảnh này. Lớp `ContactsContract.Contacts.Photo`

định nghĩa một bảng con của bảng `ContactsContract.Contacts` chứa thông tin ảnh cho ảnh chính của một liên lạc, đây cũng là ảnh chính cho dữ liệu liên lạc thô chính của liên lạc. Tương tự, lớp `ContactsContract.RawContacts.DisplayPhoto` định nghĩa một bảng con của bảng `ContactsContract.RawContacts` chứa thông tin ảnh cho ảnh chính của dữ liệu liên lạc thô.

Tài liệu tham khảo của `ContactsContract.Contacts.Photo` và `ContactsContract.RawContacts.DisplayPhoto` chứa các ví dụ minh họa về truy xuất thông tin ảnh. Không có lớp nào tiện cho việc truy xuất ảnh đại diện chính cho một dữ liệu liên lạc thô, nhưng bạn có thể gửi một truy vấn tới bảng `ContactsContract.Data`, chọn `_ID` của dữ liệu liên lạc thô, `Photo.CONTENT_ITEM_TYPE` và cột `IS_PRIMARY` để tìm hàng ảnh chính của dữ liệu liên lạc thô.

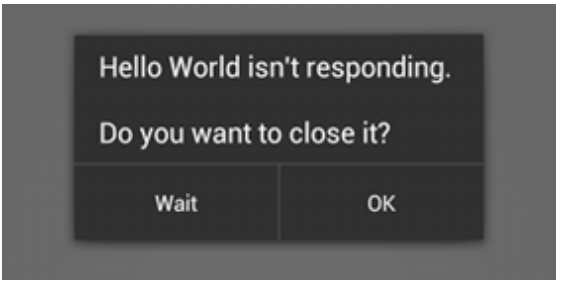
Dữ liệu luồng từ mạng xã hội của một người cũng có thể bao gồm ảnh. Các ảnh này được lưu trữ tại bảng `ContactsContract.StreamItemPhotos`, được mô tả chi tiết hơn trong mục "Dữ liệu ảnh trên luồng từ mạng xã hội".

11. Thiết kế sao cho ứng dụng có thể phản hồi tốt

Chúng ta có thể viết mã để vượt qua mọi phép kiểm thử về hiệu năng trên thế giới, song vẫn cảm thấy chậm, bị treo hoặc đóng băng (ngừng xử lý) tại một số thời điểm, hoặc mất nhiều thời gian cho việc xử lý đầu vào. Điều tồi tệ nhất có thể xảy ra với việc phản hồi của ứng dụng bạn phát triển là xuất hiện hộp thoại "Application Not Responding" (ANR) (Ứng dụng không phản hồi).

Trong hệ điều hành Android, hệ thống sẽ theo dõi để tránh xảy ra tình trạng các ứng dụng phản hồi quá chậm trong một khoảng thời gian bằng cách hiển thị hộp thoại thông báo rằng ứng dụng của bạn đã dừng phản hồi, chẳng hạn như hộp thoại ở Hình 1. Tại thời điểm này, ứng dụng của bạn đã không phản hồi trong một khoảng thời gian đáng kể nhất định; do vậy, hệ thống cung cấp một tùy chọn cho người dùng để thoát khỏi ứng dụng. Đó là giới hạn để thiết kế phản hồi bên trong ứng dụng của bạn sao cho hệ thống không bao giờ phải hiển thị hộp thoại ANR tới người dùng.

Phần này mô tả cách hệ thống Android xác định xem một ứng dụng có không phản hồi hay không và cung cấp hướng dẫn để đảm bảo ứng dụng của bạn luôn phản hồi kịp thời yêu cầu từ người dùng.



Chú thích

Hello World isn't responding:
Ứng dụng Hello World không phản hồi

Do you want to close it? :
Bạn có muốn dừng ứng dụng?

Wait:
Tiếp tục đợi phản hồi

Hình 1. Một hộp thoại ANR được hiển thị cho người dùng.

Trigger ANR là gì?

Nhìn chung, hệ thống sẽ hiển thị một ANR nếu ứng dụng không thể phản hồi đầu vào của người dùng. Ví dụ, nếu một ứng dụng chặn (block) một số thao tác I/O (Input/Output - Nhập/Xuất) (thường là truy cập mạng) trên luồng giao diện người dùng (UI thread), hệ thống sẽ không thể xử lý sự kiện nhập liệu của người dùng khi nó xảy ra. Hoặc, ứng dụng có thể mất quá nhiều thời gian để xây dựng một cấu trúc phức tạp trong bộ nhớ, hoặc tính toán bước di chuyển tiếp theo trong một trò chơi trên luồng giao diện người dùng. Bởi vậy, điều quan trọng là cần đảm bảo rằng những tính toán này thực sự đem lại hiệu quả; thế nhưng, mã hiệu quả nhất vẫn gây tốn thời gian cho việc chạy.

Trong một tình huống bất kỳ mà tại đó, ứng dụng của bạn thực hiện một thao tác tốn thời gian đáng kể, **bạn không nên thực hiện hoạt động này trên luồng giao diện người dùng**. Thay vào đó, hãy tạo một luồng công việc (worker thread) và thực hiện hầu hết các việc tại đây. Điều này giữ cho luồng giao diện người dùng - luồng điều khiển vòng lặp sự kiện giao diện người dùng - chạy và tránh cho hệ thống khỏi kết luận rằng mã của bạn bị đóng băng (ngừng hoạt động). Do việc chạy luồng như vậy thường được tiến hành tại mức lớp, nên bạn có thể nghĩ tới việc phản hồi như một vấn đề về *lớp*. (So sánh với hiệu suất mã cơ bản, việc này phải xử lý ở mức *phương thức*).

Trong Android, phản hồi của ứng dụng được các service của hệ thống, bao gồm Activity Manager (trình quản lý activity) và Window Manager (trình quản lý Window) điều khiển. Android luôn hiển thị hộp thoại ANR cho một ứng dụng cụ thể khi hệ điều hành này phát hiện một trong những trường hợp sau:

- Không có phản hồi cho một sự kiện đầu vào (chẳng hạn như sự kiện nhấn phím hoặc sự kiện chạm vào màn hình) trong vòng 5 giây.
- Đối tượng `BroadcastReceiver` không hoàn thành xử lý trong vòng 10 giây.

Cách tránh hộp thoại ANR

Thông thường, ứng dụng Android chạy hoàn toàn trên một luồng đơn mặc định là "luồng giao diện người dùng" hoặc "luồng chính". Điều này có nghĩa là, mọi hoạt động ứng dụng thực hiện trong luồng giao diện người dùng, nếu mất nhiều thời gian để hoàn thành, có thể làm xuất hiện hộp thoại ANR. Đó là vì ứng dụng của bạn không có cơ hội xử lý sự kiện đầu vào hoặc intent broadcast.

Do đó, bất cứ phương thức nào chạy trong luồng giao diện người dùng cũng nên thực hiện ít việc nhất có thể trong luồng này. Cụ thể, các activity nên thực hiện ít việc nhất có thể trong những phương thức liên quan đến vòng đời như `onCreate()` và `onResume()`. Các thao tác tốn nhiều thời gian, chẳng hạn như thao tác mạng hoặc cơ sở dữ liệu, hay những phép tính cần phải tính toán nhiều như tính lại kích thước ảnh bitmap, nên được thực hiện trong luồng công việc (hoặc trong trường hợp các thao tác cơ sở dữ liệu sẽ dùng một yêu cầu không đồng bộ).

Cách hiệu quả nhất để tạo luồng công việc cho các thao tác dài là tạo bằng lớp AsyncTask. Đơn giản là mở rộng (extend) lớp AsyncTask và cài đặt phương thức doInBackground() để thực hiện việc này. Để chuyển các thay đổi của tiến trình tới người dùng, bạn có thể gọi phương thức publishProgress(), là phương thức gọi phương thức callback onProgressUpdate(). Trong bản cài đặt phương thức onProgressUpdate() của mình (phương thức chạy trên luồng giao diện người dùng), bạn có thể thông báo cho người dùng thông tin họ cần biết. Ví dụ:

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    // Thực hiện các thao tác tốn nhiều thời gian tại đây
    protected Long doInBackground(URL... urls){
        int count = urls.length;
        long totalSize =0;
        for(int i =0; i < count; i++){
            totalSize +=Downloader.downloadFile(urls[i]);
            publishProgress((int)((i /(float) count)*100));
            // Thoát ra sớm nếu phương thức cancel() được gọi
            if (isCancelled())  break;
        }
        return totalSize;
    }

    // Phương thức sau được gọi mỗi khi bạn gọi phương thức
    // publishProgress()
    protected void onProgressUpdate(Integer... progress){
        setProgressPercent(progress[0]);
    }

    // Phương thức sau được gọi khi phương thức doInBackground() hoàn
    // thành
    protected void onPostExecute(Long result){
        showNotification("Downloaded "+ result +" bytes");
    }
}
```

Để chạy luồng công việc này, bạn chỉ cần tạo một thể hiện cho lớp trên và gọi phương thức execute():

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Mặc dù điều này phức tạp hơn việc cài đặt lớp AsyncTask, nhưng có thể bạn vẫn muốn cài đặt lớp Thread hay một lớp HandlerThread của chính mình. Khi thực hiện, bạn nên thiết lập mức ưu tiên của luồng là mức ưu tiên "background" ("chạy ngầm") bằng cách gọi phương thức Process.setThreadPriority() và truyền THREAD_PRIORITY_BACKGROUND. Nếu không thiết lập luồng do bạn xây dựng có

mức ưu tiên thấp theo cách này, có thể luồng đó vẫn làm chậm ứng dụng của bạn, do mặc định luồng được tạo ra hoạt động ở mức ưu tiên giống với luồng giao diện người dùng.

Nếu cài đặt lớp `Thread` hoặc `HandlerThread`, cần đảm bảo luồng giao diện người dùng của bạn không bị chặn trong khi chờ luồng công việc hoàn thành - không gọi phương thức `Thread.wait()` hay `Thread.sleep()`. Thay vì chặn chương trình trong khi chờ luồng công việc hoàn thành, luồng chính của bạn nên cung cấp một đối tượng `Handler` cho các luồng khác để gửi trở lại điều khiển sau khi hoàn thành. Thiết kế ứng dụng của bạn theo cách này sẽ cho phép luồng giao diện người dùng của ứng dụng duy trì việc phản hồi cho hoạt động đầu vào, từ đó tránh được hộp thoại ANR khi không kịp phản hồi hoạt động đầu vào trong vòng 5 giây.

Ràng buộc cụ thể về thời gian thực thi `BroadcastReceiver` nhấn mạnh vào những gì broadcast receiver phải làm: Thực hiện số lượng thao tác nhỏ, rời rạc trong luồng chạy ngầm, chẳng hạn như lưu một thiết lập hoặc đăng ký một thông báo `Notification`. Do đó, với các phương thức khác được gọi trong luồng giao diện người dùng, ứng dụng phải tránh những thao tác hoặc tính toán tốn nhiều thời gian trong broadcast receiver. Tuy nhiên, thay vì thực hiện các tác vụ chuyên sâu thông qua luồng công việc, bạn nên phát triển ứng dụng theo hướng khởi động một `IntentService` nếu cần thực hiện một action mất nhiều thời gian để phản hồi một intent broadcast.

> **Mách nhỏ**: Bạn có thể sử dụng `StrictMode` để tìm các thao tác tốn nhiều thời gian, chẳng hạn như thao tác với mạng hoặc trên cơ sở dữ liệu mà bạn vô tình thực hiện trong luồng chính của mình.

Tăng cường phản hồi

Thông thường, khoảng thời gian từ 100 đến 200 mili giây là ngưỡng tối đa để người dùng có thể nhận ra rằng ứng dụng của bạn đang bị chậm. Do đó, sau đây là một số mẹo khác mà bạn nên thực hiện để tránh hộp thoại ANR, đồng thời khiến ứng dụng của bạn dường như luôn phản hồi người dùng:

- Nếu ứng dụng đang thực hiện các thao tác chạy ngầm để phản hồi thao tác đầu vào của người dùng, hãy minh họa việc tiến trình đang được xử lý (chẳng hạn như sử dụng `ProgressBar` trong giao diện người dùng của bạn).
- Riêng với game, hãy tính toán các bước di chuyển trong tiến trình worker (hay còn gọi là luồng công việc).
- Nếu ứng dụng của bạn có pha thiết lập khởi tạo tốn nhiều thời gian, hãy nghĩ tới việc hiển thị một màn hình splash (màn hình xuất hiện từ từ) hoặc xuất ra màn hình chính sao cho nhanh nhất có thể, cho biết ứng dụng đang xử lý việc nạp và nhập các thông tin đồng bộ. Trong cả hai trường hợp, bạn nên chỉ ra rằng vì một vài lý do nào đó chưa rõ mà tiến trình đang được xử lý, để người dùng không cho rằng ứng dụng bị đóng băng (ngừng xử lý).
- Sử dụng các công cụ kiểm tra hiệu suất như Systrace và Traceview để xác định điểm tắc nghẽn (bottleneck) của việc phản hồi trong ứng dụng do bạn phát triển.

12. Service

Service (dịch vụ) là một thành phần của ứng dụng, có thể thực hiện các thao tác tốn thời gian chạy ngầm và không cung cấp giao diện người dùng. Các thành phần khác của ứng dụng có thể khởi động một service và service sẽ tiếp tục chạy ngầm ngay cả khi người dùng chuyển sang ứng dụng khác. Ngoài ra, mỗi thành phần có thể liên kết với một service để tương tác với service đó, thậm chí thực hiện các giao tiếp liên tiến trình (interprocess communication - IPC). Ví dụ, một service có thể xử lý các giao dịch mạng, phát nhạc, thực hiện nhập/xuất file hoặc tương tác với một content provider, tất cả đều chạy ngầm.

Về cơ bản, một service có thể nhận hai dạng sau:

Khởi động

> Một service được gọi là "service khởi động" ("started") khi một thành phần của ứng dụng (chẳng hạn như một activity) khởi động service đó bằng cách gọi phương thức `startService()`. Khi được khởi động, một service có thể chạy ngầm vô hạn định, thậm chí ngay cả khi thành phần khởi động service đã bị hủy. Một service được khởi tạo thường thực hiện một thao tác đơn và không trả lại kết quả cho thành phần gọi. Ví dụ, service này có thể tải về hoặc đẩy một file lên trên mạng. Khi thao tác này đã hoàn thành, service này nên tự dừng lại.

Có liên kết

> Một service được gọi là "có liên kết" ("bound") khi một thành phần của ứng dụng gắn với service này bằng cách gọi phương thức `bindService()`. Service có liên kết cung cấp giao diện client-server cho phép các thành phần tương tác với service này, gửi yêu cầu, nhận kết quả, thậm chí là thực hiện các tiến trình chéo (across process) với các giao tiếp liên tiến trình. Một service có liên kết chỉ chạy với lượng thời gian tương tự thành phần ứng dụng khác được gắn với service này. Nhiều thành phần có thể gắn cùng lúc với một service kiểu này, nhưng khi tất cả trong số chúng không còn liên kết, service bị hủy.

Nhìn chung, tài liệu này sẽ thảo luận riêng về hai kiểu service trên, nhưng service của bạn có thể thực hiện theo cả hai cách - có thể được khởi động (chạy vô hạn định) và cũng cho phép việc liên kết. Bạn có thể dễ dàng thực hiện điều này bằng cách cài đặt cả hai phương thức callback: Phương thức `onStartCommand()` cho phép các thành phần khởi động service và phương thức `onBind()` để tạo liên kết.

Bất kể ứng dụng của bạn đã khởi động, có liên kết hay cả hai, bất cứ thành phần ứng dụng nào cũng có thể dùng service này (thậm chí từ một ứng dụng khác), giống như cách thành phần bất kỳ có thể sử dụng một activity - bằng cách khởi động service với cùng một Intent. Tuy nhiên, bạn có thể khai báo service này là private (riêng tư), trong file kê khai, đồng thời khóa việc truy cập từ các ứng dụng khác. Mục "Khai báo service trong file kê khai" bên dưới sẽ được thảo luận chi tiết hơn về nội dung này.

Chú ý: Service sẽ chạy trong luồng chính của tiến trình chứa nó - service **không** tạo luồng riêng và **không** chạy trong tiến trình riêng (trừ phi bạn quy định kiểu khác). Điều này có nghĩa là, nếu service sẽ thực hiện công việc chiếm nhiều bộ xử lý trung tâm (Central Processing Unit - CPU) hoặc chặn các thao tác (chẳng hạn như phát nhạc MP3 hoặc kết nối mạng), bạn nên tạo một luồng mới bên trong service để thực hiện

công việc này. Bằng cách sử dụng một luồng riêng, bạn sẽ làm giảm rủi ro xuất hiện lỗi ANR (Application Not Responding - Ứng dụng không phản hồi) và luồng chính của ứng dụng có thể vẫn dành để người dùng tương tác với activity của bạn.

Cơ bản về service

Bạn nên sử dụng service hay luồng?

Service chỉ đơn giản là một thành phần có thể chạy ngầm ngay cả khi người dùng không tương tác với ứng dụng. Do vậy, bạn chỉ nên tạo một service nếu đó là những gì bạn cần.

Nếu cần thực hiện công việc bên ngoài luồng chính (main thread), nhưng chỉ trong khi người dùng đang tương tác với ứng dụng, bạn nên tạo một luồng mới, không phải là service. Ví dụ, nếu muốn phát nhạc, nhưng chỉ trong lúc activity của mình đang chạy, bạn có thể tạo một luồng trong phương thức `onCreate()`, chạy luồng này bằng phương thức `onStart()`, sau đó dừng bằng phương thức `onStop()`. Ngoài ra, bạn nên cân nhắc sử dụng `AsyncTask` hay `HandlerThread`, thay vì sử dụng lớp `Thread` truyền thống. Xem tài liệu hướng dẫn "Processes and Threads" ("Tiến trình và luồng") để tìm hiểu thêm thông tin về luồng.

Cần nhớ rằng, nếu bạn sử dụng một service, thì mặc định service đó sẽ chạy trong luồng chính của ứng dụng. Do đó, bạn vẫn nên tạo một luồng mới bên trong service nếu service này thực hiện các thao tác chiếm nhiều bộ xử lý hoặc thao tác chặn.

Để tạo một service, bạn phải tạo một lớp con của lớp `Service` (hoặc một trong những lớp con đã có của lớp này). Trong phần cài đặt, bạn nên ghi đè (override) một vài phương thức callback để xử lý các công việc chính trong vòng đời của service, đồng thời cung cấp một cơ chế cho các thành phần liên kết với service này, nếu thích hợp. Các phương thức callback quan trọng nhất cần ghi đè là:

`onStartCommand()`

Hệ thống sẽ gọi phương thức này khi một thành phần khác, chẳng hạn như một activity, yêu cầu khởi động một service bằng cách gọi `startService()`. Khi phương thức này thực thi, service được khởi tạo và có thể chạy ngầm vô hạn định. Nếu cài đặt phương thức này, nhiệm vụ của bạn là dừng service khi hoàn thành công việc, bằng cách gọi phương thức `stopSelf()` hoặc `stopService()`. (Nếu chỉ muốn cung cấp liên kết, bạn không cần cài đặt phương thức này).

`onBind()`

Hệ thống gọi phương thức này khi thành phần khác muốn liên kết với service, chẳng hạn như thực hiện RPC (Remote Procedure Call - Thủ tục gọi hàm từ xa), bằng cách gọi phương thức `bindService()`. Khi cài đặt phương thức này, bạn phải cung cấp giao diện cho những client dùng để giao tiếp với service, bằng cách trả về một `IBinder`. Bạn luôn phải cài đặt phương thức này, nhưng nếu không muốn tạo liên kết, bạn nên trả về null.

`onCreate()`

Hệ thống gọi phương thức này khi service được tạo lần đầu, nhằm thực hiện các thủ tục thiết lập một lần (trước khi hệ thống gọi phương thức `onStartCommand()` hoặc

onBind()). Nếu service đã chạy, phương thức này không được gọi.

`onDestroy()`

Hệ thống gọi phương thức này khi service không còn được sử dụng và sẽ bị hủy bỏ. Service của bạn nên cài đặt phương thức này để dọn sạch bất cứ tài nguyên nào đang sử dụng, chẳng hạn như luồng, trình lắng nghe đã đăng ký (registered listener), trình nhận (receiver),... Đây là lời gọi cuối cùng service nhận được.

Nếu một thành phần khởi động service bằng cách gọi phương thức `startService()` (dẫn đến việc gọi phương thức `onStartCommand()`), service vẫn chạy cho đến khi tự dừng bằng phương thức `stopSelf()`, hoặc khi thành phần khác dừng service này bằng cách gọi phương thức `stopService()`.

Nếu một thành phần gọi phương thức `bindService()` để tạo một service (và phương thức `onStartCommand()` *không* được gọi), service này chỉ chạy với lượng thời gian tương tự thành phần gắn với nó. Khi một service không được gắn vào bất kỳ client nào, hệ thống sẽ hủy service đó.

Hệ thống Android chỉ ép một service dừng khi sắp hết bộ nhớ và hệ điều hành phải khôi phục tài nguyên hệ thống cho activity có người dùng đang focus. Nếu service này được gắn với một activity mà người dùng đang focus thì ít có khả năng bị ép dừng. Nếu service được khai báo để chạy trên chế độ màn hình chính (sẽ được thảo luận sau), hầu như service đó sẽ không bao giờ bị ép dừng. Mặt khác, nếu service đã được khởi tạo và sẽ chạy trong thời gian dài, hệ thống sẽ giảm vị trí của service này trong danh sách các tác vụ chạy ngầm quá hạn và service đó sẽ có khả năng cao bị ép dừng - nếu là service đã khởi động, bạn phải thiết kế sao cho hệ thống có thể khởi động lại service này. Khi đó, nếu hệ thống ép service dừng, nó sẽ khởi động lại service này ngay khi có đủ tài nguyên (mặc dù việc này còn phụ thuộc vào giá trị bạn trả về từ phương thức `onStartCommand()`, sẽ được bàn luận sau). Để tìm hiểu thêm về thời điểm hệ thống có thể hủy bỏ một service, xem mục hướng dẫn về "Tiến trình và luồng".

Trong các mục dưới đây, bạn sẽ tìm hiểu cách thức tạo mỗi loại service trên và sử dụng từng loại service này từ các thành phần khác của ứng dụng.

Khai báo một service trong file kê khai

Tương tự activity (và các thành phần khác), bạn phải khai báo tất cả các service trong file kê khai của ứng dụng.

Để khai báo một service, bạn thêm một phần tử `<service>` làm phần tử con của phần tử `<application>`. Ví dụ:

```
<manifest ... >
    ...
    <application ... >
        <serviceandroid:name=".ExampleService"/>
        ...
    </application>
</manifest>
```

Bạn có thể thêm vào phần tử `<service>` các thuộc tính khác để định nghĩa những tính chất như các quyền cần thiết để khởi động service và tiến trình chạy service này. Thuộc tính `android:name` là thuộc tính bắt buộc duy nhất, xác định tên lớp cho service. Khi phát hành ứng dụng, bạn không nên thay đổi tên này bởi nếu thay đổi, bạn có thể phá hỏng một số chức năng chứa các intent tường minh (explicit intent) được dùng để tham chiếu đến service của bạn (đọc bài đăng trên blog, "Things That Cannot Change" - tạm dịch: "Những thứ không thể thay đổi").

Xem tài liệu tham khảo về phần tử `<service>` để tìm hiểu thêm về cách khai báo service trong file kê khai (manifest file) .

Tương tự activity, service có thể định nghĩa bộ lọc intent để cho phép những thành phần khác gọi service này thông qua các intent ngầm định (implicit intent). Bằng cách khai báo các bộ lọc intent, những thành phần từ một ứng dụng bất kỳ được cài đặt trên thiết bị người dùng có thể khởi động service của bạn, nếu service miêu tả một bộ lọc intent phù hợp với intent mà ứng dụng đó truyền vào phương thức `startService()`.

Nếu đã lên kế hoạch chỉ sử dụng service do mình phát triển một cách cục bộ (các ứng dụng khác không được dùng service này), bạn không cần (và cũng không nên) cung cấp bất cứ bộ lọc intent nào. Không có bộ lọc intent, bạn phải khởi động service bằng cách sử dụng một intent xác định rõ tên cho lớp service. Chi tiết về việc khởi động một service sẽ được thảo luận sau.

Ngoài ra, bạn có thể đảm bảo service là dành riêng cho ứng dụng mình đang phát triển chỉ bằng cách thêm thuộc tính `android:exported` và thiết lập là `"false"`. Đây là tính năng hiệu quả ngay cả khi service có cung cấp bộ lọc intent.

Để tìm hiểu thêm về cách tạo bộ lọc intent cho service của bạn, xem tài liệu hướng dẫn về "Intent và bộ lọc intent".

Tạo service dạng khởi động

Service dạng khởi động là service mà phần tử khác khởi động bằng cách gọi phương thức `startService()`, dẫn đến kết quả là một lời gọi tới phương thức `onStartCommand()` của service.

Khi được khởi động, vòng đời của service sẽ không phụ thuộc vào thành phần khởi động nó và service sẽ chạy ngầm vô hạn định, ngay cả khi thành phần đã khởi động service bị hủy. Do đó, service phải tự dừng khi công việc đã hoàn thành bằng cách gọi phương thức `stopSelf()`, hoặc thành phần khác có thể dừng service này bằng cách gọi phương thức `stopService()`.

Một thành phần của ứng dụng, chẳng hạn như activity, có thể khởi động service này bằng cách gọi phương thức `startService()` và truyền vào một `Intent` xác định service, sau đó bao gồm dữ liệu bất kỳ để service sử dụng. Service nhận `Intent` này trong phương thức `onStartCommand()`.

Chẳng hạn, giả sử một activity cần lưu một số dữ liệu lên cơ sở dữ liệu trực tuyến. Activity này có thể khởi động một service và truyền cho service này dữ liệu cần lưu bằng cách truyền một intent vào phương thức `startService()`. Service này nhận intent trên trong phương thức `onStartCommand()`, kết nối với Internet và thực hiện giao dịch cơ sở dữ liệu. Khi giao dịch này hoàn thành, service tự dừng, từ đó hệ thống sẽ hủy nó.

Chú ý: Một service sẽ chạy trong cùng tiến trình với ứng dụng khai báo nó và mặc định là trong luồng chính của ứng dụng. Do vậy, nếu service của bạn thực hiện các thao tác sử dụng nhiều bộ xử lý hoặc thao tác chặn trong khi người dùng tương tác với một activity cùng ứng dụng, service này sẽ làm chậm hiệu suất của activity. Nhằm tránh làm ảnh hưởng đến hiệu suất ứng dụng, bạn nên khởi động một luồng mới bên trong service.

Về cơ bản, bạn cần mở rộng hai lớp sau để tạo một service dạng khởi động:

`Service`

Đây là lớp cơ bản của mọi service. Khi mở rộng lớp này, quan trọng là bạn cần tạo một luồng mới trong lớp để thực hiện tất cả các công việc của service. Vì mặc định là service sử dụng luồng chính của ứng dụng, nên có thể làm chậm hiệu suất của bất cứ activity nào mà ứng dụng đang chạy.

`IntentService`

Đây là lớp con của `Service` sẽ sử dụng luồng công việc (worker thread) để xử lý toàn bộ yêu cầu khởi động, mỗi yêu cầu được xử lý tại một thời điểm. Đây là lựa chọn tốt nhất nếu bạn không yêu cầu service của mình xử lý cùng lúc nhiều yêu cầu. Tất cả những gì bạn cần làm là cài đặt `onHandleIntent()`, là phương thức sẽ nhận intent cho mỗi yêu cầu khởi động để bạn có thể thực hiện công việc chạy ngầm.

Các mục dưới đây sẽ hướng dẫn cài đặt service của bạn bằng cách sử dụng một trong những lớp trên.

Mở rộng lớp IntentService

Do hầu hết service dạng khởi động không cần xử lý cùng lúc nhiều yêu cầu (đây thực sự là một kịch bản đa luồng (multi-threading) nguy hiểm), nên việc sử dụng lớp `IntentService` để cài đặt service của bạn có thể là sự lựa chọn tốt nhất.

Lớp `IntentService` thực hiện các công việc sau:

- Tạo một luồng công việc mặc định thực thi tất cả các intent truyền tới phương thức `onStartCommand()` phân tách với luồng chính của ứng dụng.
- Tạo một hàng đợi (queue) công việc, truyền từng intent theo từng thời điểm tới phần cài đặt `onHandleIntent()`, để bạn không bao giờ phải lo lắng về đa luồng.
- Dừng service này sau khi tất cả các yêu cầu khởi động đã được xử lý, để bạn không bao giờ phải gọi phương thức `stopSelf()`.
- Cài đặt mặc định cho phương thức `onBind()` trả về `null`.
- Cài đặt mặc định cho phương thức `onStartCommand()` gửi intent tới hàng đợi công việc, sau đó tới phần cài đặt `onHandleIntent()` của bạn.

Mọi điều trình bày ở trên minh chứng cho thực tế rằng tất cả những việc bạn cần làm là cài đặt phương thức `onHandleIntent()` để thực hiện công việc mà client yêu cầu. (Mặc dù vậy, bạn cũng cần cung cấp một phương thức khởi tạo đơn giản cho service này).

Sau đây là một ví dụ về cài đặt IntentService:

```
public class HelloIntentService extends IntentService {

    /**
    * Cần một phương thức khởi tạo, và phải gọi phương thức khởi tạo *
    của lớp cha IntentService(String)với tên của luồng công việc.
    */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
    * IntentService gọi phương thức này trong luồng công việc mặc định
    * Khi phương thức trả về kết quả, IntentService dừng
    * service khi thích hợp.
    */
    @Override
    protected void  onHandleIntent(Intent intent){
        // Thông thường, bạn sẽ thực hiện một vài thao tác tại đây,
        // chẳng hạn như tải file về.
        // Trong ví dụ này, chúng ta chỉ tạm nghỉ (sleep)
        // chương trình trong vòng 5 giây.
        long endTime = System.currentTimeMillis() + 5*1000;
        while(System.currentTimeMillis()< endTime){
            synchronized(this){
                try {
                    wait(endTime -System.currentTimeMillis());
                } catch(Exception e) {
                }
            }
        }
    }
}
```

Tất cả những gì bạn cần là: Một phương thức khởi tạo và cài đặt phương thức onHandleIntent().

Nếu bạn cũng quyết định ghi đè các phương thức callback khác, chẳng hạn như onCreate(), onStartCommand() hoặc onDestroy(), hãy đảm bảo bạn gọi phương thức cha của chúng, để IntentService có thể xử lý đúng chu kỳ sống của luồng công việc.

Ví dụ, phương thức onStartCommand() phải trả về các thao tác được cài đặt mặc định (đây là cách gửi intent đến onHandleIntent()):

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this,"service starting",Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent, flags, startId);
}
```

Bên cạnh phương thức `onHandleIntent()`, chỉ có một phương thức mà từ đó, bạn không cần gọi lớp cha là `onBind()` (chỉ cần cài đặt để service của bạn được liên kết với thành phần ứng dụng khác).

Ở mục tiếp theo, bạn sẽ được hướng dẫn cài đặt một loại service tương tự bằng cách mở rộng lớp `Service` cơ bản. Phương pháp này sử dụng nhiều mã hơn, nhưng sẽ phù hợp nếu bạn cần xử lý cùng lúc nhiều yêu cầu khởi động.

Mở rộng lớp Service

Như bạn đã thấy ở mục trước, sử dụng `IntentService` để cài đặt một service dạng khởi động là việc rất đơn giản. Tuy nhiên, nếu cần service xử lý đa luồng (thay vì xử lý yêu cầu khởi động thông qua một hàng đợi công việc), bạn có thể mở rộng lớp `Service` để xử lý từng intent.

Để so sánh, mã ví dụ dưới đây cài đặt lớp `Service` thực hiện công việc hoàn toàn giống ví dụ trên (sử dụng `IntentService`). Cụ thể, với mỗi yêu cầu khởi động, chương trình sẽ sử dụng một luồng công việc để thực hiện công việc và chỉ xử lý một yêu cầu vào một thời điểm:

```
public classHelloService extends Service {
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;

  // Trình xử lý (handler) nhận thông điệp từ luồng
  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
      @Override
      public void handleMessage(Message msg){
           // Thông thường, bạn sẽ thực hiện một vài thao tác tại đây,
           // chẳng hạn như tải file về. Trong ví dụ này, chúng ta
           // chỉ tạm nghỉ (sleep) chương trình trong vòng 5 giây.
          long endTime =System.currentTimeMillis()+5*1000;
          while(System.currentTimeMillis()< endTime){
              synchronized(this){
                  try {
```

```
                    wait(endTime -System.currentTimeMillis());
                } catch(Exception e) {
                }
            }
        }
       // Dừng service bằng cách sử dụng startId, để không phải dừng
        // service này khi đang xử lý công việc khác
        stopSelf(msg.arg1);
    }
}

@Override
public void onCreate(){
   // Khởi động luồng chạy service này. Lưu ý, chúng ta tạo một luồng
  // riêng, bởi vì service thường chạy trong luồng chính của tiến trình,
  // đây là luồng mà chúng ta không muốn chặn. Chúng ta cũng có thể
  // đặt cho luồng này mức ưu tiên là chạy ngầm, để các thao tác
  // cần nhiều CPU sẽ không ngắt giao diện người dùng của chúng ta.
  HandlerThread thread = new HandlerThread("ServiceStartArguments",
          Process.THREAD_PRIORITY_BACKGROUND);
  thread.start();

  // Lấy trình lặp (looper) của HandlerThread và sử dụng trình này
  // cho Handler (trình xử lý) của chúng ta
  mServiceLooper = thread.getLooper();
  mServiceHandler == new ServiceHandler(mServiceLooper);
}

@Override
public int onStartCommand(Intent intent,int flags,int startId){
   Toast.makeText(this,"service starting",Toast.LENGTH_SHORT).show();

    // Với mỗi yêu cầu khởi động, gửi thông điệp để khởi động
    // công việc và gửi đi start ID để chúng ta có thể nhận biết
    // yêu cầu sẽ dừng khi hoàn thành việc này.
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    mServiceHandler.sendMessage(msg);
    // Nếu service này bị dừng, sau khi trả lại kết quả
    // từ đây, khởi động lại service
    return START_STICKY;
}
```

```
    @Override
    public IBinder onBind(Intent intent) {
        // Chúng ta không cung cấp liên kết, do đó trả về null
        return null;
    }

    @Override
    public void onDestroy() {
      Toast.makeText(this,"service done",Toast.LENGTH_SHORT).show();
    }
}
```

Như bạn thấy, có nhiều việc phải làm hơn là sử dụng `IntentService`.

Tuy nhiên, do bạn xử lý mỗi lời gọi tới phương thức `onStartCommand()` của mình, nên bạn có thể thực hiện cùng lúc nhiều yêu cầu. Đó không phải những gì ví dụ trên thực hiện nhưng nếu muốn, bạn có thể tạo một luồng mới cho từng yêu cầu và chạy đúng cách (thay vì chờ cho yêu cầu trước đó hoàn thành).

Lưu ý, phương thức `onStartCommand()` phải trả về một số nguyên (integer). Số nguyên này là một giá trị mô tả cách hệ thống nên thực hiện với service khi hệ thống ép dừng nó (như đã bàn ở trên, cài đặt mặc định của `IntentService` sẽ xử lý việc này, mặc dù bạn có thể tự xử lý nó). Giá trị trả về từ phương thức `onStartCommand()` phải là một trong những hằng sau:

`START_NOT_STICKY`

> Nếu hệ thống dừng service sau khi phương thức `onStartCommand()` trả về, *không* tạo lại service, trừ phi có intent ở trạng thái chờ gửi đi. Đây là lựa chọn an toàn nhất để tránh chạy service khi không cần thiết, cũng như khi ứng dụng của bạn có thể khởi động lại bất cứ công việc nào chưa hoàn thành.

`START_STICKY`

> Nếu hệ thống dừng service sau khi phương thức `onStartCommand()` trả về, hãy tạo lại service đó và gọi `onStartCommand()`, nhưng *không* gửi lại intent cuối cùng. Thay vào đó, hệ thống sẽ gọi phương thức `onStartCommand()` với một intent có giá trị `null`, trừ phi có intent ở trạng thái chờ khởi động service này; trong trường hợp đó, những intent này sẽ được gửi đi. Cách xử lý này phù hợp với các trình phát đa phương tiện (media player), hoặc những service tương tự, là các service không thực thi lệnh, nhưng chạy vô hạn định và chờ thực hiện công việc.

`START_REDELIVER_INTENT`

> Nếu hệ thống dừng service sau khi phương thức `onStartCommand()` trả về, hãy tạo lại service và gọi phương thức `onStartCommand()` với intent cuối cùng được gửi tới service. Bất cứ intent ở trạng thái chờ nào cũng sẽ lần lượt được gửi đi. Cách xử lý này phù hợp với các service đang chủ động thực hiện công việc cần được khôi phục lại ngay lập tức, chẳng hạn như việc tải file về.

Để tìm hiểu thêm về các hằng trả về này, xem tài liệu tham khảo tương ứng với mỗi hằng.

Khởi động service

Bạn có thể khởi động một service từ activity hoặc thành phần khác cấu thành ứng dụng bằng việc truyền một `Intent` (xác định service để khởi động) vào phương thức `startService()`. Hệ thống Android gọi phương thức `onStartCommand()` của service này và truyền `Intent` trên cho service đó. (Lời khuyên cho bạn là đừng bao giờ gọi trực tiếp phương thức `onStartCommand()`).

Ví dụ, một activity có thể khởi động service ví dụ trong mục trước (HelloService) bằng cách sử dụng một intent tường minh với phương thức `startService()`:

```
Intent intent == new Intent(this, HelloService.class);
startService(intent);
```

Phương thức `startService()` trả về kết quả ngay lập tức và hệ thống Android gọi phương thức `onStartCommand()` của service này. Nếu service chưa chạy, hệ thống sẽ gọi phương thức `onCreate()` trước, sau đó gọi `onStartCommand()`.

Nếu service không cung cấp liên kết, intent sẽ được gửi đi với phương thức `startService()`, cũng là chế độ duy nhất để kết nối giữa thành phần của ứng dụng và service. Tuy nhiên, nếu bạn muốn service gửi kết quả trở lại, client khởi động service có thể tạo một `PendingIntent` để broadcast (bằng phương thức `getBroadcast()`) và phát `PendingIntent` này tới service trong Intent khởi động service. Nhờ đó, service có thể dùng broadcast để gửi kết quả đi.

Nếu có nhiều yêu cầu khởi động service thì sẽ có nhiều lời gọi phương thức `onStartCommand()` tương ứng. Tuy nhiên, chỉ cần có một yêu cầu dừng service (bằng phương thức `stopSelf()` hoặc `stopService()`) là có thể dừng service được.

Dừng service

Một service dạng khởi động phải quản lý vòng đời của chính nó. Nghĩa là, hệ thống sẽ không dừng hoặc hủy service này, trừ phi hệ thống cần khôi phục bộ nhớ và service tiếp tục chạy sau khi phương thức `onStartCommand()` trả về. Do đó, service phải tự dừng bằng cách gọi phương thức `stopSelf()`, hoặc gọi thành phần khác có thể dừng service này bằng cách gọi phương thức `stopService()`.

Khi được yêu cầu dừng với phương thức `stopSelf()` hoặc `stopService()`, hệ thống sẽ hủy service ngay khi có thể.

Tuy nhiên, nếu service của bạn xử lý nhiều yêu cầu `onStartCommand()` cùng lúc, bạn không nên dừng service khi đang thực hiện xử ký một yêu cầu khởi động, bởi vì bạn có thể nhận một yêu cầu khởi động mới (việc dừng service ở cuối yêu cầu đầu tiên có thể sẽ kết thúc ở yêu cầu thứ hai). Để tránh vấn đề này, bạn có thể sử dụng phương thức `stopSelf(int)` để đảm bảo rằng yêu cầu dừng service luôn dựa trên yêu cầu khởi động mới nhất. Cụ thể, khi gọi phương thức `stopSelf(int)`, bạn truyền vào ID của yêu cầu khởi động (cũng chính là startId được gửi tới phương

thức `onStartCommand()`) tương ứng với yêu cầu dừng service của bạn. Sau đó, nếu service đã nhận một yêu cầu khởi động mới trước khi bạn có thể gọi phương thức `stopSelf(int)`, ID sẽ không phù hợp và service sẽ không dừng.

Chú ý: Việc ứng dụng của bạn dừng service của nó khi đã hoàn thành công việc là rất quan trọng, nhằm tránh lãng phí tài nguyên hệ thống và tiêu tốn điện năng. Nếu cần, các thành phần khác có thể dừng service bằng cách gọi phương thức `stopService()`. Thậm chí, nếu có thể kích hoạt liên kết cho service này, bạn phải luôn tự dừng service nếu đã từng nhận được lời gọi tới `onStartCommand()`.

Để tìm hiểu thêm về vòng đời của một service, xem mục "Quản lý vòng đời của service" bên dưới.

Tạo service có liên kết

Service có liên kết (bound service) là thành phần cho phép các thành phần ứng dụng khác liên kết với nó bằng cách gọi phương thức `bindService()` tương ứng để tạo một kết nối lâu dài (và nhìn chung là không cho phép các thành phần khác *khởi động* nó bằng lời gọi phương thức `startService()`).

Bạn nên tạo một service liên kết khi muốn tương tác với service kiểu này từ activity và các thành phần khác trong ứng dụng, hoặc cung cấp một số chức năng của ứng dụng cho những ứng dụng khác, thông qua việc giao tiếp liên tiến trình (interprocess communication - IPC).

Muốn tạo một service liên kết, bạn phải cài đặt phương thức callback `onBind()` để trả về một `IBinder` định nghĩa giao diện dùng cho việc kết nối với service. Sau đó, các thành phần ứng dụng khác có thể gọi phương thức `bindService()` để truy xuất vào giao diện này và bắt đầu gọi các phương thức trên service. Service chỉ tồn tại để phục vụ thành phần ứng dụng được gắn với nó; do đó, khi không còn thành phần gắn với service nữa, hệ thống sẽ hủy service (bạn *không* cần dừng service liên kết như cách phải thực hiện với service dạng khởi động thông qua phương thức `onStartCommand()`).

Để tạo một service liên kết, việc cần làm trước tiên là bạn phải định nghĩa giao diện xác định cách một client giao tiếp với service này. Giao diện giữa service và client phải là một bản cài đặt của `IBinder` và cũng là kết quả mà service của bạn phải trả về từ phương thức callback `onBind()`. Khi client nhận `IBinder`, client có thể bắt đầu tương tác với service thông qua giao diện này.

Cùng lúc, nhiều client có thể gắn với một service. Khi đang tương tác với service, client gọi phương thức `unbindService()` để ngắt liên kết. Khi không còn client liên kết với service, hệ thống sẽ hủy service.

Có nhiều cách để cài đặt một service có liên kết và bản cài đặt service này phức tạp hơn so với service dạng khởi động. Do đó, phần thảo luận về service có liên kết nằm trong một tài liệu riêng về "Bound Services" ("Service có liên kết").

Gửi thông báo tới người dùng

Khi đang chạy, một service có thể thông báo với người dùng về các sự kiện bằng cách sử dụng Thông báo nhanh (Toast Notifications) hoặc Thông báo qua thanh trang thái (Status Bar Notifications).

Thông báo nhanh là thông điệp xuất hiện tạm thời trên mặt cửa sổ hiện tại, sau đó biến mất. Trong khi đó, một thông báo qua thanh trạng thái cung cấp một biểu tượng trên thanh trạng thái cùng với một thông điệp, từ đó người dùng có thể chọn để thực hiện một hoạt động (chẳng hạn như khởi động một activity).

Thông thường, thông báo qua thanh trạng thái là kỹ thuật tốt nhất khi đã hoàn thành một số công việc chạy ngầm (chẳng hạn như hoàn thành tải file về) và giờ thì người dùng có thể thao tác trên đó. Khi người dùng chọn thông báo từ view mở rộng, thông báo có thể khởi động một activity (chẳng hạn như hiển thị file vừa tải về).

Xem phần hướng dẫn lập trình về Toast Notifications hoặc Status Bar Notifications để tìm hiểu thêm thông tin.

Chạy service trong chế độ màn hình chính

Service màn hình chính (foreground service) là service được cho là những thứ mà người dùng có thể chủ động thấy được và do đó không phải là thành phần mà hệ thống sẽ hủy khi còn ít bộ nhớ. Mỗi service màn hình chính phải cung cấp một thông báo cho thanh trạng thái, nằm dưới tiêu đề "Ongoing" (đang tiến hành), điều này có nghĩa là thông báo trên không thể bị bỏ đi, trừ phi service bị dừng hoặc bị loại khỏi màn hình chính.

Ví dụ, nên thiết lập một trình chơi nhạc (music player) phát nhạc từ một service để chạy trong chế độ màn hình chính, vì người dùng nhận thức rõ về hoạt động của trình này. Thông báo trong thanh trạng thái có thể hiển thị bài hát hiện tại, đồng thời cho phép người dùng khởi động một activity để tương tác với trình nghe nhạc.

Để yêu cầu service được chạy trong chế độ màn hình chính, gọi phương thức `startForeground()`. Phương thức này sử dụng hai tham số: Một số nguyên xác định thông báo và đối tượng `Notification` cho thanh trạng thái. Ví dụ:

```
Notification notification = new Notification(R.drawable.icon,
      getText(R.string.ticker_text), System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this,0,
notificationIntent,0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
      getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

Chú ý: ID số nguyên bạn cung cấp cho phương thức `startForeground()` phải khác 0.

Để loại service khỏi chế độ màn hình chính, gọi phương thức `stopForeground()`. Phương thức này nhận một tham số kiểu boolean, cho biết liệu có nên loại thông báo trên thanh trạng thái hay không. Phương thức này *không* dừng service. Tuy nhiên, nếu bạn dừng service trong khi service này vẫn chạyở màn hình chính, thông báo này cũng sẽ bị loại.

Để tìm hiểu thêm về thông báo, xem phần "Creating Status Bar Notifications" ("Tạo thông báo qua thanh trạng thái").

Quản lý vòng đời của service

Vòng đời của service đơn giản hơn rất nhiều so với vòng đời của một activity. Tuy nhiên, việc chú trọng tới cách service được tạo và bị hủy thậm chí còn quan trọng hơn, vì một service có thể chạy ngầm mà người dùng không biết được.

Vòng đời của service - từ khi được tạo tới khi bị hủy - có thể tuân theo hai cách khác nhau sau:

Service dạng khởi động

Service được tạo khi thành phần khác của ứng dụng gọi phương thức `startService()`. Sau đó, service sẽ chạy vô hạn định và phải tự dừng bằng cách gọi phương thức `stopSelf()`. Thành phần khác cũng có thể dừng service này bằng cách gọi phương thức `stopService()`. Khi service bị dừng, hệ thống sẽ hủy service này.

Service có liên kết

Service được tạo khi thành phần khác (một client) gọi phương thức `bindService()`. Sau đó, client sẽ kết nối với service này thông qua giao diện `IBinder`. Client cũng có thể ngắt kết nối bằng cách gọi phương thức `unbindService()`. Nhiều client có thể cùng liên kết với cùng một service và khi tất cả các client này ngắt liên kết, hệ thống sẽ hủy service. (Service kiểu này *không* cần tự dừng).

Hai cách này không hoàn toàn tách biệt. Bởi vì, bạn có thể liên kết với một service đã được khởi động bằng phương thức `startService()`. Ví dụ, một service âm nhạc chạy ngầm có thể được khởi tạo thông qua lời gọi phương thức `startService()` với một `Intent` xác định nhạc được phát. Sau đó, khả năng là khi người dùng muốn thực hiện một số điều khiển trên trình chơi nhạc này hoặc lấy thông tin về bài hát hiện tại, một activity có thể liên kết với service này bằng lời gọi phương thức `bindService()`. Trong trường hợp như vậy, phương thức `stopService()` hoặc `stopSelf()` không thực sự dừng service cho đến khi tất cả các client ngắt liên kết.

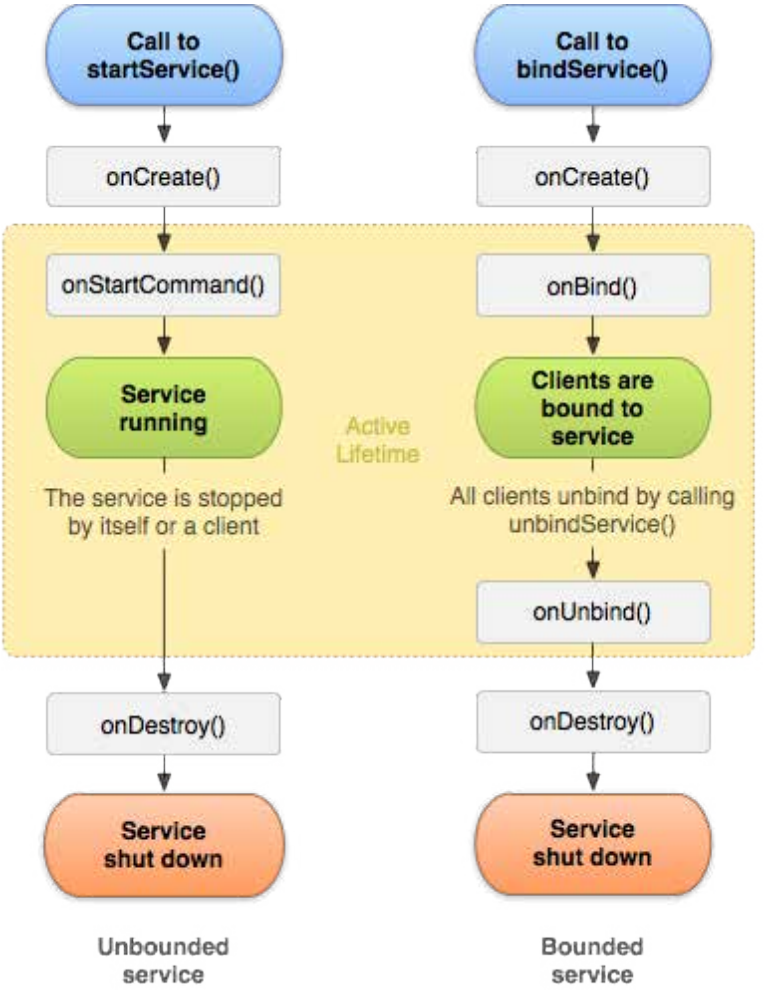
Cài đặt các phương thức callback liên quan đến vòng đời

Tương tự activity, một service có các phương thức callback liên quan đến vòng đời mà bạn có thể cài đặt để điều khiển những thay đổi bên trong trạng thái của service và thực hiện các công việc tại thời điểm thích hợp. Service khung (skeleton service) dưới đây minh họa tất cả các phương thức về vòng đời:

```
public class ExampleService extends Service {
    int mStartMode;       // cho biết cách xử lý khi service bị dừng
    IBinder mBinder;      // giao diện cho client liên kết
    boolean mAllowRebind;// cho biết có nên sử dụng
                         // phương thức onRebind hay không
    @Override
    public void onCreate() {
        // Service đang được tạo
    }
```

```
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Service đang được khởi động, do một lời gọi
        // tới phương thức startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // Một client đang liên kết với service
        // qua phương thức bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
       // Tất cả các client ngắt liên kết bằng phương thức unbindService()
       return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // Một client sẽ được liên kết với service
        // bằng phương thức bindService(),
        // sau khi phương thức onUnbind() được gọi
    }
    @Override
    public void onDestroy() {
        // Service không còn được sử dụng và đang bị hủy
    }
}
```

Ghi chú: Không giống như các phương thức callback liên quan đến vòng đời của activity, bạn *không* cần gọi bản cài đặt lớp cha của những phương thức callback này.



Call to startService()	Gọi phương thức startService()
Call to bindService()	Gọi phương thức bindService()
Service running	Service đang chạy
Clients are bound to service	Các client được liên kết với service
Active Lifetime	Vòng đời sống
The service is stopped by itself or a client	Service tự dừng hoặc một client đã dừng nó
All clients unbind by calling unbindService()	Tất cả client đều ngắt liên kết bằng cách gọi phương thức unbindService()
Service shut down	Service bị dừng hoạt động
Service shut down	Service bị dừng hoạt động
Unbounded service	Service không có liên kết
Bound service	Service có liên kết

Hình 2. Vòng đời của service. Biểu đồ bên trái minh họa vòng đời khi service được tạo bằng phương thức `startService()`. Biểu đồ bên phải minh họa vòng đời khi service được tạo bằng phương thức `bindService()`.

Bằng cách cài đặt những phương thức này, bạn có thể điều khiển hai vòng lặp lồng nhau của vòng đời service:

- **Toàn bộ vòng đời (entire lifetime)** của một service là khoảng thời gian giữa thời điểm gọi phương thức `onCreate()` và thời điểm phương thức `onDestroy()` trả về kết quả. Tương tự activity, một service thực hiện các thiết lập ban đầu của nó trong phương thức `onCreate()` và giải phóng tất cả tài nguyên đang giữ trong phương thức `onDestroy()`. Ví dụ, một service nghe nhạc có thể tạo luồng phát nhạc trong phương thức `onCreate()`, sau đó dừng luồng này trong phương thức `onDestroy()`.

  Tất cả các service đều gọi hai phương thức `onCreate()` và `onDestroy()`, bất kể chúng được tạo bằng phương thức `startService()` hay `bindService()`.

- **Vòng đời sống (active lifetime)** của service bắt đầu bằng lời gọi phương thức `onStartCommand()` hoặc `onBind()`. Mỗi phương thức sẽ xử lý `Intent` được chuyển đến từ phương thức `startService()` hay `bindService()`.

- Nếu service được khởi động, vòng đời sống sẽ kết thúc cùng thời điểm với thời điểm kết thúc toàn bộ vòng đời (service vẫn hoạt động ngay cả sau khi phương thức `onStartCommand()` trả về). Nếu service có liên kết, vòng đời sống sẽ kết thúc khi phương thức `onUnbind()` trả về.

- **Ghi chú**: Mặc dù một service dạng khởi động bị dừng bởi lời gọi phương thức `stopSelf()` hoặc `stopService()`, song không có phương thức callback nào tương ứng cho service (không có phương thức callback `onStop()`). Do vậy, trừ phi service được liên kết với một client, hệ thống mới hủy service này khi nó bị dừng - `onDestroy()` chỉ là phương thức callback được nhận.

- Hình 2 minh họa các phương thức callback điển hình của một service. Mặc dù hình này tách biệt những service do phương thức `startService()` tạo ra và các service tạo thành từ phương thức `bindService()`, nhưng hãy nhớ rằng bất cứ service nào, không quan trọng là được khởi động theo cách nào, cũng có khả năng cho phép client liên kết với service đó. Do vậy, một service được khởi động đầu tiên bằng phương thức `onStartCommand()` (khi một client gọi `startService()`) vẫn có thể nhận một lời gọi tới `onBind()` (khi client gọi `bindService()`).

- Để tìm hiểu thêm về cách tạo service cung cấp liên kết, xem tài liệu "Bound Services" ("Service có liên kết"). Chi tiết về phương thức callback `onRebind()` được trình bày trong mục "Managing the Lifecycle of a Bound Service" ("Quản lý vòng đời của service có liên kết") của tài liệu trên.

13. Lớp BroadcastReceiver

Tổng quan về lớp BroadcastReceiver

`BroadcastReceiver` là lớp cơ sở chứa mã dùng để nhận intent do phương thức `sendBroadcast()` gửi tới.

Nếu bạn không cần gửi broadcast trên nhiều ứng dụng, hãy cân nhắc dùng lớp này với lớp `LocalBroadcastManager` thay vì thêm nhiều chức năng chung sẽ được mô tả sau đây. Việc này sẽ giúp bạn cài đặt hiệu quả hơn nhiều (không cần các giao tiếp tiến trình chéo), đồng thời cho phép bạn tránh phải bận tâm đến những vấn đề bảo mật liên quan tới việc các ứng dụng khác có thể nhận hoặc gửi tín hiệu broadcast của bạn.

Bạn có thể đăng ký động một thể hiện của lớp này bằng phương thức `Context.registerReceiver()`, hoặc phát hành tĩnh một bản cài đặt thông qua thẻ `<receiver>` trong file `AndroidManifest.xml`.

Ghi chú: Nếu đăng ký một trình nhận trong bản cài đặt phương thức `Activity.onResume()`, bạn nên hủy đăng ký ở phương thức `Activity.onPause()`. (Bạn sẽ không nhận intent khi đã dừng trình nhận, và điều này sẽ làm giảm những phụ tải (overhead) không cần thiết). Đừng hủy đăng ký trong `Activity.onSaveInstanceState()`, vì phương thức này sẽ không được gọi nếu người dùng quay lại ngăn xếp history (lịch sử).

Có hai lớp broadcast chính có thể được nhận:

Broadcast thường (normal broadcast) (được gửi đi bằng cách dùng phương thức `Context.sendBroadcast`) là kiểu broadcast không đồng bộ hoàn toàn. Tất cả các trình nhận broadcast chạy đều theo thứ tự không xác định, thường là cùng lúc. Cách truyền này hiệu quả hơn, nhưng cũng có nghĩa là trình nhận không thể sử dụng kết quả hoặc ngừng API (Application Programming Interface - Giao diện lập trình ứng dụng) tại đây.

Broadcast theo thứ tự (ordered broadcast) (được gửi đi bằng cách dùng phương thức `Context.sendOrderedBroadcast`) được gửi tới một trình nhận tại một thời điểm. Khi mỗi trình nhận thực thi theo thứ tự, trình này có thể nhân bản kết quả cho trình nhận tiếp theo, hoặc có thể ngừng hoàn toàn broadcast để không chuyển broadcast này tới trình nhận khác. Thứ tự chạy của các trình nhận có thể được quản lý bằng thuộc tính `android:priority` của bộ lọc intent phù hợp; những trình nhận có cùng mức ưu tiên sẽ chạy theo thứ tự bất kỳ.

Thậm chí, trong trường hợp broadcast thường, hệ thống có thể rơi vào một số tình thế có thể khôi phục việc broadcast tới một trình nhận riêng. Cụ thể, các trình nhận có thể yêu cầu tạo một tiến trình, để chỉ một trình nhận chạy riêng nhằm tránh các tiến trình mới làm quá tải hệ thống. Tuy nhiên, trong tình huống này, việc xử lý về nghĩa vẫn là không theo thứ tự: Các trình nhận này vẫn không thể trả về kết quả hoặc ngừng việc broadcast của chúng.

Lưu ý, mặc dù lớp Intent được dùng để gửi và nhận các tín hiệu broadcast, nhưng cơ chế Intent broadcast ở đây hoàn toàn khác so với những Intent được sử dụng để khởi động các Activity thông qua phương thức `Context.startActivity()`. Không có cơ chế để đối tượng `BroadcastReceiver` thấy hoặc bắt Intent được dùng với phương thức `startActivity()`; tương tự, khi broadcast một Intent, bạn sẽ không bao giờ tìm thấy hoặc khởi động một Activity. Về mặt ngữ nghĩa, đây là hai thao tác rất khác nhau: Việc khởi động một Activity bằng Intent là một thao tác chạy trên màn hình chính (foreground), làm thay đổi những gì người dùng hiện đang tương tác; việc broadcast một Intent là một hoạt động chạy ngầm mà người dùng thường không nhận thấy.

Lớp `BroadcastReceiver` (khi khởi động giống như một thành phần thông qua thẻ `<receiver>` trong file kê khai) là một thành phần quan trọng trong toàn bộ vòng đời của ứng dụng. Xem thêm về toàn bộ vòng đời của ứng dụng tại mục "Kiến thức cơ bản về ứng dụng".

Chủ đề này sẽ thảo luận về các nội dung:

1. Bảo mật.
2. Vòng đời của trình nhận.
3. Vòng đời của tiến trình.

Hướng dẫn phát triển chương trình

Để tìm hiểu thêm thông tin về cách sử dụng lớp này để nhận và xử lý intent, mời bạn đọc phần hướng dẫn dành cho nhà phát triển "Intents and Intent Filters" ("Intent và bộ lọc Intent").

Bảo mật

Các trình nhận được sử dụng với các API (giao diện lập trình ứng dụng) `Context` bằng tính năng đa ứng dụng (cross-application) tự nhiên của chúng, nên bạn phải tính đến khả năng những ứng dụng khác có thể lạm dụng việc sử dụng của bạn trên các trình nhận này. Một số vấn đề cần cân nhắc sau:

- Namespace của Intent phải là toàn cục (global). Hãy đảm bảo tên action của Intent và các chuỗi khác phải được viết trong namespace của chính bạn; nếu không, bạn có thể vô tình gây xung đột với các ứng dụng khác.
- Khi bạn sử dụng phương thức `registerReceiver(BroadcastReceiver, IntentFilter)`, một ứng dụng *bất kỳ* có thể gửi broadcast tới trình nhận đã được đăng ký này. Bạn có thể điều khiển đối tượng được broadcast tới trình nhận thông qua các quyền được mô tả bên dưới.
- Khi bạn phát hành trình nhận trong file kê khai của ứng dụng và xác định bộ lọc intent cho file này, các ứng dụng khác có thể gửi broadcast tới trình nhận trên mà không cần quan tâm tới bộ lọc mà bạn xác định. Để tránh không cho những ứng dụng khác gửi broadcast tới, hãy cung cấp thuộc tính để trình nhận này không bị các tác nhân bên ngoài truy cập là `android:exported="false"`.
- Khi bạn sử dụng `sendBroadcast(Intent)` hoặc các phương thức liên quan, thường thì bất cứ ứng dụng nào khác cũng có thể nhận các broadcast này. Bạn có thể điều khiển đối tượng có thể nhận những broadcast này bằng các quyền được mô tả bên dưới. Một cách khác là khởi động với hằng `ICE_CREAM_SANDWICH`, bạn cũng có thể giới hạn việc broadcast một cách an toàn cho một ứng dụng bằng `Intent.setPackage`.

Các vấn đề trên sẽ không tồn tại nếu bạn sử dụng `LocalBroadcastManager`, do các intent tiến hành broadcast đối tượng này không bao giờ ra khỏi tiến trình hiện tại.

Các quyền truy cập có thể cho phép xác định hoặc trình gửi (sender) và trình nhận broadcast.

Để ép buộc một quyền khi gửi, bạn cung cấp một đối số *permission* khác null cho phương thức `sendBroadcast(Intent, String)` hoặc `sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle)`. Chỉ những trình nhận có quyền này (bằng cách yêu cầu quyền thông qua thẻ `<uses-permission>` trong file `AndroidManifest.xml`) mới có thể nhận broadcast.

Để ép buộc một quyền khi nhận, bạn cung cấp đối số *permission* khác null khi đăng ký trình nhận của bạn - khi gọi phương thức `registerReceiver(BroadcastReceiver, IntentFilter, String, android.os.Handler)`, hoặc trong thẻ tĩnh `<receiver>` của file `AndroidManifest.xml` do bạn phát triển. Chỉ các đối tượng broadcast (broadcaster) có quyền trên (bằng cách yêu cầu quyền trên với thẻ `<uses-permission>` trong file `AndroidManifest.xml` của đối tượng này) mới có thể gửi một Intent tới trình nhận.

Xem tài liệu hướng dẫn "Security and Permissions" ("Quyền và bảo mật") để tìm hiểu thêm thông tin về quyền và bảo mật.

Vòng đời của trình nhận

Đối tượng `BroadcastReceiver` chỉ tồn tại trong thời hạn của lời gọi tới phương thức `onReceive(Context, Intent)`. Khi mã của bạn trả kết quả về từ hàm này, hệ thống sẽ cho rằng đối tượng BroadcastReceiver đã hoàn thành và không còn hoạt động nữa.

Việc này có tác động quan trọng đến những gì bạn cài đặt trong phương thức `onReceiver(Context, Intent)`: Bất cứ thứ gì yêu cầu thao tác không đồng bộ đều không được xử lý tại đây, vì bạn sẽ phải trả kết quả về từ hàm này để xử lý thao tác không đồng bộ. Tuy nhiên, tại điểm đó, đối tượng `BroadcastReceiver` không còn hoạt động, nên hệ thống sẽ tự ý kết thúc tiến trình của đối tượng này trước khi thao tác không đồng bộ hoàn thành.

Cụ thể, bạn sẽ không hiển thị hộp thoại hoặc liên kết với một service từ bên trong đối tượng `BroadcastReceiver`. Đối với hộp thoại, bạn có thể sử dụng API của `NotificationManager`. Đối với service, bạn có thể dùng `Context.startService()` để gửi lệnh tới service này.

Vòng đời của tiến trình

Một tiến trình đang thực thi đối tượng `BroadcastReceiver` (nghĩa là, đang chạy mã trong phương thức `onReceive(Context, Intent)`) được xem là tiến trình chạy trên màn hình chính (foreground) và sẽ được hệ thống duy trì, trừ phi bộ nhớ cực kỳ quá tải.

Khi bạn trả kết quả về từ phương thức `onReceive()`, đối tượng `BroadcastReceiver` không còn hoạt động nữa, tiến trình chứa đối tượng này sẽ chỉ quan trọng đối với những thành phần của ứng dụng khác chạy trong đó. Điều này cực kỳ quan trọng, vì nếu tiến trình chỉ host đối tượng `BroadcastReceiver` (trường hợp thường gặp đối với các ứng dụng mà người dùng không bao giờ hoặc hiện đang không tương tác), sau khi trả kết quả về từ phương thức `onReceive()`, hệ thống sẽ coi tiến trình này là rỗng, đồng thời nhanh chóng ngắt nó để dành tài nguyên cho những tiến trình khác quan trọng hơn.

Điều này có nghĩa là, đối với những thao tác tốn thời gian, bạn thường sử dụng `Service` kết hợp với `BroadcastReceiver` để giữ tiến trình chứa tồn tại trong suốt thời gian diễn ra thao tác của bạn.

Tổng kết

Lớp lồng nhau (nested class)		
Lớp	BroadcastReceiver.PendingResult	Trạng thái của kết quả được giữ lại cho broadcast receiver.
Phương thức khởi tạo public		
	BroadcastReceiver()	
Phương thức public		
final void	abortBroadcast() Thiết lập cờ (flag) cho biết trình nhận này nên hủy broadcast hiện tại; phương thức này chỉ làm việc với những tín hiệu broadcast được gửi thông qua phương thức `Context.sendOrderedBroadcast`.	
final void	`clearAbortBroadcast`() Xóa cờ cho biết trình nhận này hủy broadcast hiện tại.	
final boolean	getAbortBroadcast() Trả về cờ cho biết liệu trình nhận này có nên hủy broadcast hiện tại hay không.	
final boolean	getDebugUnregister() Trả về giá trị cuối cùng nhận được từ phương thức `setDebugUnregister(boolean)`.	
final int	getResultCode() Truy xuất mã kết quả hiện tại, do trình nhận trước thiết lập.	
final String	getResultData() Truy xuất dữ liệu kết quả hiện tại, do trình nhận trước thiết lập.	
final Bundle	getResultExtras (boolean makeMap) Truy xuất dữ liệu kết quả phụ trợ (result extra data) hiện tại, do trình nhận trước thiết lập.	
final BroadcastReceiver.PendingResult	goAsync() Phương thức này có thể được ứng dụng gọi trong phương thức `onReceive(Context, Intent)` cho phép giữ tín hiệu broadcast hoạt động sau khi được trả về từ phương thức `onReceive(Context, Intent)`.	

final boolean	isInitialStickyBroadcast() Trả về true nếu trình nhận đang xử lý giá trị khởi tạo của tín hiệu broadcast kiểu sticky (dính) - giá trị này được broadcast sau cùng và hiện đang được lưu giữ trong cache sticky; bởi vậy; tại thời điểm đó, giá trị này không phải là kết quả trực tiếp của một broadcast.
final boolean	isOrderedBroadcast() Trả về giá trị true nếu trình nhận đang xử lý một broadcast có thứ tự.
abstract void	onReceive(Context context, Intent intent) Phương thức này được gọi khi đối tượng BroadcastReceiver đang nhận một Intent broadcast.
IBinder	peekService(Context myContext, Intent service) Cung cấp đối tượng liên kết tới service đang chạy.
final void	setDebugUnregister(boolean debug) Cung cấp tập các điều khiển trợ giúp gỡ lỗi cho những lời gọi không phù hợp với phương thức `Context.registerReceiver()`.
final void	setOrderedHint(boolean isOrdered) Phương thức này được sử dụng bên trong, thiết lập gợi ý xem đối tượng BroadcastReceiver này có đang chạy trong chế độ có thứ tự hay không.
final void	setResult(int code, String data, Bundle extras) Thay đổi tất cả các dữ liệu kết quả trả về từ việc phát broadcast này; chỉ làm việc với những tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast`.
final void	setResultCode(int code) Thay đổi mã kết quả hiện tại của tín hiệu broadcast này; phương thức này chỉ làm việc với những tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast.`
final void	setResultData(String data) Thay đổi dữ liệu kết quả hiện tại của việc broadcast này; phương thức này chỉ làm việc với những tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast`.
final void	setResultExtras(Bundle extras) Thay đổi giá trị phụ trợ của kết quả hiện tại đối với việc broadcast này; chỉ làm việc với những tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast`.
[Mở rộng] Các phương thức `thừa kế (inherited method)`	
▶ Thuộc lớp java.lang.Object	

Các phương thức khởi tạo public

public BroadcastReceiver()

Được đưa vào API cấp 1 (API level 1)

Các phương thức public

---

public final void `abortBroadcast()`

Được đưa vào API cấp 1 (API level 1)

Thiết lập cờ cho biết trình nhận này nên hủy việc broadcast hiện tại; chỉ làm việc với những tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast`. Phương thức này ngăn không cho các broadcast receiver khác không nhận được tín hiệu broadcast. Phương thức **public final void** `abortBroadcast()` vẫn gọi phương thức `onReceive(Context, Intent)` của đối tượng `BroadcastReceiver` do lời gọi phương thức `Context.sendOrderedBroadcast` truyền đến.

Phương thức public final void `abortBroadcast()` **không làm việc với các tín hiệu broadcast không theo thứ tự, chẳng hạn như những tín hiệu broadcast được gửi tới từ phương thức** `Context.sendBroadcast`.

public final boolean `clearAbortBroadcast()`

Được đưa vào API cấp 1 (API level 1)

Xóa cờ cho biết trình nhận này nên hủy việc broadcast hiện tại.

public final boolean getAbortBroadcast()

Được đưa vào API cấp 1 (API level 1)

Trả về cờ cho biết liệu trình nhận này có nên hủy việc broadcast hiện tại hay không.

Kết quả trả về

- Trả về true, nếu hủy việc broadcast này.

public final boolean `getDebugUnregister()`

Được đưa vào API cấp 1 (API level 1)

Trả về giá trị cuối cùng nhận được từ phương thức `setDebugUnregister(boolean)`.

public final int `getResultCode()`

Truy xuất mã kết quả hiện tại, do trình nhận trước thiết lập.

Kết quả trả về

- Mã kết quả hiện tại kiểu int.

public final String `getResultData()`

Được đưa vào API cấp 1 (API level 1)

Truy xuất dữ liệu kết quả hiện tại, do trình nhận trước thiết lập. Thường là null.
Kết quả trả về

- Dữ liệu kết quả hiện tại kiểu String; có thể là null.

public final Bundle `getResultExtras` **(boolean makeMap)**

Được đưa vào API cấp 1 (API level 1)

Truy xuất dữ liệu kết kết quả phụ trợ (dữ liệu extra) hiện tại, do trình nhận trước thiết lập. Bất cứ thay đổi nào bạn thực hiện trên Map (đối tượng bản đồ ánh xạ) trả về sẽ được nhân bản cho trình nhận tiếp theo.

Tham số
`makeMap`

Nếu true thì sẽ thiết lập cho bạn một Map rỗng mới khi Map hiện tại là null; Nếu false, bạn được chuẩn bị để nhận một Map null.

Kết quả trả về

- Bản đồ ánh xạ phụ trợ hiện tại kiểu Map.

public final `BroadcastReceiver.PendingResultgoAsync`()

Được đưa vào API cấp 11 (API level 11)

Phương thức này có thể được một ứng dụng gọi trong phương thức `onReceive(Context, Intent)` để cho phép ứng dụng giữ việc broadcast này vẫn hoạt động sau khi phương thức `onReceive(Context, Intent)` trả về kết quả. *Không* thay đổi thời gian chờ phản hồi tín hiệu broadcast mong muốn (hoàn thành trong vòng 10 giây), nhưng cho phép cài đặt để chuyển những công việc liên quan sang luồng khác, nhằm tránh ảnh hưởng tới luồng giao diện người dùng chính do IO (đầu vào/đầu ra) của đĩa.

Kết quả trả về

- Trả về `BroadcastReceiver.PendingResult` đại diện cho kết quả của tín hiệu broadcast đang hoạt động. Bản thân đối tượng `BroadcastRecord` đã không còn hoạt động nữa; tất cả các dữ liệu và tương tác khác phải đi qua API của `BroadcastReceiver.PendingResult.` Phương thức `PendingResult.finish()` phải được gọi khi tiến trình xử lý broadcast hoàn thành.

public final boolean `isInitialStickyBroadcast()`

Được đưa vào API cấp 5 (API level 5)

Trả về true, nếu trình nhận đang xử lý giá trị khởi tạo của tín hiệu broadcast kiểu sticky - giá trị này được broadcast sau cùng và hiện đang được lưu giữ trong cache sticky; do vậy, tại thời điểm đó, giá trị này không phải là kết quả trực tiếp của một broadcast.

public final boolean `isOrderedBroadcast()`

Được đưa vào API cấp 5 (API level 5)

Trả về true, nếu trình nhận đang xử lý một broadcast có thứ tự.

public abstract void `onReceive` (Context context, Intent intent)

Được đưa vào API cấp 1 (API level 1)

Phương thức này được gọi khi đối tượng `BroadcastReceiver` đang nhận một Intent broadcast. Suốt thời gian này, bạn có thể sử dụng các phương thức khác trong `BroadcastReceiver` để hiển thị/chỉnh sửa giá trị kết quả hiện tại. Đây là phương thức luôn được gọi trong luồng chính của tiến trình xử lý phương thức, trừ phi bạn yêu cầu cụ thể để phương thức này được đưa vào luồng khác bằng cách sử dụng phương thức `registerReceiver(BroadcastReceiver, IntentFilter, String, android.os.Handler)`. Khi phương thức này chạy trên luồng chính, lời khuyên cho bạn là đừng bao giờ thực hiện các thao tác tốn thời gian trong phương thức (thời gian chờ (timeout) là 10 giây; quá thời gian này, hệ thống sẽ cho rằng trình nhận đã bị chặn và một ứng cử viên đã bị hủy). Bạn không nên cài đặt để bật hộp thoại popup trong phương thức onReceive().

Nếu đối tượng BroadcastReceiver được phát đi thông qua thẻ <receiver>, đối tượng này sẽ không hoạt động sau khi được trả về từ hàm này. Điều này có nghĩa rằng, bạn không nên thực hiện bất kỳ thao tác nào trả về kết quả một cách không đồng bộ - cụ thể, để tương tác với service, bạn nên sử dụng phương thức `startService(Intent)` thay vì `bindService(Intent, ServiceConnection, int)`. Nếu muốn tương tác với một service đang chạy, bạn có thể sử dụng phương thức `peekService(Context, Intent)`.

Bộ lọc Intent được dùng trong phương thức `registerReceiver(BroadcastReceiver, IntentFilter)` và trong file kê khai, ứng dụng *không* đảm bảo việc broadcast được sử dụng riêng. Bộ lọc intent là gợi ý để hệ điều hành biết cách tìm trình nhận phù hợp. Chúng ta có thể thiết kế để trình gửi có khả năng chọn broadcast tới các trình nhận cụ thể, bỏ qua việc dùng bộ lọc. Do vậy, bạn nên cài đặt sao cho phương thức `onReceive()` chỉ phản hồi những hoạt động đã biết, bỏ qua những Intent không muốn nhận về.

Tham số

context

Context mà trình nhận đang chạy.

intent

Intent *đang được nhận.*

public IBinder `peekService` (`Context` myContext, `Intent` service)

Được đưa vào API cấp 3 (API level 3)

Cung cấp một đối tượng liên kết (binder) tới một service đang chạy. Đây là phương thức đồng bộ và sẽ không khởi động service đích nếu service này chưa hoạt động. Do đó, phương thức này hoạt động tốt khi được gọi từ phương thức `onReceive(Context, Intent)`.

Tham số

myContext

Context đã truyền vào phương thức `onReceive(Context, Intent)`.

service

Là Intent chỉ ra service bạn muốn sử dụng. Xem thêm phương thức `startService(Intent)` *để tìm hiểu kỹ hơn.*

public final void `setDebugUnregister` **(boolean debug)**

Được đưa vào API cấp 1 (API level 1)

Cung cấp tập các điều khiển trợ giúp gỡ lỗi cho những lời gọi không phù hợp với phương thức `Context.registerReceiver()`. Nếu phương thức này được gọi với giá trị true, trước khi nhận kết quả trả về từ phương thức `registerReceiver()`, callstack (ngăn xếp lời gọi) cho phương thức `Context.unregisterReceiver()` sau sẽ tiếp tục được sử dụng, được in ra nếu sau đó có một lời gọi chưa đăng ký không hợp lệ được thực hiện. Lưu ý, phương thức này cần duy trì thông tin về `BroadcastReceiver` trong suốt thời gian chạy ứng dụng, trả về vết lỗi (leak) - chỉ dùng để gỡ lỗi.

public final void `setOrderedHint`**(boolean isOrdered)**

Được đưa vào API cấp 1 (API level 1)

Phương thức này được sử dụng bên trong, thiết lập gợi ý về việc đối tượng `BroadcastReceiver` này có đang chạy trong chế độ có thứ tự hay không.

public final void `setResult` (int code, `String` data, `Bundle` extras)

Được đưa vào `API cấp 1 (API level 1)`

Thay đổi tất cả các dữ liệu kết quả trả về từ việc broadcast này; chỉ làm việc với tín hiệu broadcast được gửi thông qua `Context.sendOrderedBroadcast`. Toàn bộ dữ liệu kết quả hiện tại được thay thế bằng giá trị nhận về từ phương thức này.

Phương thức này không làm việc với các tín hiệu broadcast không theo thứ tự, chẳng hạn như những tín hiệu được gửi tới từ `Context.sendBroadcast`.

Tham số

code

Mã kết quả mới. Thường sử dụng các hằng Activity `RESULT_CANCELED` và `RESULT_OK`, mặc dù ý nghĩa thực sự của giá trị này liên quan tới đối tượng broadcast.

data

Dữ liệu kết quả mới. Đây là một chuỗi bất kỳ giải thích cho đối tượng broadcast; có thể là null.

extras

Bản đồ ánh xạ dữ liệu phụ trợ mới. Đây là đối tượng Bundle lưu dữ liệu bất kỳ, liên quan đến đối tượng broadcast. Có thể được thiết lập là null. Dữ liệu này góp phần thay thế hoàn toàn bản đồ ánh xạ hiện tại (nếu có).

public final void `setResultCode` **(int code)**

Được đưa vào API cấp 1 (API level 1)

Thay đổi mã kết quả hiện tại của tín hiệu broadcast này; phương thức `setResultCode()` chỉ làm việc với các tín hiệu broadcast được gửi tới thông qua `Context.sendOrderedBroadcast`. Phương thức này thường sử dụng các hằng Activity `RESULT_CANCELED` và `RESULT_OK`, mặc dù ý nghĩa thực sự của giá trị này liên quan đến đối tượng broadcast.

Phương thức setResultCode() **không làm việc với các tín hiệu broadcast không theo thứ tự, chẳng hạn như những tín hiệu được gửi tới từ** `Context.sendBroadcast.`

Tham số

code Mã kết quả mới.

Tham khảo thêm

- `setResult(int, String, Bundle)`

public final void `setResultData` (String data)

Được đưa vào API cấp 1 (API level 1)

Thay đổi dữ liệu kết quả hiện tại của việc broadcast này; phương thức `setResultData()` chỉ làm việc với tín hiệu broadcast được gửi thông qua `Context.sendOrderedBroadcast`. Chuỗi bất kỳ giải thích cho đối tượng broadcast.

Phương thức này không làm việc với các tín hiệu broadcast không theo thứ tự, chẳng hạn như những tín hiệu được gửi tới từ `Context.sendBroadcast`.

Tham số

Data Dữ liệu kết quả mới; có thể là null.

Tham khảo thêm

- `setResult(int, String, Bundle)`

public final void `setResultExtras` (Bundle extras)

Thay đổi giá trị phụ trợ (giá trị extra)cho kết quả hiện tại của việc broadcast này; chỉ làm việc với các tín hiệu được gửi tới thông qua `Context.sendOrderedBroadcast`. Đây là đối tượng `Bundle` lưu dữ liệu bất kỳ, giải thích cho đối tượng broadcast. Có thể được thiết lập là null. Gọi phương thức này để hoàn thành việc thay thế bản đồ ánh xạ hiện tại (nếu có).

Phương thức `setResultExtras` **không làm việc với các tín hiệu broadcast không theo thứ tự, chẳng hạn như những tín hiệu được gửi tới từ** `Context.sendBroadcast.`

Tham số

Extras Bản đồ ánh xạ dữ liệu phụ trợ (extra) mới, có thể là null

Tham khảo thêm

- `setResult(int, String, Bundle)`

14. Mẹo bảo mật

Android có những tính năng bảo mật được xây dựng bên trong hệ điều hành này, giúp giảm thiểu đáng kể tần suất và ảnh hưởng của các vấn đề bảo mật (security) ứng dụng. Hệ thống này được thiết kế để bạn có thể xây dựng ứng dụng của mình một cách đặc thù với hệ điều hành mặc định và các quyền trên file, giúp tránh được những quyết định khó khăn về bảo mật.

Một số tính năng bảo mật cốt lõi giúp bạn xây dựng ứng dụng an toàn, bao gồm:

- Công nghệ Sandbox (môi trường ảo chạy ứng dụng) dành cho ứng dụng Android, giúp cách ly dữ liệu ứng dụng của bạn với việc thực thi mã từ các ứng dụng khác.
- Một framework phát triển ứng dụng có cài sẵn các chức năng bảo mật chung mạnh mẽ, chẳng hạn như kỹ thuật mã hóa (cryptography), quyền và IPC (*Inter-process communication* - Giao tiếp liên tiến trình) an toàn.
- Các công nghệ như ASLR, NX, ProPolice, safe_iop, OpenBSD dlmalloc, OpenBSD calloc và Linux mmap_min_addr góp phần giảm thiểu rủi ro liên quan đến lỗi quản lý bộ nhớ thông thường.
- Hệ thống file được mã hóa (encrypted file system) cho phép bảo vệ dữ liệu khỏi bị mất hoặc tránh bị thiết bị ăn cắp.
- Quyền cấp cho người dùng để giới hạn truy cập tới các tính năng của hệ thống và dữ liệu người dùng.
- Quyền do ứng dụng định nghĩa để quản lý dữ liệu ứng dụng trên nền tảng của từng ứng dụng.

Tuy nhiên, quan trọng là bạn phải làm quen với các kỹ thuật tốt nhất về bảo mật trên hệ thống Android được hướng dẫn trong tài liệu này. Các phương pháp này sẽ rèn cho bạn những thói quen viết mã chung giúp giảm thiểu khả năng vô tình để lộ những vấn đề liên quan đến bảo mật gây bất lợi cho người dùng ứng dụng.

Lưu trữ dữ liệu

Mối lo ngại lớn nhất về bảo mật đối với một ứng dụng Android là xem xét liệu dữ liệu bạn lưu trên thiết bị có thể được các ứng dụng khác truy cập hay không. Có ba cách cơ bản để lưu dữ liệu trên thiết bị:

Sử dụng bộ nhớ trong

Mặc định, các file bạn tạo ra trên bộ nhớ trong chỉ được ứng dụng của bạn truy cập. Tính năng bảo vệ này được Android thiết lập và được áp dụng cho hầu hết ứng dụng.

Thường thì bạn nên tránh sử dụng chế độ `MODE_WORLD_WRITEABLE` hoặc `MODE_WORLD_READABLE` cho các file IPC, bởi những quyền này không cung cấp khả năng giới hạn truy cập lên các ứng dụng cụ thể, cũng như không cung cấp bất cứ điều khiển nào để định dạng dữ liệu. Nếu muốn chia sẻ dữ liệu của bạn với các tiến trình xử lý ứng dụng khác, bạn có thể nghĩ đến việc dùng một `content provider`. Provider này sẽ cung cấp quyền đọc và ghi cho các ứng dụng khác và có thể sử dụng quyền động (dynamic permission) trong từng trường hợp cụ thể.

Để cung cấp thêm tính năng bảo vệ cho dữ liệu nhạy cảm, bạn có thể chọn mã hóa các file cục bộ (local file) bằng cách dùng một khóa (key) để ứng dụng không trực tiếp truy cập được. Ví dụ, khóa có thể được đặt trong một `KeyStore` và được bảo vệ bằng password (mật khẩu) của người dùng, password này không được lưu trữ trên thiết bị. Điều này không thể bảo vệ dữ liệu khỏi quyền quản trị (root) có thể theo dõi người dùng nhập password, nhưng cũng giúp bảo vệ trong trường hợp mất thiết bị và chưa mã hóa file hệ thống.

Sử dụng bộ nhớ ngoài

Các file được tạo trên bộ nhớ ngoài, chẳng hạn như thẻ SD, có thể được đọc và ghi tổng thể. Bộ nhớ ngoài có thể được người dùng di chuyển và cũng có thể được một ứng dụng bất kỳ chỉnh sửa; do đó, bạn không nên lưu những thông tin nhạy cảm vào bộ nhớ này.

Cũng như dữ liệu từ các nguồn không đáng tin khác, bạn nên thực hiện kiểm tra tính hợp lệ của dữ liệu đầu vào khi xử lý dữ liệu lưu trong bộ nhớ ngoài. Chúng tôi nhấn mạnh rằng, bạn không nên lưu trữ các file có thể thực thi hoặc file lớp vào bộ nhớ ngoài trước khi tiến hành nạp động (dynamic loading). Nếu ứng dụng của bạn thực hiện truy xuất file có thể thực thi từ bộ nhớ ngoài, các file này nên được đánh dấu và kiểm định mã hóa trước khi nạp động.

Sử dụng content provider

Content provider cung cấp cơ chế lưu trữ có cấu trúc có thể chỉ được giới hạn trong ứng dụng của bạn, hoặc được xuất ra để những ứng dụng khác truy cập. Nếu bạn không định cho phép ứng dụng khác truy cập vào content provider của mình, hãy đánh dấu provider này bằng thuộc tính `android:exported=false` trong file kê khai của ứng dụng. Ngược lại, bạn cần thiết lập thuộc tính `android:exported` là `"true"` để cho phép ứng dụng khác truy cập vào dữ liệu được lưu trữ.

Khi tạo một content provider được xuất ra cho những ứng dụng khác dùng, bạn có thể xác định một quyền kết hợp cho phép đọc và ghi, hoặc các quyền phân biệt để đọc và ghi bên trong file kê khai. Chúng tôi khuyến nghị rằng, bạn nên hạn chế số quyền được yêu cầu vừa đủ cho những quyền trên để hoàn thành tác vụ trước mắt. Nhớ rằng, thường thì việc thêm quyền sau khi giới thiệu chức năng mới sẽ dễ hơn so với việc bỏ bớt quyền đi và phá vỡ những người dùng đã có.

Nếu bạn đang dùng một content provider để chia sẻ dữ liệu chỉ trong các ứng dụng do bạn phát triển, lựa chọn phù hợp là sử dụng thuộc tính `android:protectionLevel` và thiết lập bảo vệ là `"signature"`. Quyền signature không yêu cầu xác nhận của người dùng, nên các quyền này cung cấp trải nghiệm người dùng tốt hơn và truy cập

có kiểm soát hơn cho dữ liệu content provider khi việc truy cập dữ liệu của các ứng dụng được ký với cùng một khóa.

Content provider cũng có thể cung cấp thêm truy cập chi tiết (granular access) bằng cách khai báo thuộc tính `android:grantUriPermissions` và sử dụng các cờ `FLAG_GRANT_READ_URI_PERMISSION` cũng như `FLAG_GRANT_WRITE_URI_PERMISSION` trong đối tượng Intent kích hoạt thành phần. Phạm vi của những quyền này có thể được giới hạn cụ thể hơn bởi phần tử `<grant-uri-permission>`.

Khi truy cập một content provider, bạn cần sử dụng các phương thức truy vấn tham số hóa (parameterized query method) như `query()`, `update()` và `delete()` để tránh sự can thiệp của các lỗ hổng SQL injection (là lỗ hổng an ninh ảnh hưởng trực tiếp đến cơ sở dữ liệu và tài nguyên của một hệ thống Website) tiềm tàng từ những nguồn không đáng tin. Lưu ý, việc sử dụng các phương thức truy vấn tham số hóa là chưa đủ nếu đối số `selection` được xây dựng dựa trên sự ghép nối với dữ liệu người dùng trước khi gửi tới phương thức này.

Đừng đánh giá sai việc bảo mật đối với quyền ghi. Lưu ý, quyền ghi cùng với câu lệnh SQL cho phép xác nhận một số dữ liệu bằng cách sử dụng sáng tạo mệnh đề WHERE và phân tích kết quả trả về. Ví dụ, kẻ tấn công (attacker) có thể sẽ dò được một số điện thoại cụ thể có mặt trong bản nhật ký ghi lại các cuộc gọi bằng cách chỉnh sửa một hàng chỉ khi số điện thoại này thực sự tồn tại. Nếu dữ liệu content provider có cấu trúc dễ phát hiện, quyền ghi có thể tương đương với việc cung cấp cả quyền đọc và ghi.

Sử dụng quyền

Do hệ thống sandbox Android bảo vệ ứng dụng khỏi các ứng dụng khác, nên các ứng dụng phải chia sẻ tài nguyên và dữ liệu một cách tường minh. Ứng dụng thực hiện việc này bằng cách khai báo quyền chúng cần để dùng cho những tính năng bổ sung không được sandbox cơ bản cung cấp, bao gồm quyền truy cập tới các tính năng của thiết bị như camera.

Yêu cầu quyền

Chúng tôi khuyến nghị nên tối thiểu hóa số quyền mà ứng dụng của bạn yêu cầu. Việc không truy cập vào quyền nhạy cảm giúp giảm thiểu rủi ro có thể vô tình lạm dụng những quyền này, tăng khả năng người dùng chấp nhận ứng dụng và cũng khiến ứng dụng ít bị tấn công hơn. Thông thường, nếu không cần quyền để thực hiện chức năng nào đó trong chương trình của mình, bạn đừng yêu cầu quyền đó.

Nếu có thể thiết kế ứng dụng của bạn theo cách không cần yêu cầu quyền thì tốt hơn. Ví dụ, thay vì yêu cầu truy cập tới thông tin thiết bị để tạo một định danh duy nhất, hãy tạo một GUID (Globally unique identifier - Định danh duy nhất toàn cầu) cho ứng dụng của bạn (xem mục "Xử lý dữ liệu người dùng" bên dưới). Hoặc, thay vì sử dụng bộ nhớ ngoài (phải yêu cầu quyền), hãy lưu trữ dữ liệu ở bộ nhớ trong.

Ngoài việc yêu cầu quyền, ứng dụng của bạn có thể sử dụng `<permissions>` để bảo vệ IPC rất cần bảo mật các dữ liệu nhạy cảm và sẽ được xuất ra cho những ứng dụng khác, chẳng hạn như một `ContentProvider`. Nhìn chung, chúng tôi khuyên bạn nên

sử dụng các điều khiển truy cập (access control) thay vì quyền cần xác nhận của người dùng (user confirmed permission) ở những vị trí khả thi, vì các quyền này có thể khiến người dùng bối rối. Ví dụ, hãy xét tới việc sử dụng mức bảo vệ xác nhận bằng chữ ký (signature protection level) cho những quyền của giao tiếp IPC giữa các ứng dụng của cùng một nhà phát triển chương trình.

Tránh để rò rỉ dữ liệu được quyền bảo vệ. Điều này có thể xảy ra khi ứng dụng của bạn xuất dữ liệu thông qua IPC duy nhất có sẵn vì IPC này có một quyền xác định, nhưng không yêu cầu quyền đó cho client nào trong giao diện IPC này.

Tạo quyền

Thông thường, bạn nên cố gắng định nghĩa ít quyền nhất có thể trong khi vẫn đáp ứng được yêu cầu về bảo mật. Việc tạo quyền mới thường không phổ biến đối với đa số ứng dụng, do các quyền được hệ thống định nghĩa thường đã bao phủ rất nhiều tình huống. Tại vị trí thích hợp, hãy thực hiện kiểm tra truy cập bằng cách sử dụng những quyền đã có.

Nếu phải tạo một quyền mới, hãy cân nhắc xem liệu bạn có thể hoàn thành tác vụ này với mức bảo vệ "signature" hay không. Người dùng sẽ không nhận thấy sự có mặt của quyền xác nhận bằng chữ ký và khi ứng dụng thực hiện kiểm tra quyền, chỉ cho phép các ứng dụng được xác nhận chữ ký là của cùng một nhà phát triển chương trình thì mới có thể truy cập được.

Nếu muốn tạo một quyền với mức bảo vệ "dangerous" ("nguy hiểm"), bạn phải tính đến một số vấn phức tạp sau:

- Quyền phải là một chuỗi diễn đạt ngắn gọn cho người dùng biết về quyết định bảo mật mà họ được yêu cầu thực hiện.
- Chuỗi quyền phải được bản địa hóa cho nhiều ngôn ngữ khác nhau.
- Người dùng có thể chọn không cài đặt ứng dụng nếu có quyền nào làm họ bối rối hoặc được cho là rủi ro.
- Các ứng dụng có thể yêu cầu quyền khi trình tạo quyền không được cài đặt.

Mỗi vấn đề nêu trên đều là một thách thức phi kỹ thuật điển hình với bạn trong tư cách một nhà phát triển chương trình, ngoài ra còn khiến người dùng bối rối. Đó là lý do tại sao chúng tôi không khuyến khích bạn dùng mức quyền "dangerous".

Sử dụng kết nối mạng

Các giao dịch mạng vốn tiềm ẩn nhiều rủi ro về bảo mật, vì chúng bao gồm việc truyền dữ liệu thường là thông tin riêng tư của người dùng. Mọi người ngày càng cảnh giác với các mối lo ngại về bảo mật trên thiết bị di động, đặc biệt là khi thiết bị thực hiện giao dịch mạng. Do đó, đây là yếu tố quan trọng khi ứng dụng của bạn thực hiện các biện pháp cài đặt tốt nhất, hướng tới bảo vệ dữ liệu người dùng trong mọi hoàn cảnh.

Sử dụng kết nối mạng IP

Kết nối mạng trên hệ điều hành Android không khác nhiều so với các môi trường Linux khác. Mối bận tâm chính là đảm bảo sử dụng giao thức (protocol) phù hợp với dữ liệu nhạy cảm, chẳng hạn như `HttpsURLConnection` đối với các lưu lượng truy cập Web an toàn (secure web traffic). Chúng tôi khuyên bạn nên sử dụng giao thức HTTPS thay vì HTTP ở mọi nơi mà HTTPS được server hỗ trợ, bởi vì thiết bị di động thường kết nối tới một mạng không an toàn, chẳng hạn như điểm truy cập Wi-Fi công cộng.

Có thể dễ dàng cài đặt giao tiếp mức cổng mã hóa, đã được xác thực bằng cách sử dụng lớp `SSLSocket`. Các thiết bị Android thường xuyên kết nối với những mạng không dây thiếu an toàn thông qua Wi-Fi, nên việc sử dụng kết nối mạng an toàn cho tất cả các ứng dụng kết nối qua mạng rất được khuyến khích.

Chúng tôi cũng nhận thấy rằng một số ứng dụng dùng cổng mạng (network port) localhost để xử lý các IPC nhạy cảm. Chúng tôi phản đối cách tiếp cận như vậy, vì các ứng dụng khác trên thiết bị có thể truy cập giao diện này. Thay vào đó, bạn nên sử dụng một cơ chế IPC của Android; nhờ cơ chế này, việc xác thực có thể thực hiện được, chẳng hạn như khi dùng một `Service`. (Thậm chí điều này còn không tốt bằng sử dụng vòng lặp ngược (loopback) để liên kết với `INADDR_ANY`, bởi vì sau đó ứng dụng của bạn có thể nhận yêu cầu từ mọi nơi).

Một vấn đề chung nhằm bảo vệ việc lặp là đảm bảo rằng bạn không tin vào những dữ liệu được tải về từ HTTP hoặc các giao thức không an toàn khác. Điều này bao gồm việc kiểm tra đầu vào trên `WebView` và bất cứ phản hồi intent nào được phát ra từ HTTP.

Sử dụng kết nối mạng điện thoại

Ban đầu, giao thức SMS được thiết kế cho giao tiếp từ người dùng-tới-người dùng và không phù hợp với ứng dụng truyền dữ liệu. Do hạn chế này của SMS, nên chúng tôi rất khuyến khích việc sử dụng Google Cloud Messaging (Gửi tin nhắn qua "đám mây" Google - GCM) và mạng IP để gửi tin nhắn dữ liệu từ một Web server tới ứng dụng của bạn trên thiết bị người dùng.

Hãy cảnh giác với việc SMS không được mã hóa hoặc được xác thực cẩn thận trên mạng cũng như thiết bị. Cụ thể, bất cứ đối tượng nhận SMS nào cũng nên đề phòng việc người dùng ác ý gửi tin nhắn SMS tới ứng dụng của bạn - Không phản hồi những dữ liệu SMS chưa được xác thực có thể thực hiện các lệnh nhạy cảm. Bạn cũng nên nhận thức được rằng SMS có thể là đối tượng giả mạo và/hoặc bị chặn trên mạng. Trên chính những thiết bị chạy trên nền Android, tin nhắn SMS có thể được truyền dưới dạng các broadcast intent; do đó, những tin nhắn này có thể bị các ứng dụng khác đọc hoặc lấy cắp nếu có quyền READ_SMS.

Thực hiện kiểm tra tính hợp lệ của dữ liệu đầu vào

Việc kiểm tra tính hợp lệ của dữ liệu đầu vào không đầy đủ là một trong những vấn đề bảo mật phổ biến nhất ảnh hưởng tới ứng dụng, bất kể các dữ liệu này đang chạy trên loại nền tảng (platform) nào. Android có những biện pháp xử lý mức nền tảng làm giảm tác động xấu của vấn đề liên quan đến việc kiểm tra tính hợp lệ của dữ liệu đầu vào lên

ứng dụng và bạn nên sử dụng các tính năng này ở bất cứ nơi nào có thể. Cũng cần lưu ý rằng, việc chọn các ngôn ngữ an toàn có thể làm giảm thiểu khả năng xảy ra những vấn đề về kiểm tra đầu vào.

Nếu bạn đang sử dụng mã máy (native code), bất cứ dữ liệu nào được đọc từ file, được nhận thông qua mạng hoặc từ IPC đều có thể tiềm ẩn nguy cơ gây ra các vấn đề về bảo mật. Vấn đề phổ biến nhất là tràn bộ đệm (buffer overflow), sử dụng sau khi đã giải phóng (use after free) và lỗi lôgíc off-by-one error (lỗi bỏ sót điều kiện biên). Android cung cấp một số kỹ thuật như ASLR và DEP, giúp giảm thiểu khả năng xảy ra các lỗi này, nhưng không giải quyết được tận gốc vấn đề. Bạn có thể tránh những sai sót này bằng cách xử lý con trỏ và quản lý bộ đệm một cách cẩn thận.

Các ngôn ngữ động dựa trên chuỗi, chẳng hạn như JavaScript và SQL, cũng là đối tượng cần kiểm tra đầu vào do thoát ra ký tự và cho phép chèn script (script injection).

Nếu bạn đang sử dụng dữ liệu bên trong truy vấn sẽ được gửi tới cơ sở dữ liệu SQL hoặc content provider, SQL Injection là một vấn đề. Cách bảo vệ tốt nhất là sử dụng truy vấn tham số hóa, như đã thảo luận ở mục trước về content provider. Giới hạn các quyền là chỉ đọc (only-read) hoặc chỉ ghi (only-write) cũng làm giảm khả năng gây hại liên quan đến SQL Injection.

Nếu không thể sử dụng những tính năng bảo mật trên, chúng tôi rất khuyến khích bạn dùng các định dạng cấu trúc dữ liệu tốt và kiểm tra xem dữ liệu có phù hợp với định dạng mong muốn hay không. Việc lập danh sách đen các ký tự hoặc ký tự thay thế không được sử dụng có thể là một chiến lược hiệu quả, song bạn nên tránh những kỹ thuật này vì chúng dễ phát sinh lỗi trong thực tế.

Xử lý dữ liệu người dùng

Nhìn chung, cách tiếp cận tốt nhất để bảo mật dữ liệu người dùng là hạn chế tối đa việc sử dụng API để truy cập dữ liệu nhạy cảm hoặc cá nhân của người dùng. Nếu có truy cập tới dữ liệu của người dùng, bạn có thể tránh lưu trữ hoặc truyền thông tin, không lưu trữ hay truyền dữ liệu. Cuối cùng, hãy cân nhắc xem liệu có cách nào để cài đặt bảng băm (hash) hoặc dạng dữ liệu không đảo chiều (non-reversible) cho ứng dụng của bạn. Ví dụ, ứng dụng của bạn có thể dùng bảng băm với một địa chỉ e-mail làm khóa chính, nhằm tránh truyền hoặc lưu trữ địa chỉ e-mail. Việc này làm giảm khả năng có thể vô tình để lộ dữ liệu, đồng thời làm giảm cơ hội các đối tượng tấn công cố gắng lợi dụng ứng dụng của bạn.

Nếu ứng dụng của bạn truy cập thông tin cá nhân, chẳng hạn như password (mật khẩu) hoặc username (tên người dùng), cần nhớ rằng một số quyền tài phán (jurisdiction) có thể yêu cầu bạn cung cấp chính sách riêng giải thích việc sử dụng và lưu trữ dữ liệu đó. Do vậy, tuân theo những kỹ thuật bảo mật tốt nhất theo cách hạn chế tối đa việc truy cập vào dữ liệu người dùng cũng có thể giúp đơn giản hóa yêu cầu người dùng.

Bạn cũng nên cân nhắc xem liệu ứng dụng của bạn có vô tình để lộ thông tin cá nhân cho bên khác không, chẳng hạn như các thành phần thuộc bên thứ ba dành để quảng cáo hoặc những service bên thứ ba được ứng dụng của bạn dùng. Nếu không chắc chắn lý do tại sao một thành phần hoặc service cần thông tin cá nhân, đừng cung cấp những thông tin này. Nói chung, giảm việc ứng dụng của bạn truy cập vào thông tin cá nhân sẽ giúp giảm thiểu khả năng xảy ra vấn đề trong lĩnh vực này.

Nếu buộc phải truy cập dữ liệu nhạy cảm, hãy ước tính xem liệu các thông tin này có cần truyền tới server không, hoặc liệu thao tác đó có thể thực hiện trên client hay không. Bạn cũng cần cân nhắc về việc chạy bất cứ mã nào sử dụng dữ liệu nhạy cảm trên client để tránh truyền đi dữ liệu người dùng.

Ngoài ra, cũng cần đảm bảo rằng bạn không vô tình để lộ dữ liệu người dùng cho ứng dụng khác trên thiết bị khi quá dễ dãi với IPC, các file có thể ghi được (world writable file) hoặc cổng mạng. Đây là một trường hợp đặc biệt làm rò rỉ dữ liệu đã được bảo vệ bởi quyền, như đã thảo luận ở mục "Yêu cầu quyền".

Nếu bạn cần một GUID, hãy tạo một số lớn, duy nhất và lưu trữ nó. Không sử dụng định danh điện thoại, chẳng hạn như số điện thoại hoặc IMEI (International Mobile Equipment Identity - Số nhận dạng thiết bị di động trên toàn thế giới) có thể liên hệ với thông tin cá nhân. Chủ đề này sẽ được thảo luận chi tiết tại "Android Developer Blog" ("Blog dành cho nhà phát triển Android").

Hãy cẩn thận khi ghi nhật ký (log) trên thiết bị. Trong Android, nhật ký là tài nguyên được chia sẻ, có thể được một ứng dụng có quyền `READ_LOGS` dễ dàng truy cập. Mặc dù dữ liệu nhật ký điện thoại là tạm thời và có thể xóa được để ghi lại, nhưng nhật ký không thích hợp chứa thông tin người dùng có thể vô tình làm rò rỉ dữ liệu người dùng cho các ứng dụng khác.

Sử dụng WebView

Do `WebView` sử dụng nội dung Web có thể bao gồm HTML và JavaScript, nên nếu sử dụng không đúng có thể gặp phải những vấn đề bảo mật thông thường, chẳng hạn như cross-site-scripting (còn gọi là XSS hoặc CSS, thường được cho là một trong những lớp ứng dụng phổ biến nhất các kỹ thuật hacking. Nói chung, cross-site scripting cho thấy rằng kỹ thuật hacking đó thúc đẩy những lỗ hổng trong mã của một ứng dụng Web để cho phép một kẻ tấn công gửi nội dung độc hại từ một người dùng cuối và thu thập một số loại dữ liệu từ các nạn nhân). Android chứa một số cơ chế giúp giảm thiểu phạm vi ảnh hưởng của những vấn đề này bằng cách giới hạn khả năng của `WebView` cho một số chức năng nhỏ của ứng dụng.

Nếu ứng dụng của bạn không trực tiếp sử dụng JavaScript bên trong `WebView`, *đừng* gọi phương thức `setJavaScriptEnable()`. Một số mã mẫu sử dụng phương thức này, và có thể bạn sẽ dùng lại với cùng mục đích trong ứng dụng sản phẩm. Do đó, hãy loại bỏ lời gọi phương thức này nếu không cần thiết. Mặc định, `WebView` không thực hiện JavaScript, nên không thể xảy ra cross-site-scripting.

Sử dụng phương thức `addJavaScriptInterface()` trong các xử lý cụ thể, vì phương thức này cho phép mã JavaScript thực hiện thao tác thường chỉ dành cho ứng dụng Android. Nếu sử dụng phương thức này, chỉ cho phép tác động trên những trang Web mà tất cả dữ liệu đầu vào đều đáng tin. Nếu cho phép đầu vào không đáng tin, mã JavaScript độc có thể xử lý các phương thức Android trong ứng dụng của bạn. Nói chung, chúng tôi khuyến khích chỉ sử dụng phương thức `addJavaScriptInterface()` cho mã JavaScript bên trong APK ứng dụng của bạn.

Nếu ứng dụng truy cập dữ liệu nhạy cảm bằng một `WebView`, có thể bạn muốn dùng phương thức `clearCache()` để xóa những file được lưu trữ cục bộ. Phần header

phía server kiểu như `no-cache` (không lưu cache) có thể được dùng để quy định rằng một ứng dụng không nên lưu cache nội dung cụ thể.

Xử lý chứng thực

Nhìn chung, chúng tôi khuyến nghị bạn nên hạn chế tối đa việc yêu cầu chứng thực người dùng (user credential) - để dễ nhận ra các vụ tấn công giả mạo (phising attack), và những vụ này cũng ít có khả năng thành công. Thay vào đó, hãy sử dụng token xác thực (authorization token) và làm mới token này.

Nếu có thể, bạn không nên lưu trữ username (tên người dùng) và password (mật khẩu) trên thiết bị. Thay vào đó, hãy thực hiện xác thực khởi tạo bằng cách sử dụng username và password được người dùng cung cấp, sau đó sử dụng token xác thực đối với service cụ thể trong thời gian ngắn.

Các service mà nhiều ứng dụng nên được truy cập sử dụng `AccountManager`. Nếu có thể, hãy dùng lớp `AccountManager` để xử lý những service dựa trên "đám mây" và không lưu trữ password trên thiết bị.

Sau khi dùng `AccountManager` để truy xuất một `Account`, `CREATOR` trước khi truyền đi trong chứng thực bất kỳ, bạn sẽ không bị rơi vào tình trạng vô tình truyền sai chứng thực tới ứng dụng.

Nếu chứng thực chỉ được dùng trong những ứng dụng do bạn phát triển, bạn có thể kiểm tra ứng dụng truy cập vào `AccountManager` bằng cách sử dụng phương thức `checkSignature()`. Mặt khác, nếu chỉ có một ứng dụng dùng chứng thực này, bạn có thể sử dụng một `KeyStore` (đối tượng lưu khóa) để lưu trữ.

Sử dụng kỹ thuật mã hóa

Ngoài việc cung cấp cơ chế cách ly dữ liệu, hỗ trợ mã hóa toàn bộ file hệ thống và cung cấp kênh giao tiếp an toàn, Android còn cung cấp rất nhiều thuật toán bảo vệ dữ liệu nhờ sử dụng kỹ thuật mã hóa (cryptography).

Nhìn chung, hãy cố gắng sử dụng mức thực thi cao nhất của framework đã có từ trước để hỗ trợ trường hợp sử dụng của bạn. Nếu cần truy xuất an toàn một file từ nơi đã biết, một URI HTTPS đơn giản có thể phù hợp và không cần đến kỹ thuật mã hóa. Nếu cần một ống bảo mật (secure tunnel), hãy xem xét sử dụng `HttpsURLConnection` hoặc `SSLSocket` thay vì viết giao thức của chính bạn.

Nếu bạn đang cần cài đặt giao thức riêng, chúng tôi rất khuyến nghị rằng bạn *không* nên cài đặt thuật toán mã hóa của riêng mình. Hãy sử dụng các thuật toán mã hóa (cryptographic algorithm) đã có như thuật toán mã hóa AES hoặc RSA được cung cấp trong lớp `Cipher`.

Sử dụng cơ chế sinh ngẫu nhiên số bảo mật (secure random number generator), `SecureRandom`, để khởi tạo khóa mã hóa bất kỳ, `KeyGenerator`. Sử dụng một khóa không được tạo từ cơ chế sinh ngẫu nhiên số bảo mật thường làm giảm đáng kể khả năng bảo mật của thuật toán, tạo điều kiện cho tấn công offline (ngoại tuyến).

Nếu bạn cần lưu trữ một khóa để tái sử dụng, hãy dùng cơ chế giống như `KeyStore` cung cấp một cơ chế lưu trữ lâu dài và truy xuất khóa mã hóa (cryptographic key).

Sử dụng giao tiếp liên tiến trình

Một số ứng dụng cố gắng cài đặt IPC bằng cách sử dụng các kỹ thuật Linux truyền thống như cổng mạng và file chia sẻ. Thay vào đó, chúng tôi rất khuyến khích bạn sử dụng tính năng của hệ thống Android như `Intent`, `Binder` hoặc `Messenger` với một `Service` và `BroadcastReceiver`. Cơ chế giao tiếp liên tiến trình của Android cho phép bạn xác nhận danh tính của việc kết nối ứng dụng tới IPC của bạn và thiết lập chính sách bảo mật cho từng cơ chế IPC.

Rất nhiều phần tử bảo mật được chia sẻ thông qua cơ chế IPC. Nếu cơ chế IPC của bạn không xác định để ứng dụng khác dùng, hãy thiết lập thuộc tính `android:exported` là "false" trong phần tử tương ứng của file kê khai, chẳng hạn như phần tử `<service>`. Việc này rất hữu dụng đối với những ứng dụng bao gồm nhiều tiến trình trong cùng UID (Unique identification - Định danh duy nhất), hoặc trong trường hợp bạn ra quyết định muộn khi phát triển ứng dụng là thực sự không muốn xuất ra bên ngoài chức năng như IPC nhưng lại không muốn viết lại mã.

Nếu IPC được xác định là cho phép ứng dụng khác truy cập, bạn có thể áp dụng một chính sách bảo mật bằng cách sử dụng phần tử `<permission>`. Nếu IPC thực hiện việc giao tiếp giữa từng ứng dụng khác nhau của bạn được ký với cùng chữ ký, tốt hơn hết bạn nên sử dụng quyền mức `"signature"` trong thành phần `android:protectionLevel`.

Sử dụng intent

Intent là cơ chế được ưa dùng cho IPC không đồng bộ trong Android. Tùy thuộc vào yêu cầu của ứng dụng, bạn có thể sử dụng `sendBroadcast()`, `sendOrderedBroadcast()`, hoặc một intent tường minh cho một thành phần ứng dụng xác định.

Lưu ý, việc broadcast theo thứ tự có thể bị một đối tượng nhận "tiêu thụ", nên có thể chúng không được gửi tới tất cả ứng dụng. Nếu đang gửi một intent cần chuyển tới một trình nhận xác định, bạn phải sử dụng một intent tường minh khai báo trình nhận bằng intent tên.

Trình gửi intent có thể kiểm định trình nhận có một quyền xác định quyền khác null gửi cùng lời gọi phương thức. Chỉ những ứng dụng có quyền này mới được nhận intent. Nếu dữ liệu bên trong intent broadcast có tính chất nhạy cảm, bạn nên cân nhắc áp dụng quyền để đảm bảo rằng những ứng dụng xấu không thể đăng ký nhận các thông điệp này nếu thiếu quyền thích hợp. Trong những hoàn cảnh này, bạn cũng có thể cân nhắc tới việc trực tiếp chỉ ra trình nhận thay vì gửi broadcast.

> **Ghi chú:** Không nên cân nhắc bộ lọc intent cho tính năng bảo mật - các thành phần có thể được dẫn ra bằng những intent tường minh và có thể không chứa dữ liệu phù hợp với bộ lọc intent. Bạn nên thực hiện xác thực đầu vào bên trong trình nhận intent để chắc chắn dữ liệu này được định dạng phù hợp với trình nhận, service hoặc activity được dẫn ra.

Sử dụng service

`Service` thường được dùng để cung cấp tính năng sử dụng cho ứng dụng khác. Mỗi lớp service phải có khai báo tương ứng trong file kê khai của lớp này.

Mặc định, service không được xuất ra và các ứng dụng khác cũng không thể gọi đến. Tuy nhiên, nếu bạn thêm bộ lọc intent bất kỳ để khai báo service, thì mặc định service này được xuất ra. Tốt nhất, bạn nên khai báo rõ thuộc tính `android:exported` để đảm bảo service sẽ hoạt động như cách bạn muốn. Chúng ta cũng có thể bảo vệ các service bằng cách sử dụng thuộc tính `android:permission`. Khi làm vậy, các ứng dụng khác sẽ phải khai báo phần tử `<uses-permission>` tương ứng trong file kê khai của chính những ứng dụng đó để có thể khởi động, dừng hoặc liên kết với service.

Một service có thể bảo vệ các lời gọi IPC cá nhân trong service bằng quyền, nhờ gọi phương thức `checkCallingPermission()` trước khi thực thi cài đặt lời gọi IPC này. Thông thường, chúng tôi khuyến nghị bạn nên sử dụng các quyền khai báo trong file kê khai, như vậy những quyền này sẽ ít bị xâm phạm hơn.

Sử dụng đối tượng Binder và Message

Sử dụng `Binder` và `Messenger` là cơ chế được ưa dùng cho IPC kiểu RPC (*Remote procedure call - Thủ tục gọi hàm từ xa)*. Hai đối tượng này cung cấp một giao diện được định nghĩa tốt, cho phép xác thực qua lại giữa các điểm endpoint (điểm giao tiếp cuối), nếu cần.

Chúng tôi rất khuyến khích việc thiết kế giao diện theo cách không yêu cầu kiểm tra quyền cụ thể của giao diện. Các đối tượng `Binder` và `Messenger` không được khai báo bên trong file kê khai của ứng dụng; do đó, bạn không thể áp dụng quyền khai báo trực tiếp lên những đối tượng này. Thông thường, hai đối tượng này thừa kế các quyền được khai báo trong file kê khai của ứng dụng cho `Service` và `Activity` chứa chúng. Nếu bạn đang tạo giao diện yêu cầu xác thực và/hoặc điều khiển truy cập, các điều khiển này phải được thêm vào một cách rõ ràng dưới dạng mã trong giao diện `Binder` hoặc `Messenger`.

Nếu cung cấp một giao diện yêu cầu điều khiển truy cập, bạn có thể sử dụng phương thức `checkCallingPermission()` để kiểm tra xem liệu lời gọi có kèm theo quyền hay không. Việc này đặc biệt quan trọng trước khi truy cập một service đại diện cho lời gọi này, nếu định danh của ứng dụng do bạn phát triển được chuyển tới các giao diện khác. Nếu gọi ra một giao diện do `Service` cung cấp, việc dẫn `bindService()` ra có thể gặp lỗi nếu bạn không có quyền truy cập tới service đã xác định đó. Nếu gọi một giao diện được chính ứng dụng của bạn cung cấp cục bộ, thì việc sử dụng phương thức `clearCallingIdentify()` sẽ có ích khi cần vượt qua quá trình kiểm tra bảo mật bên trong.

Để tìm hiểu thêm về việc thực hiện IPC với service, xem mục "Bound Services" ("Service có liên kết").

Sử dụng broadcast receiver

Mỗi đối tượng `BroadcastReceiver` xử lý các yêu cầu không đồng bộ do một `Intent` khởi tạo.

Mặc định, trình nhận được xuất ra bên ngoài và bất cứ ứng dụng nào khác đều có thể gọi trình này. Nếu đối tượng `BroadcastReceiver` của bạn được thiết kế để các ứng dụng khác sử dụng, có thể bạn muốn áp dụng quyền bảo mật cho trình nhận bằng cách dùng phần tử `<receiver>` trong file kê khai của ứng dụng. Việc này sẽ ngăn những ứng dụng chưa có quyền thích hợp không gửi được intent tới `BroadcastReceiver` này.

Nạp mã động

Chúng tôi khuyến cáo bạn không được nạp mã từ bên ngoài APK ứng dụng. Việc làm này sẽ gia tăng đáng kể khả năng ứng dụng thỏa hiệp với việc chèn mã (code injection) hoặc can thiệp mã (code tampering). Làm như vậy cũng khiến việc quản lý phiên bản và kiểm thử ứng dụng trở nên phức tạp hơn. Cuối cùng, việc nạp mã từ bên ngoài APK ứng dụng khiến chúng ta không thể kiểm tra hành vi của ứng dụng; do đó, hoạt động này có thể bị cấm trong một số môi trường.

Nếu ứng dụng của bạn nạp mã động, điều quan trọng nhất nên nhớ là mã được nạp động chạy cùng quyền bảo mật với APK của ứng dụng. Người dùng quyết định cài đặt ứng dụng dựa trên danh tính của bạn, và họ mong muốn được cung cấp bất cứ mã nào chạy bên trong ứng dụng, bao gồm cả mã nạp động.

Rủi ro bảo mật chủ yếu liên quan tới mã nạp động là mã này cần được lấy từ một nguồn có thể xác minh được. Nếu các mô đun này được thêm trực tiếp vào APK của bạn, thì ứng dụng khác không thể chỉnh sửa được. Điều này luôn đúng, bất kể mã của của bạn là thư viện mã máy (native library) hay một lớp được nạp bằng `DexClassLoader`. Chúng ta từng thấy rất nhiều ứng dụng cố gắng nạp mã từ những địa điểm không an toàn, như tải về từ mạng thông qua giao thức không mã hóa, hoặc từ địa điểm có thể ghi được như bộ nhớ ngoài. Những nơi địa điểm có thể cho phép đối tượng tấn công trên mạng chỉnh sửa nội dung trong khi truyền, hoặc cho phép ứng dụng khác trong thiết bị người dùng chỉnh sửa nội dung.

Bảo mật trên máy ảo

Dalvik là máy ảo (VM - Virtual Machine) chạy trên hệ thống Android. Dalvik được xây dựng dành riêng cho Android, nhưng nhiều mối quan ngại liên quan đến mã bảo mật trong các máy ảo khác cũng tồn tại ở Android. Nói chung, bạn không cần bận tâm đến vấn đề bảo mật liên quan đến máy ảo. Ứng dụng của bạn chạy trong môi trường sandbox an toàn, nên các tiến trình khác trong hệ thống không thể truy cập vào mã hoặc dữ liệu riêng của bạn.

Nếu bạn quan tâm tới chủ đề bảo mật trên máy ảo, chúng tôi khuyến nghị bạn nên làm quen dần với các tài liệu hiện có về chủ đề này. Hai nguồn tài liệu phổ biến hơn là:

- http://www.securingjava.com/toc.html.
- https://www.owasp.org/index.php/Java_Security_Resources.

Tài liệu này chỉ tập trung vào các khía cạnh liên quan đến Android hoặc những điểm khác môi trường máy ảo khác. Đối với các nhà phát triển chương trình đã có kinh nghiệm lập trình máy ảo trong môi trường khác, hai vấn đề lớn dưới đây có thể khác với khi viết ứng dụng cho Android:

- Một số máy ảo, chẳng hạn như JVM hoặc .net runtime, hoạt động giống như một biên bảo mật (security boundary), cách ly mã khỏi các tính năng của hệ điều hành bên dưới. Trong Android, máy ảo Dalvik không phải là một biên bảo mật - sandbox của ứng dụng được cài tại mức hệ điều hành, nên Dalvik có thể hoạt động tương thích với mã máy trong cùng ứng dụng mà không có bất cứ ràng buộc nào về bảo mật.
- Nhận một không gian lưu trữ giới hạn trên thiết bị di động, việc này vốn quen thuộc với những nhà phát triển muốn xây dựng ứng dụng kiểu mô đun và sử dụng việc nạp lớp động. Khi làm vậy, hãy xét tới cả nguồn bạn truy xuất lôgíc ứng dụng và vị trí lưu trữ ứng dụng cục bộ. Không nên nạp lớp động từ nguồn chưa được kiểm định, chẳng hạn như những nguồn mạng không được bảo mật hoặc bộ nhớ ngoài, vì mã có thể bị chỉnh sửa để chứa các hành vi độc hại.

Bảo mật trong mã máy

Nhìn chung, chúng tôi khuyến khích các lập trình viên dùng Android SDK (S*oftware development kit - Bộ phát triển phần mềm)* để phát triển ứng dụng thay vì dùng mã máy đi kèm Android NDK (Native Development Kit - Bộ phát triển mã máy). Các ứng dụng được xây dựng với mã máy phức tạp hơn, ít linh động hơn và thường bao gồm nhiều lỗi bộ nhớ hơn, chẳng hạn như tràn bộ đệm.

Android được xây dựng dựa trên nhân (kernel) Linux; bởi vậy, nắm rõ việc phát triển bảo mật trên Linux sẽ đặc biệt có ích trong trường hợp bạn muốn sử dụng mã máy. Hướng dẫn về bảo mật trên Linux là nội dung nằm ngoài phạm vi của tài liệu này, nhưng đã được bàn luận trong một tài liệu vào loại phổ biến nhất là “Programming for Linux and Unix HOWTO”, tại địa chỉ http://www.dwheeler.com/secure-programs.

Một khác biệt quan trọng giữa Android và môi trường Linux là Sandbox của ứng dụng. Trên Android, tất cả các ứng dụng đều chạy trong Sandbox của ứng dụng, bao gồm cả các ứng dụng được viết bằng mã máy. Ở mức cơ bản nhất, một cách hay mà các lập trình viên đã quen với Linux nghĩ về Sandbox của Andorid là mọi ứng dụng đều nhận một UID duy nhất với các quyền được giới hạn. Điều này đã được thảo luận chi tiết hơn trong mục “Android Security Overview” (“Tổng quan về bảo mật trong Android”). Ngoài ra, bạn nên làm quen với quyền của ứng dụng ngay cả khi đang sử dụng mã máy.

15. WebView

15.1 Tổng quan về lớp WebView

WebView là một View dùng để hiển thị trang Web. Đây là lớp cơ bản mà bạn có thể dựa vào đó để cuộn trình duyệt Web, hoặc chỉ đơn giản là hiển thị một số nội dung trực tuyến bên trong Activity của bạn. Lớp này sử dụng WebKit rendering engine để hiển thị trang Web và bao gồm các phương thức để điều hướng tiến và lùi thông qua một history (lịch sử), phóng to và thu nhỏ, thực hiện tìm kiếm văn bản,…

Lưu ý, để Activity của bạn truy cập Internet và nạp trang Web trong một WebView, bạn phải thêm quyền `Internet` vào file kê khai của Android:

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

Khai báo quyền trên phải là con của phần tử `INTERNET`.

Để tìm hiểu thêm thông tin, mời bạn tham khảo mục "Xây dựng ứng dụng Web thông qua WebView" bên dưới.

Các trường hợp sử dụng cơ bản

Mặc định, một WebView không cung cấp widget giống như trình duyệt, không cho phép mã JavaScript và bỏ qua lỗi Web. Nếu mục đích chỉ là hiển thị một số HTML là một phần trong giao diện người dùng của bạn thì sử dụng WebView có vẻ ổn; người dùng không cần tương tác với trang Web, ngoại trừ việc đọc nó và trang Web cũng không cần tương tác với người dùng. Nếu thực sự muốn một trình duyệt Web đang phát triển mạnh, có thể bạn sẽ muốn gọi tới ứng dụng Browser với một intent URL hơn là hiển thị trang Web bằng WebView. Ví dụ:

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

Xem chủ đề về `Intent` để tìm hiểu thêm thông tin.

Để cung cấp một WebView trong Activity của bạn, hãy bao gồm một Activity trong layout Web của bạn, hoặc thiết lập toàn bộ cửa sổ Activity làm một WebView trong phương thức `onCreate()`.

```
WebView webview = new WebView(this);
setContentView(webview);
```

Sau đó nạp trang Web mong muốn:

```
// Cách sử dụng đơn giản nhất: Lưu ý, nếu có lỗi khi nạp trang này
// thì sẽ KHÔNG có ngoại lệ nào được ném  ra (xem bên dưới).
  webview.loadUrl("http://slashdot.org/");

 // HOẶC, bạn cũng có thể nạp từ một chuỗi HTML:
 String summary = "<html><body>You scored <b>192</b>points.</body></html>";
 webview.loadData(summary, "text/html", null);
 // ... mặc dù lưu ý rằng tồn tại một số hạn chế về những gì HTML
 // này có thể thực hiện.
 // Xem JavaDocs về loadData() và loadDataWithBaseURL() để tìm hiểu thêm.
```

Một WebView có vài điểm tùy chỉnh và tại đây, bạn có thể thêm hành vi của ứng dụng. Đó là:

- Tạo và thiết lập một lớp con `WebChromeClient`. Lớp này được gọi khi xảy ra một số sự kiện có thể gây ảnh hưởng đến giao diện người dùng trình duyệt, ví dụ như cập nhật tiến trình và cảnh báo JavaScript được gửi tới WebView. Xem mục "Debugging Tasks" ("Gỡ lỗi tác vụ") để biết thêm thông tin.
- Tạo và thiết lập một lớp con `WebViewClient`. Lớp con này có thể được gọi khi xảy ra một số sự kiện ảnh hưởng tới việc hiển thị nội dung, ví dụ như các lỗi hoặc tiến trình gửi form. Bạn cũng có thể chặn việc nạp URL ở đây (thông qua phương thức `shouldOverrideUrlLoading()`).
- Chỉnh sửa `WebSettings`, chẳng hạn như cho phép chạy mã JavaScript bằng phương thức `setJavaScriptEnabled()`.
- Chèn đối tượng Java vào WebView bằng cách sử dụng `addJavascriptInterface(Object, String)`. Phương thức này cho phép bạn chèn đối tượng Java vào ngữ cảnh JavaScript của trang; do đó, các đối tượng này có thể được mã JavaScript trên trang truy cập.

Sau đây là một ví dụ phức tạp hơn, minh họa quá trình xử lý lỗi, cài đặt và thông báo về tiến trình:

```
// Cho phép hiển thị tiến trình trong thanh tiêu đề của activity,
// giống như cách ứng dụng trình duyệt làm.
 getWindow().requestFeature(Window.FEATURE_PROGRESS);

 webview.getSettings().setJavaScriptEnabled(true);

 final Activity activity = this;
 webview.setWebChromeClient(new WebChromeClient() {
   public void onProgressChanged(WebView view, int progress) {
     // Các activity và WebView ước lượng tiến trình bằng những tiêu
// chuẩn khác nhau. Thước đo tiến trình (progress meter) sẽ tự
     // động biến mất khi chúng ta đạt tới 100%
     activity.setProgress(progress * 1000);
```

```
    }
  });
  webview.setWebViewClient(new WebViewClient() {
    public void onReceivedError(WebView view, int errorCode, String
description, String failingUrl) {
      Toast.makeText(activity, "Oh no! " + description,
                                          Toast.LENGTH_SHORT).show();
    }
  });

  webview.loadUrl("http://developer.android.com/");
```

Thu phóng WebView

Để cho phép chức năng thu phóng (phóng to, thu nhỏ) có sẵn, hãy thiết lập `WebSettings.setBuiltInZoomControls(boolean)` (được giới thiệu trong API cấp `CUPCAKE`).

Ghi chú: Sử dụng chức năng thu phóng khi chiều cao hoặc độ rộng được thiết lập là `WRAP_CONTENT` có thể dẫn đến hành vi không xác định và nên tránh.

Quản lý cookie và cửa sổ

Vì lý do bảo mật đã được giải thích rõ ở phần trước, nên ứng dụng của bạn có cache riêng, lưu trữ cookie,... - cache này không chia sẻ với dữ liệu ứng dụng trình duyệt.

Mặc định, yêu cầu mở cửa sổ mới của HTML bị bỏ qua. Điều này đúng trong cả trường hợp cửa sổ được mở bằng mã JavaScript lẫn bằng thuộc tính đích trên liên kết (link). Bạn có thể tùy chỉnh phương thức `WebChromeClient` để cung cấp hành vi của chính mình nhằm mở nhiều cửa sổ, đồng thời hiển thị chúng theo cách bạn muốn.

Hành vi chuẩn của Activity bị hủy và được tạo lại khi hướng của thiết bị thay đổi, hoặc khi có bất cứ cấu hình nào khác thay đổi. Điều này cũng làm cho WebView nạp lại trang hiện tại. Nếu không thích cách xử lý trên, bạn có thể thiết lập Activity của mình để xử lý thay đổi trong `orientation` và `keyboardHidden`, sau đó chỉ để lại một mình WebView. Khi đó, WebView sẽ tự động định hướng lại cho phù hợp. Đọc mục "Handling Runtime Changes" ("Xử lý các thay đổi trong thời gian chạy") để tìm hiểu thêm về cách xử lý việc thay đổi cấu hình trong thời gian chạy.

Xây dựng trang Web để hỗ trợ các mật độ màn hình khác nhau

Mật độ màn hình (screen density) của một thiết bị dựa trên độ phân giải màn hình (screen resolution). Một màn hình với mật độ thấp sẽ có số pixel trên mỗi inch (1 inch = 2,54 cm) thấp hơn; trong khi đó, một màn hình có mật độ lớn hơn đôi khi có số lượng pixel trên mỗi inch nhiều hơn đáng kể. Mật độ của màn hình là một thông số quan trọng bởi vì, nếu các thông số khác đều như nhau, một thành phần giao diện người dùng (chẳng hạn như một button) có chiều cao và độ rộng được định nghĩa theo số pixel màn hình sẽ xuất hiện lớn hơn trên màn hình mật độ thấp, đồng thời nhỏ hơn trên màn hình

mật độ cao hơn. Để đơn giản, Android chia tất cả các mật độ màn hình thực thành ba mật độ chung: Cao, trung bình và thấp.

Mặc định, WebView sẽ căn tỷ lệ một trang Web để trang này được vẽ ở kích thước phù hợp với việc hiển thị mặc định trên màn hình mật độ trung bình. Do đó, Android áp dụng tỷ lệ 1,5x trên màn hình mật độ cao (do số pixel của màn hình này nhỏ hơn) và tỷ lệ 0,75 trên màn hình mật độ thấp (do số pixel lớn hơn). Bắt đầu với API cấp ECLAIR, WebView hỗ trợ DOM, CSS và các tính năng của thẻ meta để giúp bạn (trong vai trò một nhà lập trình Web) xử lý màn hình với những mật độ khác nhau.

Sau đây là phần tổng kết những tính năng bạn có thể dùng để xử lý các mật độ màn hình khác nhau:

- Thuộc tính DOM `window.devicePixelRatio`. Giá trị của thuộc tính này xác định yếu tố tỷ lệ mặc định dùng cho thiết bị hiện tại. Ví dụ, nếu giá trị `window.devicePixelRatio` là "1,0", thiết bị được xem là có mật độ trung bình (medium density - mdpi) và việc điều chỉnh kích thước mặc định sẽ không được áp dụng cho trang Web này; nếu giá trị này là "1,5", thiết bị này được xem là thiết bị mật độ cao (high density device - hdpi) và nội dung trang Web sẽ được kéo với tỷ lệ 1,5x; nếu giá trị này là "0,75", thiết bị được xem là thiết bị có mật độ thấp (low density device - ldpi) và nội dung được kéo với tỷ lệ 0,75x.
- Truy vấn media CSS `-webkit-device-pixel-ratio`. Sử dụng thuộc tính này để xác định mật độ màn hình mà stylesheet này thường dùng. Giá trị tương ứng có thể là "0,75", "1" hoặc "1,5", cho biết kiểu này dùng cho thiết bị có màn hình mật độ thấp, trung bình hoặc cao tương ứng. Ví dụ:

```
<link rel="stylesheet" media="screen and
(-webkit-device-pixel-ratio:1.5)" href="hdpi.css" />
```

  Stylesheet `hdpi.css` chỉ được sử dụng cho thiết bị có tỷ lệ pixel màn hình mật độ cao là 1,5.

Hỗ trợ video trong HTML5

Để hỗ trợ video HTML5 nội tuyến trong ứng dụng, bạn cần bật chế độ tăng tốc phần cứng (hardware acceleration), đồng thời cài đặt một `WebChromeClient`. Để hỗ trợ toàn màn hình, bạn cần cài đặt `onShowCustomView(View, WebChromeClient.CustomViewCallback)` và `onHideCustomView()`, còn phương thức `getVideoLoadingProgressView()` thì không bắt buộc.

15.2 Xây dựng ứng dụng Web trong WebView

Nếu muốn cung cấp một ứng dụng Web (hoặc chỉ một trang Web) là một phần của ứng dụng client, bạn có thể dùng `WebView`. Lớp `WebView` là lớp mở rộng từ lớp `View` cho phép hiển thị trang Web thành một phần của layout activity. WebView *không* bao gồm bất kỳ tính năng nào của một trình duyệt Web đầy đủ, chẳng hạn như điều khiển điều hướng hoặc thanh địa chỉ (address bar). Mặc định, tất cả những gì `WebView` thực hiện, đều là hiển thị trang Web.

Một kịch bản phổ biến cho thấy việc sử dụng WebView rất có ích là khi bạn muốn cung cấp thông tin trong ứng dụng có thể cần cập nhật, chẳng hạn như một thỏa thuận với người dùng cuối hoặc hướng dẫn người dùng. Bên trong ứng dụng Android, bạn có thể tạo một Activity chứa WebView, sau đó dùng Activity này để hiển thị văn bản được host trực tuyến.

Kịch bản khác mà trong đó, WebView có thể hỗ trợ là khi ứng dụng của bạn cung cấp dữ liệu cho người dùng, những người luôn muốn yêu cầu kết nối Internet để truy xuất dữ liệu, chẳng hạn như e-mail. Trong trường hợp này, bạn có thể thấy thật dễ dàng để xây dựng một WebView trong ứng dụng Android của mình nhằm hiển thị trang Web với toàn bộ dữ liệu người dùng thay vì thực hiện một yêu cầu mạng, sau đó phân tích và hiển thị dữ liệu trên màn hình Android. Thay vì thế, bạn có thể thiết kế một trang Web phù hợp với thiết bị Android, sau đó cài đặt WebView trong ứng dụng Android để nạp trang Web.

Tài liệu này sẽ hướng dẫn bạn cách khởi động với WebView cũng như cách bổ sung thêm một số tính năng, chẳng hạn như xử lý điều hướng trang và liên kết JavaScript từ trang Web tới mã phía client (client-side code) trong ứng dụng Android.

Thêm WebView vào ứng dụng của bạn

Để thêm WebView vào ứng dụng của mình, bạn chỉ cần thêm phần tử <WebView> vào layout cho activity. Ví dụ, sau đây là một file layout, trong đó WebView sẽ hiển thị toàn màn hình:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

Để nạp trang Web trong WebView, sử dụng phương thức loadUrl(). Ví dụ:

```
WebView myWebView =(WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

Tuy nhiên, trước khi thực hiện mã trên, ứng dụng của bạn phải có truy cập vào Internet. Để lấy truy cập vào Internet, hãy yêu cầu quyền INTERNET trong file kê khai. Ví dụ:

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

Tất cả những gì bạn cần là một WebView cơ bản để hiển thị trang Web.

Sử dụng mã JavaScript trong WebView

Nếu một trang Web dự định sẽ nạp trong WebView sử dụng mã JavaScript, bạn phải cho phép chạy mã JavaScript trong WebView của mình. Khi cho phép JavaScript, bạn cũng có thể tạo giao diện giữa mã ứng dụng và mã JavaScript.

Cho phép chạy mã JavaScript

Mặc định, JavaScript bị chặn trong WebView. Bạn có thể cho phép chạy mã này thông qua đối tượng WebSettings được gắn vào WebView. Bạn có thể truy xuất WebSettings bằng phương thức getSettings(), sau đó cho phép chạy mã JavaScript bằng phương thức setJavaScriptEnabled().

Ví dụ:

```
WebView myWebView =(WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

WebSettings cung cấp truy cập vào rất nhiều thiết lập mà bạn có thể thấy hữu dụng. Ví dụ, nếu đang phát triển một ứng dụng Web được thiết kế dành riêng cho WebView trong ứng dụng Android, bạn có thể định nghĩa một chuỗi agent tùy chọn người dùng (custom user agent string) bằng phương thức setUserAgentString(), sau đó thực hiện truy vấn agent tùy chọn người dùng trong trang Web của mình để xác định client vấn tin trang Web của bạn thực sự là ứng dụng Android của bạn hay không.

Liên kết mã JavaScript với mã Android

Khi phát triển một ứng dụng Web được thiết kế riêng cho WebView trong ứng dụng Android, bạn có thể tạo giao diện giữa mã JavaScript và mã Android phía client. Ví dụ, mã JavaScript có thể gọi một phương thức trong mã Android để hiển thị một đối tượng Dialog (hộp thoại), thay vì sử dụng hàm `alert()` của JavaScript.

Để liên kết một giao diện mới giữa mã JavaScript với mã Android, bạn gọi phương thức addJavascriptInterface(), truyền cho phương thức này một thể hiện của lớp để liên kết JavaScript với tên một giao diện mà mã JavaScript có thể gọi để truy cập lớp này.

Ví dụ, bạn có thể đưa lớp sau vào ứng dụng Android của mình:

```
public class WebAppInterface {
    Context mContext;

    /** Tạo giao diện và thiết lập ngữ cảnh */
    WebAppInterface(Context c){
        mContext = c;
    }
```

```
    /** Hiển thị một thông điệp nhanh từ trang Web */
    @JavascriptInterface
    public void showToast(String toast){
        Toast.makeText(mContext, toast,Toast.LENGTH_SHORT).show();
    }
}
```

> **Chú ý** Nếu thiết lập `targetSdkVersion` là 17 hoặc cao hơn, bạn **phải thêm chú thích** `@JavascriptInterface` vào bất kỳ phương thức nào bạn muốn dùng cho JavaScript (phương thức này cũng phải là public). Nếu bạn không cung cấp chú thích này, trang Web của bạn sẽ không truy cập được phương thức đó khi chạy trên Android 4.2 hoặc phiên bản cao hơn.

Trong ví dụ dưới đây, lớp `WebAppInterface` cho phép trang Web tạo một thông điệp nhanh (`Toast`), sử dụng phương thức `showToast()`.

Bạn có thể liên kết lớp này với JavaScript chạy trong `WebView` của mình bằng phương thức `addJavascriptInterface()` và đặt tên cho giao diện `Android`. Ví dụ:

```
WebView webView =(WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new WebAppInterface(this), ”Android”);
```

Mã trên sẽ tạo ra một giao diện được gọi là `Android`, cho phép JavaScript chạy trong `WebView`. Khi đó, ứng dụng Web của bạn có thể truy cập vào lớp `WebAppInterface`. Ví dụ, dưới đây là một số mã HTML và JavaScript dùng để tạo thông điệp nhanh sẽ sử dụng giao diện mới khi người dùng nhấn vào button:

```
<input type=”button” value=”Say hello” onClick=”showAndroidToast
(‘Hello Android!’)” />

<script type=”text/javascript”>
    function showAndroidToast(toast){
        Android.showToast(toast);
    }
</script>
```

Không cần khởi tạo giao diện `Android` từ mã JavaScript. `WebView` sẽ tự động khiến giao diện này có thể sử dụng được cho trang Web của bạn. Do đó, khi nhấn vào button này, phương thức `showAndroidToast()` sử dụng giao diện `Android` để gọi phương thức `WebAppInterface.showToast()`.

> **Ghi chú:** Đối tượng này được liên kết với mã JavaScript chạy trong luồng khác và không nằm trong luồng đã xây dựng đối tượng này.

> **Chú ý** Sử dụng phương thức addJavascriptInterface() cho phép JavaScript điều khiển ứng dụng Android của bạn. Đây có thể là một tính năng hữu dụng, hoặc là một vấn đề bảo mật nguy hiểm. Khi HTML trong WebView có nguồn gốc không đáng tin (ví dụ, một phần hoặc tất cả HTML được một người hoặc tiến trình không xác định cung cấp), đối tượng tấn công có thể đưa vào mã HTML thực thi mã phía client của bạn và đưa vào bất cứ mã nào mà đối tượng tấn công muốn. Như vậy, bạn không nên sử dụng phương thức addJavascriptInterface(), trừ phi muốn ghi cả mã HTML lẫn JavaScript vào WebView của bạn. Bạn cũng không nên cho phép người dùng điều hướng sang trang Web khác không phải của mình, bên trong đối tượng WebView (thay vào đó, hãy cho phép ứng dụng trình duyệt mặc định mở liên kết ngoại (foreign link) - mặc định, trình duyệt Web của người dùng mở tất cả liên kết URL, nên bạn cần cẩn thận khi xử lý điều hướng trang như miêu tả ở mục sau).

Xử lý điều hướng trang

Khi người dùng nhấn vào liên kết từ trang Web trong WebView của bạn, hành vi mặc định của Android là khởi chạy một ứng dụng xử lý URL. Thường thì trình duyệt Web mặc định mở và nạp URL đích (destination URL). Tuy nhiên, bạn có thể ghi đè hành vi này cho WebView của mình; do đó, liên kết mở bên trong WebView của bạn. Sau đó, bạn có thể cho phép người dùng điều hướng lùi hoặc tiến thông qua history trang Web của họ được lưu trong WebView do bạn phát triển.

Để mở liên kết được người dùng nhấn vào, bạn chỉ cần cung cấp WebViewClient cho WebView của bạn, bằng cách sử dụng phương thức setWebViewClient(). Ví dụ:

```
WebView myWebView =(WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```

Như vậy là đến đây, tất cả liên kết người dùng nhấn vào đều được nạp bên trong WebView của bạn.

Nếu bạn muốn thêm điều khiển ở những nơi liên kết được nhấn nạp, hãy tạo WebViewClient của chính bạn, trong đó WebViewClient này ghi đè phương thức shouldOverrideUrlLoading(). Ví dụ:

```
private class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        if(Uri.parse(url).getHost().equals("www.example.com")){
            // Đây là trang Web của tôi, do đó không ghi đè; hãy để
            // WebView của tôi nạp trang
           return false;
        }
        // Ngược lại, liên kết này không phải là một trang trên miền
```

```
        // của tôi, nên hãy khởi động Activity khác để xử lý các URL
        Intent intent ==new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        startActivity(intent);
        return true;
    }
}
```

Sau đó, bạn tạo một thể hiện cho `WebViewClient` mới này của `WebView`:

```
WebView myWebView =(WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new MyWebViewClient());
```

Lúc này, khi người dùng nhấn vào một liên kết, hệ thống sẽ gọi `shouldOverrideUrlLoading()`, là phương thức sẽ kiểm tra xem liệu host của URL có phù hợp với một miền cụ thể (được định nghĩa như bên trên) không. Nếu phù hợp, phương thức này trả về false để *không* ghi đè việc nạp URL (điều này cho phép `WebView` nạp URL như thường lệ). Nếu host của URL không phù hợp, một `Intent` sẽ được tạo để khởi động Activity mặc định xử lý URL (xử lý trình duyệt mặc định của người dùng).

Điều hướng lịch sử trang Web

Khi `WebView` của bạn ghi đè việc nạp URL, WebView sẽ tự động lưu lại một history của các trang Web đã được ghé thăm. Bạn có thể điều hướng tiến và lùi thông qua history này bằng phương thức `goBack()` và `goForward()`.

Ví dụ, sau đây là cách thức `Activity` có thể sử dụng button Back của thiết bị để điều hướng lùi:

```
@Override
public boolean onKeyDown(int keyCode, KeyEventevent) {
    // Kiểm tra xem sự kiện phím có phải là của button Back không
    // và xem có history trang Web nào không.
    if((keyCode ==KeyEvent.KEYCODE_BACK)&& myWebView.canGoBack()){
        myWebView.goBack();
        return true;
    }
    // Nếu không phải phím Back hoặc không có history trang
    // Web, hành vi xử lý mặc định của hệ thống (có thể là
    // thoát khỏi activity) sẽ được sử dụng
    return super.onKeyDown(keyCode, event);
}
```

Phương thức `canGoBack()` trả về `true` nếu đây thực sự là history trang Web để người dùng ghé thăm. Tương tự, bạn có thể sử dụng phương thức `canGoForward()` để kiểm tra xem liệu đây có phải là history tiến hay không. Nếu bạn không thực hiện việc kiểm tra này, thì khi người dùng tiến tới cuối history, các phương thức `goBack()` hay `goForward()` đều không có tác dụng.