Fpt University

FPT POLYTECHNIC

Lập trình Android nâng cao

TÀI LIỆU LƯU HÀNH NỘI BỘ

PBNB 0.2_28.08.2015

MỤC LỤC

1. Định vị và Bản đồ

Lưu ý: Đây là tài liệu hướng dẫn về các API định vị của *framework Android* nằm trong gói android.location. Google Location Services API (API Dịch vụ Định vị của Google), là một phần của Google Play Services[1], cung cấp một framework cấp cao mạnh mẽ giúp tự động hóa các tác vụ như lựa chọn location provider (nguồn cung cấp kết quả định vị) và quản lý điện năng. Location Services còn cung cấp những chức năng mới vốn không có trong API của framework Android, chẳng hạn như chức năng phát hiện hoạt động của người dùng. Những lập trình viên đang sử dụng API của framework Android hay đang có kế hoạch hỗ trợ khả năng định vị trong ứng dụng, thực sự nên tính đến việc sử dụng Location Services API.

Để hiểu hơn về Location Services API, bạn hãy tham khảo trang Google Location Services for Android (tạm dịch: Google Location Services dành cho Android).

Các ứng dụng dựa trên khả năng định vị và bản đồ đem lại những trải nghiệm hấp dẫn trên các thiết bị di động. Bạn có thể tích hợp những tính năng này vào ứng dụng bằng cách sử dụng các lớp trong gói android.location và Google Maps Android API. Các phần dưới đây cung cấp thông tin cơ bản để bạn có thể tích hợp những tính năng này.

Location Services

Android cho phép các ứng dụng của bạn truy cập tới location services được thiết bị hỗ trợ thông qua các lớp trong gói android.location. Thành phần trung tâm của framework định vị này là service hệ thống LocationManager; service này cung cấp các API để xác định vị trí và hướng của thiết bị (nếu thiết bị hỗ trợ).

Cũng như với các service hệ thống khác, bạn không khởi tạo LocationManager trực tiếp. Thay vào đó, bạn yêu cầu một thể hiện từ hệ thống bằng cách gọi getSystemService(Context.LOCATION_SERVICE). Phương thức này trả về đối tượng tương ứng với một thể hiện mới của LocationManager.

Khi đã có LocationManager, ứng dụng của bạn có thể làm ba việc:

- Truy xuất danh sách tất cả các LocationProvider (nguồn cung cấp kết quả định vị) để thu được kết quả định vị lần cuối của người dùng.

[1] *Google Play Services là bộ API hỗ trợ làm việc với các dịch vụ của Google như Google Maps, Google+, Google Location Service APIs, Google Play Games…*

- Đăng ký/hủy đăng ký nhận cập nhật định kỳ về vị trí hiện tại của người dùng từ một location provider nào đó (được chỉ ra bằng các tiêu chí hoặc bằng tên).
- Đăng ký/hủy đăng ký một Intent nào đó, để Intent đó được kích hoạt khi thiết bị tiếp cận một vị trí nào đó (trong bán kính theo mét) với vĩ độ/kinh độ xác định.

Để hiểu hơn về cách xác định vị trí người dùng, bạn hãy đọc phần "Các chiến lược định vị" bên dưới.

Google Maps Android API

Với Google Maps Android API, bạn có thể thêm bản đồ vào ứng dụng dựa trên dữ liệu của Google Maps. API này tự động xử lý việc truy cập các máy chủ của Google Maps, việc tải dữ liệu, hiển thị bản đồ, và các thao tác cảm ứng (cử động chạm) trên bản đồ. Bạn cũng có thể sử dụng các lời gọi trong API này để đánh dấu, tạo các hình đa giác và các lớp phủ và thay đổi khung nhìn của người dùng trên một vùng bản đồ cụ thể.

Lớp then chốt trong Google Maps Android API là MapView.MapView hiển thị một bản đồ với dữ liệu lấy từ Google Maps service. Khi MapView đang nhận focus (đang được chọn), MapView sẽ thu các sự kiện gõ phím và các thao tác cảm ứng để di chuyển và thu phóng bản đồ một cách tự động, bao gồm cả việc gửi yêu cầu để lấy các ô bản đồ (map tile) bổ sung. MapView cũng cung cấp tất cả các thành phần giao diện người dùng cần thiết để người dùng điều khiển bản đồ. Ứng dụng của bạn cũng có thể sử dụng các phương thức của lớp MapView để điều khiển bản đồ bằng mã lập trình và vẽ một số lớp phủ lên bản đồ.

Các Google Maps Android API không nằm trong nền tảng Android, nhưng có thể cài đặt trên bất kỳ thiết bị nào có Google Play Store chạy Android 2.2 hoặc cao hơn thông qua Google Play Services.

Để tích hợp Google Maps vào ứng dụng, bạn cần cài đặt các thư viện của Google Play Services cho bộ Android SDK (Bộ công cụ phát triển phần mềm Android) của bạn. Để tìm hiểu thêm, bạn có thể tham khảo Google Play Services.

Các chiến lược định vị

- Những thử thách trong việc Xác định Vị trí Người dùng
- Yêu cầu Cập nhật Vị trí
  - Yêu cầu cấp Quyền Người dùng
- Thiết lập Mô hình cho Hiệu suất Tối ưu
  - Các bước để xác định vị trí người dùng
  - Quyết định khi nào bắt đầu lắng nghe cập nhật
  - Xác định nhanh vị trí với kết quả định vị lần cuối
  - Quyết định khi nào ngừng lắng nghe cập nhật
  - Duy trì vị trí ước tính tốt nhất
  - Điều chỉnh mô hình để tiết kiệm pin và trao đổi dữ liệu
- Cung cấp Dữ liệu Định vị Giả

CÁC LỚP TRỌNG TÂM

- LocationManager
- LocationListener

Lưu ý: Các chiến lược được mô tả trong tài liệu này áp dụng với API định vị trong gói android.location. Google Location Services API, một phần của Google Play Services, cung cấp một framework cấp cao mạnh mẽ giúp xử lý các location provider (nguồn cung cấp kết quả định vị), sự di chuyển của người dùng, và độ chính xác của vị trí. API này cũng xử lý việc lập lịch cập nhật vị trí dựa trên các thông số tiêu thụ điện năng do bạn cung cấp. Trong hầu hết các trường hợp, khi sử dụng Location Services API, bạn sẽ đạt được hiệu suất sử dụng pin tốt hơn, đồng thời có độ chính xác hợp lý hơn.

Để hiểu hơn về Location Services API, bạn hãy tham khảo Google Location Services for Android. Nắm được vị trí của người dùng cho phép ứng dụng của bạn xử lý thông minh hơn và đưa ra những thông tin hữu ích hơn cho người dùng. Khi phát triển một ứng dụng hỗ trợ định vị trên Android, bạn có thể sử dụng GPS và Network Location Provider (Nguồn cung cấp kết quả định vị bằng Mạng) của Android để thu được vị trí người dùng. Dù định vị qua GPS chính xác nhất, GPS chỉ hoạt động ngoài trời, tiêu tốn năng lượng pin nhanh chóng, và không thu được kết quả định vị nhanh như người dùng mong đợi. Network Location Provider của Android xác định vị trí người dùng bằng tín hiệu từ trạm

thu phát sóng thông tin di động (cell tower) và tín hiệu Wi-Fi, từ đó có thể cung cấp thông tin định vị khi người dùng ở trong nhà hay ngoài trời, thời gian đáp ứng nhanh hơn và dùng ít năng lượng pin hơn. Trong ứng dụng của bạn, để lấy được vị trí người dùng, bạn có thể sử dụng cả GPS và Network Location Provider, hoặc một trong hai.

Những thử thách trong việc Xác định Vị trí Người dùng

Truy xuất vị trí người dùng từ một thiết bị di động có thể rất phức tạp. Có một số lý do giải thích vì sao một kết quả định vị (bất kể nguồn cung cấp) có thể có lỗi hoặc không chính xác. Một số nguyên nhân gây lỗi trong việc định vị người dùng bao gồm:

- **Nhiều nguồn cung cấp vị trị**

  GPS, Cell-ID và Wi-Fi đều có thể cung cấp thông tin về vị trí người dùng. Việc quyết định sử dụng và tin tưởng thông tin nào cần đánh đổi giữa độ chính xác, tốc độ và hiệu suất sử dụng pin.

- **Sự di chuyển của người dùng**

  Vì vị trí người dùng thay đổi, bạn phải tính toán sự di chuyển đó bằng cách thường xuyên ước tính lại vị trí người dùng.

- **Độ chính xác khác nhau**

  Kết quả định vị ước tính từ mỗi nguồn cung cấp không đồng nhất về độ chính xác. Một vị trí có được 10 giây trước từ một nguồn có thể chính xác hơn vị trí mới nhất từ một nguồn khác hoặc chính nguồn đó.

Các vấn đề này tạo trở ngại trong việc truy xuất một kết quả tin cậy về vị trí người dùng. Tài liệu này cung cấp thông tin giúp bạn vượt qua những trở ngại này để có được kết quả định vị tin cậy. Tài liệu này cũng cung cấp một số ý tưởng bạn có thể sử dụng trong ứng dụng của mình để cung cấp trải nghiệm định vị địa lý chính xác và nhanh nhạy cho người dùng.

Yêu cầu Cập nhật Vị trí

Trước khi xử lý một số lỗi định vị nêu trên, bạn cần hiểu căn bản cách xác định vị trí người dùng trên Android.

Truy xuất vị trí người dùng trong Android cần sử dụng một phương thức callback. Bạn cần chỉ ra rằng bạn muốn nhận cập nhật vị trí từ LocationManager bằng cách gọi phương thức requestLocationUpdates() với tham số là một LocationListener. LocationListener phải hiện thực hóa (implement - viết mã xử lý cụ thể cho phương thức) một số phương thức callback mà Location Manager cần gọi khi vị trí người dùng thay đổi hoặc khi trạng thái của dịch vụ thay đổi.

Ví dụ, đoạn mã sau minh họa cách định nghĩa một LocationListener và yêu cầu cập nhật vị trí:

```
// Lấy một tham chiếu đến Location Manager của hệ thống
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// Khai báo một listener để đáp ứng các cập nhật vị trí
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Được gọi khi một vị trí mới được tìm thấy bởi
      // Network Location Provider
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status,
                                Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
  };

// Đăng ký listener với Location Manager để nhận cập nhật vị trí
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                                       0, 0,  locationListener);
```

Tham số đầu tiên trong phương thức requestLocationUpdates() - LocationManager.NETWORK_PROVIDER - là kiểu location provider (trong ví dụ này là Network Location Provider với khả năng định vị dựa trên thông tin từ trạm thu phát sóng và Wi-Fi). Bạn có thể kiểm soát tần suất lắng nghe cập nhật của listener với tham số thứ hai và thứ ba, tham số thứ hai là khoảng thời gian tối thiểu giữa các lần thông báo và tham số thứ ba là khoảng cách thay đổi tối thiểu giữa các lần thông báo – đặt cả hai tham số là 0 sẽ yêu cầu thông báo vị trí nhiều nhất có thể. Tham số cuối cùng là đối tượng LocationListener của bạn, đối tượng này sẽ nhận callback về các cập nhật vị trí.

Để yêu cầu cập nhật vị trí từ GPS provider, bạn hãy thay thế GPS_PROVIDER vào NETWORK_PROVIDER. Bạn cũng có thể yêu cầu cập nhật vị trí từ cả GPS và Network Location Provider bằng cách gọi requestLocationUpdates() hai lần – một lần cho NETWORK_PROVIDER và một lần cho GPS_PROVIDER.

Yêu cầu cấp Quyền Người dùng

Để nhận cập nhật vị trí từ NETWORK_PROVIDER hoặc GPS_PROVIDER, bạn phải yêu cầu cấp quyền người dùng bằng cách khai báo riêng rẽ quyền ACCESS_COARSE_LOCATION hoặc quyền ACCESS_FINE_LOCATION trong file kê khai của Android. Ví dụ:

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

Thiếu các quyền này, ứng dụng của bạn sẽ gặp lỗi thực thi - khi yêu cầu cập nhật vị trí.

> **Lưu ý:** Nếu bạn dùng cả NETWORK_ PROVIDER và GPS_PROVIDER, bạn chỉ cần khai báo quyền ACCESS_FINE_LOCATION vì quyền này cho phép sử dụng cả hai provider trên. (Quyền ACCESS_COARSE_LOCATION chỉ cấp phép sử dụng NETWORK_PROVIDER.)

Thiết lập Mô hình cho Hiệu suất Tối ưu

Hiện nay, các ứng dụng dựa trên vị trí thiết bị rất phổ biến, tuy nhiên do hạn chế về độ chính xác, sự di chuyển của người dùng, cách thức định vị đa dạng và yêu cầu tiết kiệm pin, việc lấy được vị trí người dùng rất phức tạp. Để khắc phục những trở ngại này và có được vị trí chính xác của người dùng đồng thời tiết kiệm pin, bạn phải thiết lập một mô hình hoạt động nhất quán cho ứng dụng. Mô hình này chỉ ra cách xác định vị trí người dùng, thời điểm bạn bắt đầu và kết thúc lắng nghe cập nhật và thời điểm sử dụng dữ liệu vị trí từ cache (bộ nhớ đệm).

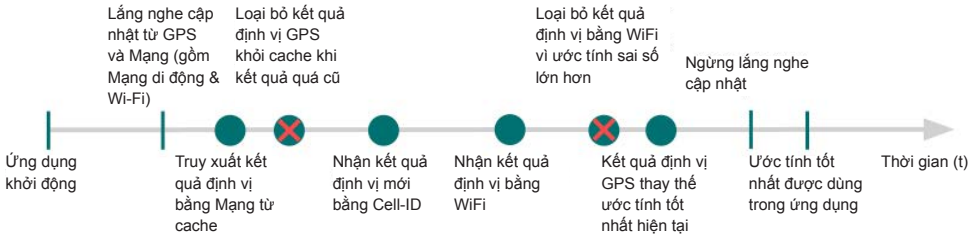
Các bước để xác định vị trí người dùng

Sau đây là các bước điển hình để xác định vị trí người dùng:

1. Khởi động ứng dụng
2. Sau một lúc, bắt đầu lắng nghe cập nhật từ các location provider muốn dùng.

3. Duy trì "vị trí ước tính tốt nhất hiện thời" bằng cách loại bớt các kết quả định vị mới, kém chính xác hơn.
4. Ngừng lắng nghe cập nhật vị trí.
5. Sử dụng vị trí ước tính tốt nhất trước đó.

Hình 1.1 giải thích mô hình này theo một trục thời gian biểu diễn khung thời gian trong đó ứng dụng lắng nghe cập nhật vị trí và các sự kiện có thể xảy ra.



Hình 1.1. Trục thời gian biểu diễn khung thời gian trong đó ứng dụng lắng nghe cập nhật vị trí.

Khung thời gian cho mô hình này – trong đó các cập nhật vị trí được xử lý – đòi hỏi bạn phải đưa ra nhiều quyết định để có thể hỗ trợ chức năng định vị trong ứng dụng của bạn.

Quyết định khi nào bắt đầu lắng nghe cập nhật

Bạn có thể bắt đầu lắng nghe cập nhật vị trí ngay khi ứng dụng khởi động, hoặc chỉ ngay sau khi người dùng kích hoạt một tính năng nào đó. Lưu ý khi lắng nghe cập nhật vị trí, sử dụng các quãng thời gian dài có thể tốn rất nhiều pin, nhưng các quãng thời gian ngắn lại có thể không mang lại kết quả đủ chính xác.

Như ví dụ bên dưới, bạn có thể bắt đầu lắng nghe cập nhật bằng cách gọi requestLocationUpdates():

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Hoặc, sử dụng dữ liệu vị trí GPS:
// String locationProvider = LocationManager.GPS_PROVIDER;

locationManager.requestLocationUpdates(locationProvider, 0, 0,
                                       locationListener);
```

Xác định nhanh vị trí với kết quả định vị trước đó

Thời gian cần để người dùng nhận được kết quả định vị đầu tiên cũng mất khá lâu. Trong khi chờ kết quả định vị chính xác hơn, bạn nên tận dụng vị trí đã xác định từ trước trong cache thông qua phương thức getLastKnownLocation(String):

```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Hoặc sử dụng LocationManager.GPS_PROVIDER

Location lastKnownLocation =
      locationManager.getLastKnownLocation(locationProvider);
```

Quyết định khi nào ngừng lắng nghe cập nhật

Tùy vào ứng dụng của bạn, quyết định khi nào không cần nhận cập nhật vị trí nữa có thể rất đơn giản hoặc rất phức tạp. Độ chính xác của vị trí ước tính được cải thiện trong quãng thời gian ngắn từ khi vị trí được xác định cho đến khi vị trí được sử dụng. Luôn chú ý rằng lắng nghe cập nhật trong thời gian dài tốn rất nhiều pin, vì vậy ngay khi có đủ thông tin, bạn nên ngừng lắng nghe cập nhật bằng cách gọi removeUpdates(PendingIntent):

```
// Xóa listener bạn đã thêm vào trước đó
locationManager.removeUpdates(locationListener);
```

Duy trì vị trí ước tính tốt nhất

Có thể bạn nghĩ rằng kết quả định vị mới nhất là chính xác nhất. Tuy nhiên, vì độ chính xác của các kết quả định vị không đồng nhất, kết quả mới nhất không phải luôn là kết quả tốt nhất. Bạn nên tính đến việc lựa chọn các kết quả định vị dựa trên nhiều tiêu chí. Các tiêu chí thay đổi tùy vào các tình huống sử dụng trong ứng dụng và kết quả thử nghiệm ứng dụng trong thực tế.

Sau đây là một số bước bạn có thể áp dụng để đánh giá độ chính xác của một kết quả định vị:

- Kiểm tra kết quả định vị thu được có mới hơn đáng kể so với ước tính trước đó không.
- Kiểm tra xem độ chính xác của vị trí vừa thu được tốt hơn hay kém hơn ước tính trước đó.
- Kiểm tra provider vừa đưa ra kết quả định vị mới và quyết định liệu kết quả có đáng tin cậy không.

Để thực hiện điều này, bạn có thể tham khảo ví dụ sau:

```
private static final int TWO_MINUTES = 1000*60*2;

/** Quyết định kết quả định vị thu được có tốt hơn vị trí hiện có không.
  * @param location  Vị trí mới bạn muốn đánh giá
  * @param currentBestLocation  Vị trí tốt nhất hiện có, để so sánh
  * với kết quả mới thu được
  */
protected boolean isBetterLocation(Location location, Location
currentBestLocation) {
    if (currentBestLocation == null){
        // Kết quả nào cũng tốt hơn là không có gì
        return true;
    }
    // Kiểm tra xem kết quả định vị thu được mới hơn hay cũ hơn
    long timeDelta = location.getTime() - currentBestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
    boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
    boolean isNewer = timeDelta > 0;

    // Nếu vị trí mới được cập nhật sau vị trí hiện có hai phút,
    // sử dụng vị trí mới vì có thể người dùng đã di chuyển
    if (isSignificantlyNewer) {
        return true;
    // Nếu vị trí thu được cũ hơn hai phút, vị trí đó hẳn là không
    // chính xác
    } else if (isSignificantlyOlder) {
        return false;
    }

    // Kiểm tra kết quả định vị mới chính xác hơn hay ngược lại
    int accuracyDelta =(int) (location.getAccuracy() -
                              currentBestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    // Kiểm tra vị trí cũ và mới có từ cùng một provider không
    boolean isFromSameProvider = isSameProvider(location.getProvider(),
                                    currentBestLocation.getProvider());

    // Quyết định chất lượng vị trí bằng cách đánh giá kết hợp thời
    // gian cập nhật và độ chính xác
    if (isMoreAccurate) {
        return true;
    } else if (isNewer && !isLessAccurate) {
```

```
        return true;
    } else if (isNewer && !isSignificantlyLessAccurate &&
                isFromSameProvider) {
        return true;
    }
    return false;
}
/** Kiểm tra xem liệu hai provider có giống nhau không */
private boolean isSameProvider(String provider1, String provider2)
{
    if (provider1 == null) {
      return provider2 == null;
    }
      return provider1.equals(provider2);
}
```

Điều chỉnh mô hình để tiết kiệm pin và trao đổi dữ liệu

Trong khi kiểm thử ứng dụng, bạn có thể nhận thấy nhu cầu điều chỉnh mô hình để cung cấp vị trí chính xác và đồng thời đạt hiệu suất cao. Sau đây là một số việc bạn có thể thực hiện để cân bằng giữa hai nhu cầu đó.

Giảm khung thời gian

Với khung thời gian hẹp lại, khi lắng nghe cập nhật vị trí, ứng dụng tương tác ít hơn với GPS và network location services, nhờ đó kéo dài thời lượng pin. Tuy nhiên, điều đó cũng đồng nghĩa với ít vị trí để chọn hơn trong khi tìm vị trí ước tính tốt nhất.

Giảm tần suất cập nhật từ các location provider

Giảm tần suất cập nhật mới xuất hiện trong khung thời gian cũng có thể cải thiện hiệu suất dùng pin, nhưng đổi lại là giảm độ chính xác. Giá trị của việc đánh đổi này phụ thuộc vào cách người dùng sử dụng ứng dụng của bạn. Bạn có thể giảm tần suất cập nhật bằng cách tăng giá trị của các tham số chỉ định khoảng thời gian và khoảng cách thay đổi tối thiểu khi bạn gọi phương thức requestLocationUpdates().

Hạn chế số provider

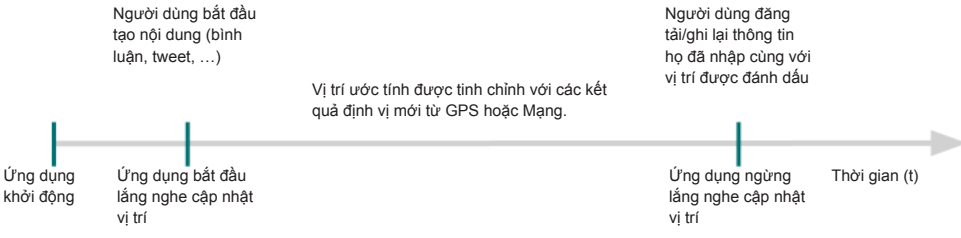
Tùy vào môi trường nơi ứng dụng của bạn được sử dụng hoặc nhu cầu về mức độ chính xác, bạn có thể lựa chọn chỉ dùng Network Location Provider hoặc chỉ dùng GPS, thay vì cả hai. Chỉ tương tác với một trong những service này giúp giảm tiêu hao pin, đi kèm là khả năng giảm độ chính xác.

Một số tình huống ứng dụng phổ biến

Có nhiều lý do khiến bạn muốn xác định vị trí người dùng trong ứng dụng. Dưới đây là một số tình huống trong đó bạn có thể sử dụng vị trí người dùng để cải tiến ứng dụng. Mỗi tình huống đều mô tả những cách làm tốt cho việc xác định thời điểm bạn nên bắt đầu và kết thúc lắng nghe cập nhật vị trí, nhằm có được kết quả tốt và giúp kéo dài thời lượng pin.

Đánh dấu nội dung do người dùng tạo với thông tin vị trí

Bạn có thể tạo một ứng dụng trong đó nội dung do người dùng tạo được đánh dấu vị trí. Có thể người dùng đang chia sẻ kinh nghiệm về nơi họ ở, đăng bài đánh giá về một nhà hàng, hoặc ghi lại một số thông tin về vị trí hiện tại của họ. Mô hình trong Hình 1.2 minh họa một tình huống như vậy liên quan đến location services.



Hình 1.2. Trục thời gian biểu diễn khung thời gian trong đó vị trí người dùng được truy xuất và việc lắng nghe dừng lại khi người dùng sử dụng vị trí hiện tại.

Mô hình này kết hợp với mô hình trước đó về cách thức truy xuất vị trí người dùng bằng mã lập trình (Hình 1.1). Để có kết quả định vị chính xác nhất, bạn có thể quyết định bắt đầu lắng nghe cập nhật khi người dùng bắt đầu tạo nội dung hoặc khi ứng dụng khởi động, sau đó ngừng lắng nghe cập nhật khi nội dung đã sẵn sàng để đăng tải hoặc ghi lại. Bạn có thể cần cân nhắc lượng thời gian mà người dùng thường cần để tạo nội dung và đánh giá xem liệu thời gian đó có cho phép thu thập vị trí ước tính một cách hiệu quả hay không.

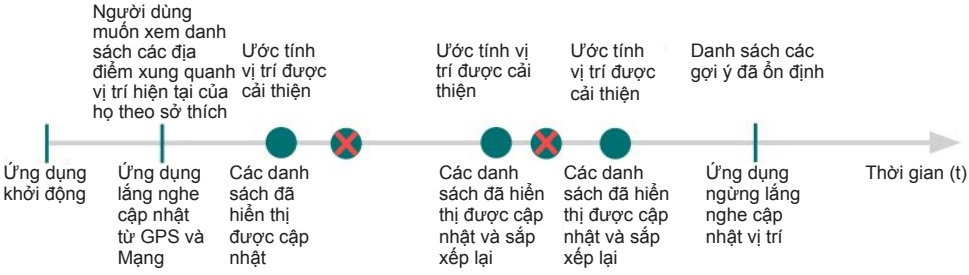
Giúp người dùng quyết định địa điểm sẽ tới

Bạn có thể tạo một ứng dụng cung cấp cho người dùng một danh sách các lựa chọn về nơi họ sẽ đến. Ví dụ, bạn cung cấp một danh sách các nhà hàng, cửa hàng và địa điểm giải trí ở khu vực xung quanh và thứ tự các gợi ý sẽ thay đổi tùy thuộc vào vị trí người dùng.

Để đáp ứng nhu cầu như vậy, bạn có thể quyết định:

- Sắp xếp lại các gợi ý khi có được một vị trí ước tính tốt nhất mới.
- Ngừng lắng nghe cập nhật nếu thứ tự các gợi ý đã ổn định

Kiểu mô hình này được minh họa trong Hình 1.3.



Hình 1.3. Trục thời gian biểu diễn khung thời gian trong đó một tập dữ liệu động được cập nhật mỗi khi vị trí người dùng được cập nhật.

Cung cấp Dữ liệu Định vị Giả

Khi phát triển ứng dụng, chắc chắn bạn cần kiểm thử khả năng xác định vị trí người dùng của mô hình của mình. Điều này có thể được thực hiện dễ dàng nhất với một thiết bị Android thật. Tuy nhiên, nếu không có thiết bị nào, bạn vẫn có thể kiểm thử các tính năng dựa trên định vị với dữ liệu định vị giả trong bộ giả lập Android. Có ba cách để gửi dữ liệu định vị giả tới ứng dụng. Dùng Eclipse, DDMS hoặc lệnh "geo" trong console (giao diện dòng lệnh) của bộ giả lập.

> **Lưu ý:** Việc cung cấp dữ liệu định vị giả được đưa vào dưới dạng dữ liệu định vị GPS, vì vậy bạn phải yêu cầu cập nhật vị trí từ GPS_PROVIDER để có thể sử dụng dữ liệu định vị giả.

Sử dụng Eclipse

Chọn **Window** > **Show View** > **Other** > **Emulator Control**.

Trong khung Emulator Control, dưới phần Location Controls, bạn có thể nhập tọa độ GPS dưới dạng kinh vĩ độ/kinh độ riêng biệt; ngoài ra, bạn có thể nhập một file GPX để biểu diễn tuyến đường, hoặc một file KML để biểu diễn các địa điểm đã đánh dấu. (Hãy đảm bảo bạn đã chọn một thiết bị trong khung Devices – có trong menu **Window** > **Show View** > **Other** > **Devices.**)

Sử dụng DDMS

Với công cụ DDMS, bạn có thể giả lập dữ liệu định vị theo nhiều cách khác nhau:

- Gửi thủ công từng thông tin kinh độ/vĩ độ riêng biệt tới thiết bị.
- Sử dụng một file GPX mô tả tuyến đường để biểu diễn trên thiết bị.
- Sử dụng một file KML mô tả các địa điểm đã đánh dấu để biểu diễn tuần tự trên thiết bị.

Để tìm hiểu thêm về cách sử dụng DDMS để mô phỏng dữ liệu định vị, bạn hãy đọc bài Sử dụng DDMS.

Dùng lệnh "geo" trong console của bộ giả lập.

Để gửi dữ liệu định vị giả từ dòng lệnh:

1. Khởi động ứng dụng của bạn trong bộ giả lập Android và mở một cửa sổ console tại thư mục /tools trong bộ SDK của bạn.
2. Kết nối tới console của bộ giả lập:

```
telnet localhost <cổng-của-console>
```

3. Gửi dữ liệu định vị:
    - Dùng lệnh geo fix để gửi một vị trí địa lý cố định.

Lệnh này nhận một giá trị kinh độ và một giá trị vĩ độ bằng số thập phân, và một giá trị không bắt buộc về độ cao tính theo mét. Ví dụ:

```
geo fix -121.45356 46.51119 4392
```

- Dùng lệnh geo nmea để gửi một câu lệnh NMEA 0183.

Lệnh này nhận duy nhất một câu lệnh NMEA theo dạng '$GPGGA' (dữ liệu kết quả định vị) or '$GPRMC' (dữ liệu truyền tải). Ví dụ:

```
geo nmea $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,
011.3,E*62
```

Để tìm hiểu thêm về cách thức kết nối tới console của bộ giả lập, bạn hãy đọc Sử dụng cửa sổ Console của bộ giả lập.

2. MapView

> **Lưu ý:** Phiên bản 1 của Google Maps Android API đã chính thức bị khuyến cáo sẽ ngừng sử dụng trong tương lai (deprecated) từ ngày 3 tháng 12, 2012. Điều đó có nghĩa từ ngày 18 tháng 3, 2013 bạn sẽ không thể yêu cầu khóa API mới cho phiên bản này và sẽ không có tính năng mới cho Google Maps Android API phiên bản 1. Tuy nhiên, các ứng dụng đang sử dụng phiên bản 1 sẽ vẫn tiếp tục hoạt động trên các thiết bị. Các nhà phát triển cũ và mới đều được khuyến khích sử dụng Google Maps Android API phiên bản 2.

Với thư viện Google Maps, bạn có thể tự mình tạo một Activity hiển thị bản đồ. Trong bài hướng dẫn này, bạn sẽ tạo một ứng dụng bản đồ đơn giản theo hai phần. Trong Phần 1, bạn sẽ tạo một ứng dụng hiển thị một bản đồ mà người dùng có thể di chuyển màn hình hiển thị[1] và thu phóng. Trong Phần 2, bạn sẽ tạo thêm các đối tượng phủ[2] để đánh dấu các địa điểm quan tâm.

Bài hướng dẫn này đòi hỏi bạn có thư viện Google Maps bên ngoài đã được cài đặt trong môi trường SDK. Thư viện Google Maps này nằm trong Google APIs add-on, bạn có thể cài đặt add-on này bằng Android SDK và AVD Manager (Trình quản lý Thiết bị Android Ảo). Bạn có thể đọc thêm bài Installing the Google APIs Add-On (tạm dịch: Cài đặt Goolge APIs Add-on).

Sau khi cài đặt Google APIs add-on cho SDK, bạn cần thiết lập các thuộc tính của project (dự án) để sử dụng Phiên bản đích (Build target[3]) có tên "Google APIs by Google Inc."[4] (tạm dịch: Google APIs của công ty Google). Tùy thuộc vào nhu cầu, bạn có thể xem hướng dẫn thiết lập Phiên bản đích qua bài viết Creating and Managing Projects in Eclipse (tạm dịch: Tạo và quản lý project trong Eclipse) hoặc Creating and Managing Projects on the Command Line (tạm dịch: Tạo và quản lý project trên giao diện dòng lệnh).

Bạn cũng cần cài đặt một AVD (Thiết bị Android Ảo) mới sử dụng Phiên bản đích tương ứng bản Google API đã cài đặt. Bạn có thể đọc thêm bài Creating and Managing Virtual Devices (tạm dịch: Tạo và quản lý các thiết bị ảo).

Tài liệu tham khảo có tại Google Maps library documentation (tạm dịch: Tài liệu về thư viện Google Maps).

[1] *Người dùng có thể di chuyển màn hình hiển thị lên, xuống, sang ngang nhằm hiển thị những phần dữ liệu địa lý hiện đang nằm bên ngoài màn hình hiển thị ở tỷ lệ hiện thời.*

[2] *Đối tượng phủ trong Google Maps là các đối tượng được vẽ lên trên bản đồ tại các tọa độ nhất định, có thể là một điểm, một đường thẳng, một hình tròn hay hình đa giác…*

[3] *Build Target (tạm dịch: Phiên bản đích) chỉ định phiên bản Android ứng dụng của bạn sẽ hỗ trợ.*

[4] *Goolge APIs bao hàm Android API tương ứng với từng phiên bản Android.*

Tạo Map Activity

1. Tạo một project mới có tên *HelloGoogleMaps.*
2. Vì thư viện Maps không nằm trong thư viện Android tiêu chuẩn, bạn phải khai báo nó trong file kê khai của Android. Mở file AndroidManifest.xml và thêm dòng sau bên trong phần tử <application>:

```
<uses-library android:name="com.google.android.maps" />
```

3. Bạn cũng cần kết nối Internet để tải về dữ liệu các ô bản đồ, vì vậy bạn phải yêu cầu quyền INTERNET. Trong file kê khai, hãy thêm dòng sau bên trong phần tử <manifest>:

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

4. Để có thêm không gian cho bản đồ, bạn có thể loại bỏ thanh tiêu đề bằng cách sử dụng theme "NoTitleBar":

```
<activity android:name=".HelloGoogleMaps" android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar">
```

5. Mở file res/layout/main.xml và khai báo một đối tượng MapView duy nhất ở nút gốc.

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.maps.MapView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/mapview"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:clickable="true" android:apiKey="API_KEY"
/>
```

Thuộc tính android:clickable chỉ định bạn có cho phép người dùng tương tác với bản đồ hay không. Nếu đặt là "false", thao tác chạm lên bản đồ sẽ không có phản ứng gì.

Thuộc tính android:apiKey chứa khóa API của Google Maps Android API phiên bản 1 cho ứng dụng của bạn, thuộc tính này chứng minh ứng dụng của bạn và chứng chỉ của người ký (signer certificate) đã được đăng ký với dịch vụ Google Maps.

Đây là thuộc tính bắt buộc để nhận được dữ liệu bản đồ, kể cả khi bạn đang phát triển. Việc đăng ký không còn được hỗ trợ, nhưng bạn có thể sử dụng một khóa API hiện có của Google Maps Android API.

6. Mở file HelloGoogleMaps.java. Với Activity này, bạn hãy kế thừa MapActivity thay vì android.app.Activity:

```
public class HelloGoogleMaps extends MapActivity
```

MapActivity là một lớp con đặc biệt của Activity, nằm trong thư viện Maps, cung cấp các tính năng quan trọng để làm việc với bản đồ.

7. Trong mỗi MapActivity, phương thức isRouteDisplayed() là bắt buộc, vì vậy bạn cần ghi đè (override) phương thức này:

```
@Override
protected boolean isRouteDisplayed() {
        return false;
}
```

Phương thức này là bắt buộc vì một số tính toán trong dịch vụ Maps cần biết hiện bạn có đang hiển thị thông tin tuyến đường nào không. Trong trường hợp này, bạn không cần tính năng đó, vì vậy trả về giá trị false.

8. Thêm phương thức calback onCreate() tiêu chuẩn vào lớp:

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

Phương thức này nạp file layout (bố cục) đã tạo bên trên. Trên thực tế, đây đã là một ứng dụng chạy được, hiển thị các ô bản đồ và cho phép người dùng di chuyển màn hình hiển thị trong bản đồ nhưng không có khả năng thu phóng. Thật may, lớp MapView có sẵn tính năng xử lý thu phóng rất đơn giản, và bạn có thể sử dụng bằng cách gọi setBuiltInZoomControls(). Bạn hãy gọi phương thức này ở cuối phương thức onCreate().

```
MapView mapView = (MapView) findViewById(R.id.mapview);
mapView.setBuiltInZoomControls(true);
```

9. Như vậy là xong. Chạy ứng dụng. Hãy nhớ bạn phải có một thiết bị ảo được cấu hình để sử dụng phiên bản đích là Google APIs, hoặc bạn cần có một thiết bị thật có chứa thư viện Maps.

Tạo các Đối tượng Phủ

Giờ bạn đã một bản đồ, nhưng trong nhiều trường hợp bạn còn muốn tạo ra các điểm đánh dấu và lớp phủ ngay trên bản đồ. Đó chính là điều bạn sẽ làm ngay bây giờ. Đầu tiên, bạn phải kế thừa lớp ItemizedOverlay vốn có nhiệm vụ quản lý tập hợp các đối tượng phủ Overlay (các đối tượng riêng lẻ được đặt trên bản đồ).

1. Tạo một lớp Java có tên HelloItemizedOverlay kế thừa lớp ItemizedOverlay: Với Eclipse, nháy phải chuột vào tên gói trong Eclipse Package Explorer và chọn **New > Class**. Đặt giá trị cho trường **Name** là *HelloItemizedOverlay.* Trong mục **Superclass** (lớp cha), nhập *"com.google.android.maps.ItemizedOverlay"*. Chọn ô *Constructors from superclass* (Các phương thức khởi tạo từ lớp cha). Chọn *Finish*.

2. Trước tiên, bạn cần một ArrayList cho OverlayItem, trong đó bạn sẽ chứa từng đối tượng OverlayItem muốn hiển thị trên bản đồ. Bạn hãy thêm dòng sau vào phần đầu lớp HelloItemizedOverlay:

```
private ArrayList<OverlayItem> mOverlays =
    new ArrayList<OverlayItem>();
```

3. Giờ bạn hãy định nghĩa các phương thức khởi tạo cho HelloItemizedOverlay. Phương thức khởi tạo phải chỉ định điểm đánh dấu mặc định cho mỗi đối tượng OverlayItem. Để đối tượng dạng Drawable được thực sự hiển thị, ranh giới của đối tượng cần được định rõ. Thông thường, bạn sẽ muốn điểm giữa ở đáy bức ảnh làm điểm mà tại đó điểm đánh dấu mặc định gắn vào tọa độ trên bản đồ. Bạn có thể làm được điều này khi gọi phương thức boundCenterBottom() với tham số là đối tượng defaultMarker. Lời gọi tới phương thức khởi tạo của lớp cha sẽ giống như sau:

```
public HelloItemizedOverlay(Drawable defaultMarker) {
   super(boundCenterBottom(defaultMarker));
}
```

4. Để thêm các đối tượng OverlayItem mới vào mảng ArrayList, bạn cần một phương thức mới:

```
public void addOverlay(OverlayItem overlay) {
    mOverlays.add(overlay);
    populate();
}
```

Mỗi lần bạn thêm một OverlayItem mới vào mảng ArrayList, bạn phải gọi phương thức populate() của lớp ItemizedOverlay, phương thức này sẽ đọc mỗi đối tượng OverlayItem và chuẩn bị cho việc hiển thị chúng.

5. Khi chạy, phương thức populate() sẽ gọi createItem(int) trong lớp ItemizedOverlay để truy xuất từng đối tượng OverlayItem. Bạn phải ghi đè phương thức này để đọc từ mảng ArrayList và trả về đối tượng OverlayItem ở vị trí định bởi số nguyên đã cho. Phương thức ghi đè của bạn sẽ như sau:

```
@Override
protected OverlayItem createItem(int i) {
    return mOverlays.get(i);
}
```

6. Bạn cũng phải ghi đè phương thức size() để trả về số lượng các đối tượng hiện có trong mảng ArrayList:

```
@Override
public int size() {
    return mOverlays.size();
}
```

7. Giờ bạn có thể thiết lập tính năng xử lý các sự kiện cảm ứng trên các đối tượng phủ. Đầu tiên, bạn cần một tham chiếu tới đối tượng Context của ứng dụng dưới dạng một thành viên của lớp HelloItemizedOverlay. Khai báo thành viên Context mContext, sau đó khởi tạo biến trong một phương thức khởi tạo mới:

```
public HelloItemizedOverlay(Drawable defaultMarker, Context context) {
    super(boundCenterBottom(defaultMarker));
    mContext = context;
}
```

Phương thức khởi tạo này truyền defaultMarker lên phương thức khởi tạo mặc định để khai báo tọa độ của điểm đánh dấu mặc định và sau đó khởi tạo biến mContext với đối tượng Context được cung cấp.

Tiếp theo, bạn cần ghi đè phương thức callback onTap(); phương thức này sẽ xử lý sự kiện khi một đối tượng được người dùng chạm vào.

```
@Override
protected boolean onTap(int index) {
    OverlayItem item = mOverlays.get(index);
    AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
    dialog.setTitle(item.getTitle());
    dialog.setMessage(item.getSnippet());
    dialog.show();
    return true;
}
```

Đoạn mã sử dụng biến thành viên android.content.Context để tạo mới một đối tượng AlertDialog.Builder và sử dụng tiêu đề và phần tóm tắt của đối tượng OverlayItem - mà người dùng chạm vào - cho tiêu đề và nội dung của hộp thoại. (Bạn sẽ thấy phần định nghĩa tiêu đề và phần tóm tắt của đối tượng OverlayItem khi bạn tạo đối tượng như bên dưới).

Giờ bạn đã hoàn thành lớp HelloItemizedOverlay và có thể bắt đầu sử dụng lớp này để tạo các đối tượng trên bản đồ.

Trở lại lớp HelloGoogleMaps. Trong bước tiếp theo, bạn sẽ tạo một đối tượng OverlayItem và thêm nó vào một thể hiện của lớp HelloItemizedOverlay, và sau đó thêm đối tượng HelloItemizedOverlay vào MapView bằng cách sử dụng một đối tượng GeoPoint để chỉ định tọa độ trên bản đồ.

1. Đầu tiên, bạn cần có ảnh cho đối tượng phủ bản đồ. Kéo bức ảnh vào thư mục res/drawable/ trong project của bạn.

2. Ở cuối phương thức onCreate(), khởi tạo:

```
List<Overlay> mapOverlays = mapView.getOverlays();
Drawable drawable =
      this.getResources().getDrawable(R.drawable.androidmarker);
HelloItemizedOverlay itemizedoverlay =
      new HelloItemizedOverlay(drawable, this);
```

Tất cả các đối tượng phủ trên bản đồ được giữ bởi đối tượng MapView, vì vậy khi bạn muốn thêm đối tượng phủ, bạn phải lấy một danh sách từ phương thức getOverlays(). Sau đó, bạn cần khởi tạo đối tượng Drawable dùng làm điểm đánh dấu trên bản đồ, được lưu trong thư mục res/drawable/. Phương thức khởi tạo lớp HelloItemizedOverlay (lớp tùy chỉnh từ lớp ItemizedOverlay) nhận đối tượng Drawable làm điểm đánh dấu mặc định cho tất cả các đối tượng phủ.

3. Giờ bạn có thể tạo một đối tượng GeoPoint định nghĩa tọa độ cho đối tượng phủ đầu tiên và truyền đối tượng GeoPoint này vào phương thức khởi tạo đối tượng OverlayItem:

```
GeoPoint point = new GeoPoint(19240000,-99120000);
OverlayItem overlayitem = new OverlayItem(point, "Hola, Mundo!",
                                          "I'm in Mexico City!");
```

   Các tọa độ trong GeoPoint tính theo đơn vị vi độ (microdegrees, bằng độ * 1e6). Phương thức khởi tạo OverlayItem nhận thông tin vị trí qua đối tượng GeoPoint, một chuỗi cho tiêu đề và một chuỗi cho phần tóm tắt của đối tượng OverlayItem, theo thứ tự tương ứng.

4. Phần việc còn lại là thêm đối tượng OverlayItem này vào mảng trong thể hiện của HelloItemizedOverlay, và sau đó thêm đối tượng HelloItemizedOverlay vào MapView:

```
itemizedoverlay.addOverlay(overlayitem);
mapOverlays.add(itemizedoverlay);
```

5. Chạy ứng dụng.

   Bạn sẽ thấy như hình sau:

Khi bạn chạm vào đối tượng phủ, bạn sẽ thấy một hộp thoại xuất hiện.

Vì lớp ItemizedOverlay sử dụng một mảng java.util.ArrayList cho tất cả các đối tượng OverlayItem, việc tạo thêm đối tượng phủ rất dễ dàng. Để tạo thêm một đối tượng phủ nữa, trước khi phương thức addOverlay() được gọi, bạn hãy chèn đoạn mã sau:

```
GeoPoint point2 = new GeoPoint(35410000, 139460000);
OverlayItem overlayitem2 = new OverlayItem(point2, "Sekai,
                                    konichiwa!", "I'm in Japan!");
```

Chạy ứng dụng. (Có thể bạn cần di chuyển bản đồ để tìm thấy đối tượng phủ mới.)

3. Phát dữ liệu đa phương tiện

NỘI DUNG BÀI HỌC

- Kiến thức Cơ bản
- Khai báo Cấu hình
- Sử dụng MediaPlayer
  - Bước chuẩn bị không đồng bộ
  - Quản lý Trạng Thái
  - Giải phóng MediaPlayer
- Sử dụng Service với MediaPlayer
  - Chạy không đồng bộ
  - Xử lý các lỗi không đồng bộ
  - Sử dụng Khóa thức
  - Service chạy "nổi"
  - Xử lý tiêu điểm âm thanh
  - Giải phóng tài nguyên
- Xử lý Intent AUDIO_BECOMING_NOISY
- Truy xuất Media từ Content Resolver

CÁC LỚP TRỌNG TÂM

- MediaPlayer
- AudioManager
- SoundPool

BÀI ĐỌC THÊM

- JetPlayer
- Audio Capture (Thu âm)
- Android Supported Media Formats (Các định dạng dữ liệu đa phương tiện được Android hỗ trợ)
- Data Storage (Lưu trữ dữ liệu)

Framework đa phương tiện của Android hỗ trợ phát nhiều loại dữ liệu đa phương tiện phổ biến, vì vậy bạn có thể dễ dàng tích hợp âm thanh, video và ảnh vào ứng dụng của mình. Bạn có thể phát nhạc hoặc video từ các file đa phương tiện nằm trong tài nguyên của ứng dụng (tài nguyên thô), từ các file riêng lẻ trong hệ thống file (file system), hoặc từ một dòng dữ liệu ổn định liên tục (data stream) đến từ kết nối mạng, tất cả đều dùng tới các Media Player API.

Tài liệu này hướng dẫn bạn cách viết một ứng dụng phát dữ liệu đa phương tiện có thể tương tác với người dùng và hệ thống nhằm đạt được hiệu suất cao và trải nghiệm người dùng tốt.

> **Lưu ý:** Bạn chỉ có thể phát dữ liệu âm thanh ra thiết bị đầu ra tiêu chuẩn. Hiện nay, đó là loa của thiết bị di động hoặc tai nghe Bluetooth. Bạn không thể phát file âm thanh trong các cuộc gọi điện thoại.

Kiến thức Cơ bản

Các lớp sau được dùng để phát âm thanh và video trong Android framework:

MediaPlayer

Lớp này là API cơ bản để phát âm thanh và video.

AudioManager

Lớp này quản lý các nguồn âm thanh và đầu ra âm thanh trên một thiết bị.

Khai báo Cấu hình

Trước khi bắt đầu phát triển ứng dụng với MediaPlayer, bạn cần đảm bảo file kê khai chứa các khai báo phù hợp để sử dụng những tính năng liên quan.

Quyền Internet – Nếu bạn dùng MediaPlayer để truyền các nội dung trên mạng dưới dạng các dòng ổn định và liên tục, ứng dụng của bạn phải yêu cầu quyền truy xuất mạng.

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

Quyền Khóa thức (Wake lock) – Nếu ứng dụng của bạn không muốn màn hình tối đi hoặc bộ vi xử lý chuyển sang trạng thái ngủ, hay nếu bạn cần sử dụng phương thức MediaPlayer.setScreenOnWhilePlaying() hoặc MediaPlayer.setWakeMode(), bạn phải yêu cầu quyền sau.

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

Sử dụng MediaPlayer

Một trong những thành phần quan trọng nhất của framework đa phương tiện là lớp MediaPlayer. Một đối tượng của lớp này có thể truy xuất, giải mã và phát cả âm thanh, video mà chỉ cần cài đặt tối thiểu. Lớp này hỗ trợ một số nguồn dữ liệu đa phương tiện khác nhau như:

- Tài nguyên cục bộ (lưu trên thiết bị)
- Các URI nội bộ, như URI bạn lấy từ một Content Resolver
- Các URL bên ngoài (data stream)

Bạn có thể tham khảo danh sách các định dạng dữ liệu đa phương tiện được Android hỗ trợ qua tài liệu Android Supported Media Formats.

Đây là một ví dụ về cách phát âm thanh có trong tài nguyên thô cục bộ (lưu trong thư mục res/raw/ trong ứng dụng):

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);

mediaPlayer.start(); // Không cần gọi prepare(); create() đã làm
                     // việc đó cho bạn
```

Trong trường hợp này, tài nguyên "thô" là một file mà hệ thống không cần phải phân tách theo cách cụ thể nào. Tuy nhiên, nội dung của tài nguyên này không phải là âm thanh định dạng thô (raw). Đó là một file đa phương tiện đã được mã hóa và định dạng theo một trong các định dạng được hỗ trợ.

Sau đây là cách phát dữ liệu đa phương tiện từ một URI nội bộ có sẵn trong hệ thống (ví dụ, như URI bạn lấy từ một Content Resolve):

```
Uri myUri = ....; // khởi tạo Uri ở đây
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

Phát từ một URL từ xa thông qua dòng dữ liệu HTTP (HTTP stream) sẽ như sau:

```
String url = "http://........"; // URL của bạn
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // có thể mất nhiều thời gian! (để lưu dữ liệu
                       // vào bộ đệm (buffering), …)
mediaPlayer.start();
```

> **Lưu ý:** Nếu bạn truyền một URL để truyền một file đa phương tiện trực tuyến dưới dạng dòng dữ liệu ổn định liên tục, file này phải có khả năng tải lũy tiến (từng phần).

> **Chú ý:** Bạn phải bắt hoặc truyền đi ngoại lệ IllegalArgumentException và IOException khi sử dụng setDataSource(), vì file bạn đang trỏ tới có thể không tồn tại.

Bước chuẩn bị không đồng bộ

Về nguyên tắc, việc sử dụng MediaPlayer rất đơn giản. Tuy nhiên, bạn cần lưu tâm một số khía cạnh khác để có thể tích hợp MediaPlayer vào một ứng dụng Android điển hình một cách hợp lý. Ví dụ, lời gọi phương thức prepare() có thể chạy trong khoảng thời gian dài, vì trong đó có thể bao gồm tác vụ truy xuất và giải mã dữ liệu đa phương tiện. Vì vậy, giống như bất cứ phương thức nào có khả năng chạy kéo dài, bạn **không bao giờ nên gọi phương thức đó từ luồng UI (luồng giao diện người dùng) của ứng dụng**. Thao tác như vậy sẽ làm treo giao diện người dùng cho đến khi phương thức trả về, đó là một trải nghiệm người dùng rất tệ và có thể gây lỗi ANR (Ứng dụng không đáp ứng). Kể cả khi bạn nghĩ một tài nguyên nào đó tải nhanh chóng, hãy nhớ bất cứ thao tác nào tốn hơn 1/10 giây để có phản hồi trên giao diện sẽ gây một quãng dừng đáng kể, và khiến người dùng cảm nhận rằng ứng dụng của bạn chậm.

Để tránh làm treo luồng UI, bạn hãy tạo một luồng khác cho bước chuẩn bị MediaPlayer và thông báo với luồng chính khi hoàn thành. Dù bạn có thể tự viết đoạn mã xử lý luồng này, tác vụ này quá phổ biến khi sử dụng MediaPlayer nên framework đã cung cấp sẵn một cách làm tiện lợi với phương thức prepareAsync(). Phương thức này bắt đầu chuẩn bị dữ liệu đa phương tiện dưới dạng chạy ngầm (hay chạy nền) và sẽ trả về ngay lập tức. Khi bước chuẩn bị đã xong, phương thức onPrepared() của MediaPlayer.OnPreparedListener, được cấu hình qua phương thức setOnPreparedListener() sẽ được gọi.

Quản lý Trạng thái

Một khía cạnh nữa cần lưu ý là MediaPlayer hoạt động dựa trên trạng thái. Tức là MediaPlayer có một trạng thái nội tại mà bạn luôn phải chú ý tới khi lập trình, vì một số thao tác chỉ có hiệu lực khi đối tượng đang trong một số trạng thái nhất định. Nếu bạn thực hiện một thao tác không đúng với trạng thái, hệ thống có thể ném ra ngoại lệ hoặc gây ra các hành vi không mong muốn khác.

Tài liệu đặc tả lớp MediaPlayer có một biểu đồ trạng thái hoàn chỉnh, trong đó giải thích rõ phương thức nào khiến MediaPlayer chuyển từ trạng thái này sang trạng thái khác. Ví dụ, khi bạn tạo một MediaPlayer mới, nó ở trạng thái *Idle* (Nghỉ). Tại thời điểm đó, bạn có thể khởi tạo MediaPlayer bằng cách gọi setDataSource() để chuyển sang trạng thái *Initialized* (Đã khởi tạo). Sau đó, bạn chuẩn bị MediaPlayer bằng phương thức prepare() hoặc prepareAsync(). Khi bước chuẩn bị hoàn thành, MediaPlayer sẽ vào trạng thái *Prepared* (Đã chuẩn bị), tức là bạn có thể gọi start() để bắt đầu phát đa phương tiện. Tại thời điểm đó, như minh họa trong biểu đồ, bạn có thể chuyển qua lại giữa các trạng thái như *Started* (Đã bắt đầu), *Paused* (Đã tạm dừng) và *PlaybackCompleted* (Đã phát toàn bộ) bằng cách gọi các phương thức như start(), pause() và seekTo() … Tuy nhiên, khi bạn gọi stop(), lưu ý rằng bạn không thể gọi start() lần nữa cho đến khi bạn thực hiện lại bước chuẩn bị MediaPlayer.

Hãy luôn ghi nhớ biểu đồ trạng thái khi viết mã với đối tượng MediaPlayer vì việc gọi các phương thức không đúng với trạng thái là nguyên nhân gây lỗi thường gặp.

Giải phóng MediaPlayer

Một đối tượng MediaPlayer có thể tiêu tốn nhiều tài nguyên hệ thống giá trị. Vì vậy, bạn nên thường trực chú ý để tránh sử dụng một thể hiện MediaPlayer lâu hơn cần thiết. Khi đã sử dụng xong, bạn nên luôn gọi phương thức release() để đảm bảo bất kỳ tài nguyên hệ thống nào đã cấp phát cho MediaPlayer sẽ được giải phóng đúng cách. Ví dụ, nếu bạn đang dùng MediaPlayer và Activity của bạn nhận lời gọi onStop(), bạn phải giải phóng đối tượng MediaPlayer vì không có lý do gì để giữ đối tượng này khi Activity của bạn không tương tác với người dùng (trừ khi bạn đang phát dữ liệu đa phương tiện ở dạng chạy ngầm, tác vụ đó sẽ được bàn ở phần tiếp theo). Dĩ nhiên, khi Activity của bạn được khôi phục hoặc khởi động lại, bạn cần tạo một đối tượng MediaPlayer mới và thực hiện lại bước chuẩn bị trước khi tiếp tục phát.

Dưới đây là cách giải phóng và hủy đối tượng MediaPlayer:

```
mediaPlayer.release();
mediaPlayer = null;
```

Ví dụ nếu bạn quên giải phóng MediaPlayer khi Activity của bạn bị dừng, nhưng bạn lại tạo một đối tượng mới khi Activity khởi động lại, điều này có thể gây ra vấn đề. Như bạn biết, khi người dùng thay đổi hướng màn hình (hoặc thay đổi cấu hình thiết bị theo một cách nào đó), hệ thống sẽ xử lý bằng cách khởi động lại Activity (theo mặc định), như vậy bạn có thể nhanh chóng sử dụng hết tài nguyên hệ thống khi người dùng xoay đi xoay lại thiết bị, vì mỗi lần hướng thiết bị thay đổi, bạn lại tạo mới một MediaPlayer mà bạn không bao giờ giải phóng. (Để tìm hiểu thêm về việc khởi động lại trong thời gian chạy, bạn hãy xem tài liệu Handling Runtimes Changes (tạm dịch: Xử lý những thay đổi trong thời gian chạy).

Bạn có thể thắc mắc điều gì sẽ xảy ra nếu bạn muốn tiếp tục phát "file đa phương tiện chạy ngầm" kể cả khi người dùng rời Activity, gần giống như cách hoạt động của ứng dụng Music tích hợp sẵn. Trong trường hợp này, cái bạn cần là một đối tượng MediaPlayer được điều khiển bởi một Service được mô tả trong phần "Sử dụng Service với MediaPlayer".

Sử dụng service với MediaPlayer

Nếu bạn muốn phát file đa phương tiện chạy ngầm ngay cả khi ứng dụng của bạn không hiển thị trên màn hình - tức là khi đó người dùng đang tương tác với các ứng dụng khác – bạn phải khởi động một Service và điều khiển thể hiện MediaPlayer từ đó. Bạn cần cẩn thận với cách thiết lập này, vì người dùng và hệ thống có một số yêu cầu nhất định về cách thức tương tác của một ứng dụng chạy service ngầm với phần còn lại của hệ thống. Nếu ứng dụng của bạn không thỏa mãn được những yêu cầu đó, người dùng có thể gặp phải những trải nghiệm xấu. Phần này mô tả các vấn đề chính bạn cần lưu ý và đề ra một số gợi ý cách xử lý chúng.

Chạy không đồng bộ

Trước tiên, giống như Activity, tất cả công việc trong một Service được thực hiện mặc định bởi một luồng duy nhất – trên thực tế, nếu bạn chạy một Activity và một service trong cùng một ứng dụng, chúng mặc định sử dụng cùng một luồng ("luồng chính"). Vì vậy, các service cần xử lý các Intent tới nhanh chóng

và không bao giờ thực hiện các tính toán dài dòng khi phản hồi Intent. Nếu bạn dự kiến xử lý tải nặng hoặc các lời gọi dạng blocking (chặn luồng chính), bạn phải thực hiện các tác vụ đó một cách không đồng bộ: hoặc từ một luồng khác bạn tự định nghĩa, hoặc sử dụng nhiều tiện ích có sẵn của framework cho việc xử lý không đồng bộ.

Ví dụ, khi sử dụng MediaPlayer từ luồng chính, bạn nên gọi phương thức prepareAsync() thay vì prepare(), và kế thừa (implement) giao diện MediaPlayer.OnPreparedListener để được thông báo khi bước chuẩn bị hoàn tất và bạn có thể bắt đầu phát đa phương tiện. Ví dụ:

```
public class MyService extends Service implements
                                    MediaPlayer.OnPreparedListener {
    private static final ACTION_PLAY = "com.example.action.PLAY";
    MediaPlayer mMediaPlayer = null;

    public int onStartCommand(Intent intent, int flags, int startId)
{
        ...
        if (intent.getAction().equals(ACTION_PLAY)) {
            mMediaPlayer = .../ khởi tạo ở đây
            mMediaPlayer.setOnPreparedListener(this);
            mMediaPlayer.prepareAsync(); // chuẩn bị không đồng bộ
                                         // để tránh chặn luồng chính
        }
    }
    /** Được gọi khi MediaPlayer sẵn sàng */
    public void onPrepared(MediaPlayer player) {
        player.start();
    }
}
```

Xử lý các lỗi không đồng bộ

Trong các thao tác đồng bộ, các lỗi thường được báo hiệu bằng một ngoại lệ hoặc một mã lỗi, nhưng khi sử dụng các tài nguyên một cách không đồng độ, bạn nên đảm bảo ứng dụng thông báo lỗi một cách thích hợp. Trong trường hợp MediaPlayer, bạn có thể thực hiện điều đó bằng cách kế thừa giao diện MediaPlayer.OnErrorListener và thiết lập lỗi trong thể hiện MediaPlayer của bạn:

```
public class MyService extends Service implements
                                    MediaPlayer.OnErrorListener {
   MediaPlayer mMediaPlayer;

   public void initMediaPlayer() {
      // ...khởi tạo MediaPlayer ở đây ...

      mMediaPlayer.setOnErrorListener(this);
   }
   @Override

   public boolean onError(MediaPlayer mp, int what,  int extra) {
      // ... phản hồi thích hợp ...
      // MediaPlayer đã chuyển sang trạng thái Error (Lỗi) và
      // phải được khởi tạo lại!
   }
}
```

Điều quan trọng cần nhớ là khi lỗi xảy ra, MediaPlayer chuyển sang trạng thái *Error* (Lỗi) (xem biểu đồ trạng thái hoàn chỉnh trong tài liệu đặc tả lớp MediaPlayer) và bạn phải khởi tạo lại nó trước khi có thể sử dụng tiếp.

Sử dụng Khóa thức

Khi có ứng dụng phát đa phương tiện chạy ngầm, thiết bị vẫn có thể chuyển sang trạng thái ngủ. Vì hệ thống Android tiết kiệm pin khi thiết bị ngủ, thiết bị sẽ tắt bất cứ tính năng nào không cần thiết, bao gồm vi xử lý và WiFi. Tuy vậy, nếu service của bạn đang phát nhạc hoặc truyền nhạc dưới dạng dòng ổn định liên tục, bạn không muốn hệ thống can dự vào hoạt động của mình.

Để đảm bảo service tiếp tục chạy trong những tình huống đó, bạn phải sử dụng "khóa thức". Khóa thức là cách báo hiệu cho hệ thống rằng ứng dụng đang sử dụng một tính năng nào đấy cần chạy ngay cả khi thiết bị ở trạng thái nghỉ.

> **Lưu ý:** Bạn nên luôn dè chừng khi sử dụng khóa thức và chỉ nên giữ chúng trong khoảng thời gian thực sự cần thiết vì chúng làm giảm đáng kể thời lượng pin của thiết bị.

Để đảm bảo CPU tiếp tục chạy khi MediaPlayer đang phát, hãy gọi phương thức setWakeMode() khi khởi tạo đối tượng MediaPlayer. Khi làm vậy, MediaPlayer giữ khóa trong khi phát và giải phóng khóa khi tạm dừng hoặc dừng hẳn.

```
mMediaPlayer = new MediaPlayer();
// ... các thao tác khởi tạo khác ở đây ...
mMediaPlayer.setWakeMode(getApplicationContext(),
                                PowerManager.PARTIAL_WAKE_LOCK);
```

Tuy nhiên, khóa thức trong ví dụ này chỉ đảm bảo CPU tiếp tục hoạt động. Nếu bạn đang truyền dữ liệu đa phương tiện qua mạng dưới dạng dòng ổn định liên tục (data stream) và đang sử dụng Wi-Fi, bạn có thể cần giữ một WifiLock (khóa Wifi) nữa, với khóa này bạn phải yêu cầu và giải phóng thủ công. Vì vậy, khi bạn bắt đầu bước chuẩn bị MediaPlayer với URL từ xa, bạn nên tạo và duy trì khóa Wi-Fi. Ví dụ:

```
WifiLock wifiLock = ((WifiManager) getSystemService(
   Context.WIFI_SERVICE)).createWifiLock(WifiManager.WIFI_MODE_FULL,
                                                        "mylock");

wifiLock.acquire();
```

Khi bạn tạm dừng hoặc dừng dòng dữ liệu đa phương tiện, hoặc khi bạn không còn cần mạng, bạn nên giải phóng khóa:

```
wifiLock.release();
```

Service chạy "nổi"

Các service thường được dùng để thực hiện các tác vụ ngầm như lấy email, đồng bộ dữ liệu, tải nội dung … Trong các trường hợp này, người dùng không biết đến hoạt động của service và có thể không chú ý đến việc các service này bị gián đoạn và khởi động lại sau đó.

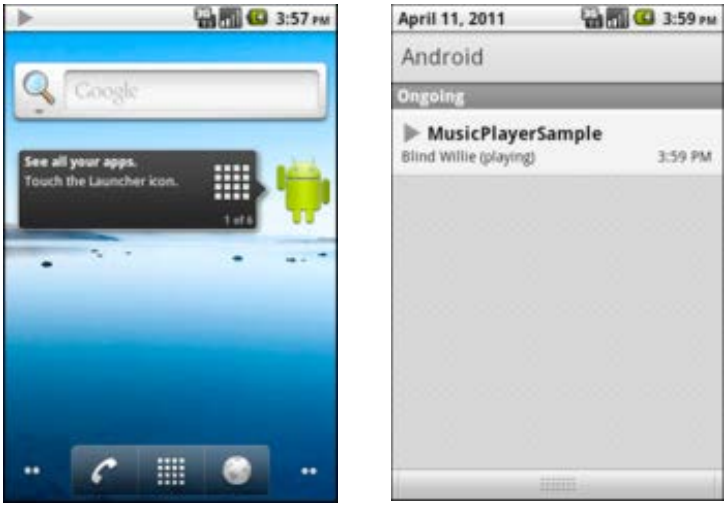
Nhưng hãy xem xét trường hợp một service đang phát nhạc. Rõ ràng đây là một service người dùng chủ động chú ý tới và trải nghiệm sẽ bị ảnh hưởng nặng nề nếu có bất cứ sự gián đoạn nào. Ngoài ra, đây là service có khả năng người dùng sẽ muốn tương tác trong quá trình chạy. Trong trường hợp này, service nên được chạy theo dạng "service chạy nổi" (foreground service). Một service chạy nổi có mức ưu tiên cao hơn trong hệ thống – hệ thống sẽ gần như không bao giờ triệt tiêu service này, vì nó liên quan trực tiếp đến người dùng. Khi chạy nổi, service cũng phải cung cấp một thông báo ở thanh trạng thái để đảm bảo người dùng biết đến service đang chạy và cho phép người dùng mở một Activity để tương tác với service.

Để biến service của bạn thành một service chạy nổi, bạn phải tạo một đối tượng Notification (Thông báo) cho thanh trạng thái và gọi phương thức startForeground() từ Service. Ví dụ:

```
String songName;
// gán tên bài hát vào biến songName
PendingIntent pi = PendingIntent.getActivity(getApplicationContext(), 0,
         new Intent(getApplicationContext(), MainActivity.class),
                                 PendingIntent.FLAG_UPDATE_CURRENT);
Notification notification = new Notification();
notification.tickerText = text;
notification.icon = R.drawable.play0;
notification.flags |= Notification.FLAG_ONGOING_EVENT;
notification.setLatestEventInfo(getApplicationContext(),
                    "MusicPlayerSample","Playing: "+ songName, pi);
startForeground(NOTIFICATION_ID, notification);
```

Khi service đang chạy nổi, đối tượng thông báo bạn định nghĩa sẽ hiển thị ở vùng thông báo của thiết bị. Nếu người dùng chọn thông báo đó, hệ thống sẽ kích hoạt đối tượng PendingIntent bạn đã cung cấp. Trong ví dụ trên, hệ thống sẽ mở một Activity (MainActivity).

Hình 3.1 cho thấy thông báo của bạn hiển thị như thế nào đối với người dùng:



Hình 3.1. Hình chụp thông báo của một service chạy nổi, cho thấy biểu tượng thông báo trong thanh trạng thái (hình bên trái) và thông báo khi mở rộng (hình bên phải).

Bạn chỉ nên giữ trạng thái "service chạy nổi" khi service của bạn đang thực sự thi hành tác vụ nào đó mà người dùng chủ động chú ý tới. Một khi điều đó không còn đúng, bạn nên giải phóng trạng thái đó bằng cách gọi stopForeground():

```
stopForeground(true);
```

Để tìm hiểu thêm, bạn hãy xem tài liệu đặc tả về Service và Status Bar Notifications (Các thông báo trên thanh trạng thái).

Xử lý tiêu điểm âm thanh

Dù chỉ một Activity có thể chạy tại một thời điểm, Android là một môi trường đa tác vụ. Đặc điểm này đặt ra một thử thách đặc thù cho các ứng dụng sử dụng âm thanh, vì chỉ có một đầu ra âm thanh và có thể có nhiều service đa phương tiện cạnh tranh quyền sử dụng. Trước Android 2.2, không có cơ chế nội tại để giải quyết vấn đề này và nhiều khi dẫn tới trải nghiệm không tốt cho người dùng. Ví dụ, khi người dùng đang nghe nhạc và một ứng dụng khác cần thông báo cho người dùng một thông tin rất quan trọng, người dùng có thể không nghe thấy âm thông báo vì tiếng nhạc lớn. Kể từ Android 2.2, nền tảng cung cấp một cách thức để các ứng dụng thương lượng quyền sử dụng đầu ra âm thanh của thiết bị. Cơ chế này được gọi là Tiêu điểm Âm thanh (Audio Focus).

Khi ứng dụng của bạn cần phát âm thanh như nhạc hay thông báo, bạn nên luôn yêu cầu tiêu điểm âm thanh. Một khi có tiêu điểm, ứng dụng có thể thoải mái sử dụng đầu ra âm thanh, nhưng ứng dụng cũng cần luôn lắng nghe sự kiện tiêu điểm thay đổi. Nếu được thông báo đã mất tiêu điểm âm thanh, ứng dụng nên ngay lập tức dừng phát âm thanh hoặc giảm âm lượng xuống mức nhỏ (có một cờ chỉ định cách làm nào thích hợp) và chỉ nên khôi phục việc phát âm lượng lớn sau khi nhận lại được tiêu điểm.

Bản chất cơ chế "Tiêu điểm Âm thanh" là tính hợp tác. Tức là, các ứng dụng được "kỳ vọng" (và rất "khuyến khích") tuân theo những hướng dẫn về tiêu điểm âm thanh, nhưng hệ thống không ép buộc tuân theo các quy tắc đó. Nếu một ứng dụng muốn phát nhạc lớn ngay cả sau khi mất tiêu điểm âm thanh, hệ thống sẽ không làm gì để ngăn chặn. Tuy vậy, có khả năng người dùng sẽ có trải nghiệm xấu và có thể sẽ gỡ ứng dụng có cách xử lý không thích hợp đó.

Để yêu cầu tiêu điểm âm thanh, bạn phải gọi requestAudioFocus() từ AudioManager, như ví dụ dưới đây:

```
AudioManager audioManager =
   (AudioManager) getSystemServiceContext.AUDIO_SERVICE);
int result =
   audioManager.requestAudioFocus(this, AudioManager.STREAM_MUSIC,
                                  AudioManager.AUDIOFOCUS_GAIN);
if (result != AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
   // không lấy được tiêu điểm âm thanh.
}
```

Tham số đầu tiên của phương thức requestAudioFocus() là một đối tượng AudioManager.OnAudioFocusChangeListener, trong đó phương thức onAudioFocusChange() sẽ được gọi bất cứ khi nào có thay đổi tiêu điểm âm

thanh. Vì vậy, bạn cũng cần kế thừa giao diện này trong service và các Activity của bạn. Ví dụ:

```
class MyService extends Service
            implements AudioManager.OnAudioFocusChangeListener {
   // ....
   public void onAudioFocusChange(int focusChange) {
      // Làm gì đó dựa trên thay đổi tiêu điểm âm thanh...
   }
}
```

Tham số focusChange cho bạn biết tiêu điểm âm thanh đã thay đổi như thế nào, và tham số này có thể nhận một trong các giá trị sau (chúng đều là hằng số định nghĩa trong AudioManager):

- AUDIOFOCUS_GAIN: Bạn đã lấy được tiêu điểm âm thanh.
- AUDIOFOCUS_LOSS: Bạn đã mất tiêu điểm âm thanh, có thể trong một thời gian dài. Bạn phải dừng tất cả hoạt động phát âm thanh. Vì bạn không nhận lại được tiêu điểm trong thời gian dài, đây là thời điểm tốt để dọn dẹp càng nhiều tài nguyên càng tốt. Ví dụ, bạn nên giải phóng MediaPlayer.
- AUDIOFOCUS_LOSS_TRANSIENT: Bạn đã tạm thời mất tiêu điểm âm thanh, nhưng bạn sẽ có lại trong thời gian ngắn. Bạn phải dừng tất cả hoạt động phát âm thanh, nhưng bạn có thể giữ các tài nguyên vì có thể bạn sẽ có tiêu điểm trở lại không lâu sau đó.
- AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK: Bạn đã tạm thời mất tiêu điểm âm thanh, nhưng bạn được cho phép tiếp tục phát âm thanh với âm lượng thấp thay vì dừng phát âm thanh hoàn toàn.

Đây là một ví dụ về cách xử lý:

```
public void onAudioFocusChange(int focusChange) {
    switch (focusChange) {
        case AudioManager.AUDIOFOCUS_GAIN:
            // phục hồi việc phát
            if (mMediaPlayer ==null) initMediaPlayer();
            else if (!mMediaPlayer.isPlaying()) mMediaPlayer.start();
                mMediaPlayer.setVolume(1.0f,1.0f);
            break;

        case AudioManager.AUDIOFOCUS_LOSS:
            // Mất tiêu điểm trong khoảng thời gian không xác định:
            // ngừng phát và giải phóng media player


            if (mMediaPlayer.isPlaying()) mMediaPlayer.stop();
               mMediaPlayer.release();
               mMediaPlayer = null;
            break;

        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
            // Mất tiêu điểm trong thời gian ngắn, nhưng chúng ta
            // phải ngừng phát.
            // Chúng ta không giải phóng media player vì có thể việc
            // phát sẽ được tiếp tục trở lại
            if (mMediaPlayer.isPlaying()) mMediaPlayer.pause();
            break;
```

```
        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
            // Mất tiêu điểm trong thời gian ngắn, nhưng được phép
            // tiếp tục phát với âm lượng giảm bớt
            if (mMediaPlayer.isPlaying()) mMediaPlayer.setVolume(0.1f, 0.1f);
            break;
    }
}
```

Cần lưu ý rằng các API về tiêu điểm âm thanh chỉ có ở API cấp 8 (Android 2.2) trở lên, vì vậy nếu muốn hỗ trợ các phiên bản Android trước, bạn nên thực hiện một chiến lược tương thích ngược để có thể sử dụng tính năng này nếu có, và có cách xử trí dự phòng trong trường hợp ngược lại.

Bạn có thể đạt được tương thích ngược bằng cách gọi các phương thức tiêu điểm âm thanh qua kỹ thuật reflection[2] hoặc bằng cách hiện thực hóa tất cả các tính năng về tiêu điểm âm thanh trong một lớp riêng biệt (ví dụ, AudioFocusHelper). Dưới đây là một ví dụ minh họa:

[2] *Là kỹ thuật đọc và sửa đổi cấu trúc và hành vi (cụ thể là các giá trị, siêu dữ liệu, thuộc tính và phương thức) của chương trình trong thời gian chạy. Nguồn tham khảo: http://en.wikipedia.org/wiki/Reflection_(computer_programming).*

```
public class AudioFocusHelper implements
AudioManager.OnAudioFocusChangeListener { AudioManager mAudioManager;

   // các trường khác ở đây, có thể bạn sẽ giữ một tham chiếu tới
   // một giao diện mà qua đó bạn có thể truyền các thay đổi tiêu
   // điểm đến Service của bạn

   public AudioFocusHelper(Context ctx, /* các đối số khác */) {
      mAudioManager = (AudioManager)
                mContext.getSystemService(Context.AUDIO_SERVICE);
      // ...
   }
   public boolean requestFocus() {
      return AudioManager.AUDIOFOCUS_REQUEST_GRANTED ==
         mAudioManager.requestAudioFocus(mContext,
                                    AudioManager.STREAM_MUSIC,
                                    AudioManager.AUDIOFOCUS_GAIN);
   }
```

Bạn có thể tạo một thể hiện của lớp AudioFocusHelper chỉ khi bạn phát hiện thấy hệ thống đang chạy API cấp 8 hoặc cao hơn. Ví dụ:

```
If (android.os.Build.VERSION.SDK_INT >= 8) {
   mAudioFocusHelper =
         new AudioFocusHelper(getApplicationContext(), this);
} else {
   mAudioFocusHelper = null;
}
```

Giải phóng tài nguyên

Như đã đề cập ở trên, một đối tượng MediaPlayer có thể chiếm lượng tài nguyên hệ thống đáng kể, vì vậy bạn chỉ nên giữ đối tượng này trong thời gian cần thiết và nên gọi release() khi sử dụng xong. Việc gọi phương thức dọn dẹp này một cách tường minh thay vì phụ thuộc vào cơ chế dọn rác (garbage collection) của hệ thống là rất quan trọng vì có thể mất một thời gian trước khi trình dọn rác thu hồi đối tượng MediaPlayer và hoạt động đó chỉ phản ứng khi có sức ép về bộ nhớ và không phản ứng với các tài nguyên liên quan tới dữ liệu đa phương tiện. Vì vậy, trong trường hợp bạn đang sử dụng một service, bạn nên luôn ghi đè phương thức onDestroy() để đảm bảo bạn sẽ giải phóng MediaPlayer:

```
public class MyService extends Service {
   MediaPlayer mMediaPlayer;
   // ...

   @Override
   public void onDestroy() {
      if (mMediaPlayer != null) mMediaPlayer.release();
   }
}
```

Bạn cũng nên luôn tìm kiếm cơ hội để giải phóng MediaPlayer, ngoài khả năng giải phóng nó khi service bị triệt tiêu. Ví dụ, nếu bạn nghĩ sẽ không thể phát đa phương tiện trong một khoảng thời gian dài (ví dụ, sau khi mất tiêu điểm âm thanh), chắc chắn bạn nên giải phóng đối tượng MediaPlayer đang có và tạo lại sau. Mặt khác, nếu bạn dự kiến chỉ ngừng phát trong một thời gian ngắn, có thể bạn nên giữ lại đối tượng MediaPlayer để tránh tốn công sức tạo và chuẩn bị lại.

Xử lý Intent AUDIO_BECOMING_NOISY

Nhiều ứng dụng phát âm thanh được viết tốt sẽ ngừng phát ngay khi một sự kiện xảy ra khiến âm thanh trở nên ồn (phát ra loa ngoài). Ví dụ, tình huống này có thể xảy ra khi người dùng đang nghe nhạc qua tai nghe và vô tình tháo tai nghe khỏi thiết bị. Tuy nhiên, hành vi ngừng phát âm thanh không tự động diễn ra. Nếu bạn không hiện thực hóa tính năng này, âm thanh sẽ phát ra loa ngoài của thiết bị, và đó có thể không phải điều người dùng mong muốn.

Bạn có thể đảm bảo ứng dụng ngừng phát nhạc trong các tình huống này bằng cách xử lý Intent ACTION_AUDIO_BECOMING_NOISY, trong đó bạn có thể đăng kí một receiver bằng cách thêm đoạn sau vào file kê khai của bạn:

```
<receiver android:name=".MusicIntentReceiver">
   <intent-filter>
      <action android:name="android.media.AUDIO_BECOMING_NOISY" />
   </intent-filter>
</receiver>
```

Đoạn này đăng ký lớp MusicIntentReceiver làm broadcast receiver dành cho Intent đó. Sau đó bạn định nghĩa lớp này như sau:

```
public class MusicIntentReceiver implements
android.content.BroadcastReceiver {
   @Override
   public void onReceive(Context ctx, Intent intent) {
      if (intent.getAction().equals(
           android.media.AudioManager.ACTION_AUDIO_BECOMING_NOISY)) {
         // ra hiệu cho service của bạn ngừng phát âm thanh
         // (ví dụ, thông qua một Intent)
      }
   }
}
```

Truy xuất Media từ Content Resolver

Một tính năng khác có thể hữu dụng trong một ứng dụng phát đa phương tiện là khả năng truy xuất nhạc mà người dùng có trên thiết bị. Bạn có thể làm điều đó bằng cách truy vấn dữ liệu đa phương tiện ngoài từ ContentResolver:

```
ContentResolver contentResolver = getContentResolver();
Uri uri = android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
Cursor cursor = contentResolver.query(uri, null, null, null, null);
if (cursor == null) {
   // truy vấn thất bại, xử lý lỗi.
} else if (!cursor.moveToFirst()) {
   // không có dữ liệu đa phương tiện nào trên thiết bị
} else {
   int titleColumn =
   cursor.getColumnIndex(android.provider.MediaStore.Audio.Media.TITLE);

   int idColumn =
   cursor.getColumnIndex(android.provider.MediaStore.Audio.Media._ID);
   do {
      long thisId = cursor.getLong(idColumn);
      String thisTitle = cursor.getString(titleColumn);
      // ...xử lý từng bản ghi...
   } while (cursor.moveToNext());
}
```

Để sử dụng đoạn mã này với MediaPlayer, bạn có viết như sau:

```
long id = /* lấy từ một nơi nào đó */;
Uri contentUri = ContentUris.withAppendedId(
   android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, id);

mMediaPlayer = new MediaPlayer();
mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mMediaPlayer.setDataSource(getApplicationContext(), contentUri);

// ...chuẩn bị và bắt đầu phát...
```

4. Tiến trình và Luồng

TỔNG QUAN

- Theo mặc định, mỗi ứng dụng chạy trong tiến trình riêng của nó và tất cả các thành phần của ứng dụng đều chạy trong tiến trình đó.
- Bất cứ thao tác nào của một activity gây chậm, hoặc chặn tiến trình nên được xử lý trong một luồng khác để tránh làm chậm giao diện người dùng.

NỘI DUNG BÀI HỌC

- Tiến trình
  - Vòng đời của Tiến trình
- Luồng (Thread)
  - Luồng Thợ (Worker thread)
  - Các phương thức đảm bảo an toàn cho luồng
- Giao tiếp giữa các tiến trình

Khi một thành phần của ứng dụng khởi động và ứng dụng không có bất cứ thành phần nào khác đang chạy, hệ thống Android khởi động một tiến trình Linux mới cho ứng dụng với một luồng thực thi duy nhất. Mặc định, tất cả các thành phần của cùng một ứng dụng chạy cùng tiến trình và luồng (gọi là luồng "chính"). Nếu một thành phần bắt đầu chạy và đã có sẵn một tiến trình cho ứng dụng chứa thành phần đó (vì một thành phần khác của ứng dụng đã tồn tại), thành phần này sẽ được chạy trong tiến trình đã có và sử dụng cùng luồng thực thi. Tuy nhiên, bạn có thể sắp xếp cho các thành phần khác nhau trong ứng dụng chạy trong những tiến trình riêng biệt, và có thể tạo thêm luồng cho bất cứ tiến trình nào.

Tài liệu này đề cập cách thức hoạt động của các tiến trình và luồng trong một ứng dụng Android.

Tiến trình

Mặc định, tất cả các thành phần trong cùng một ứng dụng chạy chung một tiến trình và hầu hết các ứng dụng không nên thay đổi điều này. Tuy nhiên, nếu bạn

cảm thấy cần kiểm soát việc một thành phần cụ thể thuộc về tiến trình nào, bạn có thể làm điều đó trong file kê khai.

Các mục trong file kê khai tương ứng với mỗi kiểu thành phần - `<activity>`, `<service>`, `<receiver>` và `<provider>` - hỗ trợ thuộc tính `android:process`, thuộc tính này cho phép chỉ định tiến trình trong đó thành phần sẽ chạy. Bạn có thể thiết lập thuộc tính này để mỗi thành phần chạy trong một tiến trình riêng, hoặc để một số thành phần chia sẻ một tiến trình khác biệt với các thành phần khác. Bạn cũng có thể thiết lập thuộc tính android:process để những thành phần của các ứng dụng khác nhau chạy chung một tiến trình – miễn là các ứng dụng có chung Linux user ID và được ký với các chứng thư (certificate) giống nhau.

Mục `<application>` cũng hỗ trợ thuộc tính `android:process` để thiết lập giá trị mặc định cho tất cả các thành phần.

Android có thể quyết định triệt tiêu một tiến trình tại thời điểm nào đó khi bộ nhớ xuống thấp và cần dùng bộ nhớ cho các tiến trình khác phục vụ cấp thiết hơn cho người dùng. Do đó, các thành phần của ứng dụng chạy trong tiến trình bị triệt tiêu cũng bị hủy. Một tiến trình sẽ được khởi động trở lại cho các thành phần đó khi lại có công việc cần đến chúng.

Khi quyết định triệt tiêu tiến trình nào, hệ thống Android đánh giá mức quan trọng tương đối của chúng đối với người dùng. Ví dụ, hệ thống sẵn sàng triệt tiêu một tiến trình chứa các Activity không còn hiển thị trên màn hình hơn so với một tiến trình chứa các Activity đang hiển thị. Vì vậy, quyết định chấm dứt một tiến trình phụ thuộc vào trạng thái các thành phần đang chạy trong tiến trình đó. Các quy tắc dùng để quyết định chấm dứt tiến trình nào được trình bày dưới đây.

Vòng đời của Tiến trình

Hệ thống Android cố gắng duy trì một tiến trình ứng dụng lâu nhất có thể, nhưng cuối cùng vẫn cần loại bỏ các tiến trình cũ để thu hồi bộ nhớ cho các tiến trình mới hoặc quan trọng hơn. Hệ thống quyết định giữ hay loại bỏ tiến trình nào dựa trên các thành phần chạy trong mỗi tiến trình và trạng thái của các thành phần đó, hệ thống sẽ đặt các tiến trình vào một “hệ phân cấp độ quan trọng”. Các tiến trình có cấp độ quan trọng thấp nhất bị chấm dứt đầu tiên, sau đó là các tiến trình ở cấp độ quan trọng thấp tiếp theo, cứ như vậy cho đến khi thu hồi đủ tài nguyên cần thiết cho hệ thống.

Có năm cấp độ trong hệ phân cấp độ quan trọng. Danh sách sau liệt kê các kiểu tiến trình khác nhau theo thứ tự độ quan trọng (tiến trình đầu tiên là *quan trọng nhất* và bị *chấm dứt* cuối cùng):

a. Tiến trình chạy nổi (foreground process)

Một tiến trình cần cho tương tác hiện thời của người dùng. Một tiến trình được coi là chạy nổi nếu thỏa mãn bất cứ điều kiện sau:

- Chứa một Activity mà người dùng đang tương tác (phương thức onResume() của Activity đã được gọi).
- Chứa một Service liên kết với Activity mà người dùng đang tương tác.
- Chứa một Service đang chạy "nổi" – service đã gọi startForeground().
- Chứa một Service đang chạy một trong các phương thức callback liên quan đến vòng đời của nó (onCreate(), onStart() hoặc onDestroy()).
- Chứa một BroadcastReceiver đang chạy phương thức onReceive() của nó.

Nhìn chung, chỉ tồn tại một vài tiến trình chạy nổi tại một thời điểm nhất định. Chúng chỉ bị chấm dứt khi không còn phương án nào khác – nếu bộ nhớ xuống thấp đến mức hệ thống không thể tiếp tục hoạt động. Tại thời điểm đó, thiết bị đã đạt tới tình trạng phải dùng bộ nhớ bổ trợ (paging), vì vậy hệ thống cần triệt tiêu một số tiến trình chạy nổi để giữ khả năng đáp ứng cho giao diện người dùng.

b. Tiến trình thấy được (visible process)

Một tiến trình không có thành phần chạy nổi nào, nhưng vẫn có thể tác động đến cái người dùng thấy trên màn hình. Một tiến trình được coi là tiến trình thấy được nếu thỏa mãn một trong hai điều kiện sau:

- Chứa một Activity không phải là Activity chạy nổi nhưng vẫn hiển thị đối với người dùng (phương thức onPause() của nó đã bị gọi). Tình huống này có thể xảy ra, ví dụ khi một Activity chạy nổi được khởi chạy từ một hộp hội thoại, Activity trước đó vẫn có thể được nhìn thấy bên dưới.
- Chứa một Service liên kết với một Activity thấy được (hoặc chạy nổi).

Một tiến trình thấy được được coi là cực kỳ quan trọng và sẽ không bị triệt tiêu trừ khi việc đó cần thiết để duy trì tất cả các tiến trình chạy nổi.

c. Tiến trình Service (service process)

Một tiến trình đang chạy một service mà đã được khởi chạy với phương thức startService() và không rơi vào một trong hai cấp độ cao hơn ở trên. Dù các tiến trình service không liên quan trực tiếp với những thứ mà người dùng nhìn thấy,

chúng thường làm những việc người dùng quan tâm đến (như phát nhạc ở nền hoặc tải dữ liệu qua mạng), vì vậy hệ thống duy trì chúng trừ khi không có đủ bộ nhớ để giữ chúng cùng với tất cả các tiến trình chạy nổi hoặc thấy được.

d. Tiến trình chạy ngầm (background process)

Một tiến trình chứa một Activity không hiển thị đối với người dùng (phương thức onStop() của Activity đã bị gọi). Các tiến trình này không có tác động trực tiếp đối với trải nghiệm người dùng và hệ thống có thể chấm dứt chúng bất cứ lúc nào để thu hồi bộ nhớ cho một tiến trình chạy nổi, thấy được, hoặc service. Thường có nhiều tiến trình chạy ngầm, vì vậy hệ thống giữ một danh sách tiến trình ít sử dụng gần đây nhất (LRU) để đảm bảo tiến trình có Activity hiển thị gần thời điểm hiện tại nhất với người dùng sẽ là tiến trình cuối cùng bị chấm dứt. Nếu phần thân của các phương thức trong vòng đời của Activity được viết hợp lý, và lưu trạng thái hiện tại, việc chấm dứt tiến trình của Activity sẽ không gây tác động thấy được đối với trải nghiệm người dùng, vì khi người dùng quay lại Activity, Activity có thể khôi phục toàn bộ trạng thái hiển thị của nó. Bạn có thể đọc tài liệu về Activity để hiểu thêm cơ chế lưu và khôi phục trạng thái.

e. Tiến trình rỗng (empty process)

Một tiến trình không chứa bất cứ thành phần ứng dụng nào đang hoạt động. Lý do duy nhất để duy trì loại tiến trình này là để lưu trữ đệm, nhằm cải thiện thời gian khởi động lần tiếp theo của một thành phần cần chạy trong tiến trình. Hệ thống thường loại bỏ các tiến trình này để cân bằng toàn bộ tài nguyên hệ thống giữa các bộ nhớ đệm của tiến trình và các bộ nhớ đệm của nhân hệ điều hành (kernel) bên dưới.

Android xếp hạng một tiến trình ở cấp độ cao nhất mà nó có thể đạt được dựa trên độ quan trọng của các thành phần đang hoạt động trong tiến trình đó. Ví dụ, nếu một tiến trình chứa một service và một Activity thấy được, tiến trình được xếp hạng là một tiến trình thấy được, không phải một tiến trình service.

Ngoài ra, cấp độ của một tiến trình có thể tăng lên nhờ các tiến trình khác phụ thuộc vào nó – một tiến trình đang phục vụ một tiến trình khác không bao giờ bị xếp hạng thấp hơn tiến trình nó đang phục vụ. Ví dụ, nếu một Content Provider (Nguồn cung cấp Nội dung) trong tiến trình A đang phục vụ một chương trình trong tiến trình B, hoặc nếu một service trong tiến trình A liên kết với một thành phần trong tiến trình B, tiến trình A luôn được coi tối thiểu quan trọng bằng tiến trình B.

Vì một tiến trình đang chạy một service được xếp hạng cao hơn một tiến trình có Activity chạy ngầm, nên nếu một Activity cần chạy một tác vụ tốn thời gian bạn hãy sử dụng một Service cho tác vụ đó thay vì đơn giản tạo một luồng thợ (worker thread) – đặc biệt khi tác vụ đó có khả năng sẽ kéo dài lâu hơn bản thân

Activity. Ví dụ, một Activity cần đăng tải một bức ảnh lên một website nên chạy một service để thực hiện việc đăng tải, nhờ đó việc đăng tải có thể tiếp tục chạy ngầm ngay cả khi người dùng rời khỏi Activity. Sử dụng một service đảm bảo rằng tác vụ đó ít nhất sẽ có mức ưu tiên của "tiến trình service" bất kể điều gì xảy ra với Activity. Đây cũng là lý do các broadcast receiver nên sử dụng service thay vì đơn giản đẩy các tác vụ tốn thời gian vào một luồng.

Luồng

Khi một ứng dụng khởi động, hệ thống tạo một luồng thực thi cho ứng dụng gọi là "luồng chính". Luồng này rất quan trọng vì nó đảm nhiệm việc gửi sự kiện đến các widget tương ứng trên Giao diện Người dùng, bao gồm các sự kiện vẽ đồ họa. Cũng trong luồng này ứng dụng của bạn tương tác với các thành phần từ bộ công cụ Android UI (Android UI toolkit - các thành phần từ gói android.widget và android.view). Do đó, đôi khi luồng chính cũng được gọi là luồng UI.

Hệ thống *không* tạo luồng riêng cho mỗi thể hiện của một thành phần. Tất cả các thành phần chạy trong cùng tiến trình được khởi tạo trong luồng UI, và các lời gọi hệ thống tới mỗi thành phần được gửi đi từ luồng đó. Do đó, các phương thức xử lý các sự kiện callback của hệ thống (như onKeyDown() để thông báo hành động của người dùng hoặc một phương thức callback liên quan đến vòng đời đối tượng) luôn chạy trong luồng UI của tiến trình.

Ví dụ, khi người dùng chạm vào một nút bấm (button) trên màn hình, luồng UI trong ứng dụng gửi sự kiện chạm tới widget, sau đó widget tự thiết lập trạng thái đã được nhấn và gửi một yêu cầu hủy hiệu lực (invalidate) tới hàng đợi sự kiện. Luồng UI lấy sự kiện khỏi hàng đợi và thông báo cho widget vẽ lại chính nó.

Khi ứng dụng thực hiện tác vụ cần xử lý đặc biệt sau sự kiện tương tác của người dùng, mô hình đơn luồng này có thể bộc lộ điểm yếu về hiệu suất trừ khi bạn viết ứng dụng hợp lý. Cụ thể, nếu mọi thứ xảy ra trong luồng UI, việc thực hiện các tác vụ kéo dài như truy cập mạng hoặc truy vấn cơ sở dữ liệu sẽ chặn toàn bộ giao diện người dùng. Khi luồng bị chặn, các sự kiện sẽ không được gửi đi, bao gồm các sự kiện vẽ. Từ góc độ người dùng, ứng dụng có vẻ đang treo. Tệ hơn, nếu luồng UI bị chặn lâu hơn một vài giây (hiện nay khoảng 5 giây), người dùng sẽ nhìn thấy hộp hội thoại tai tiếng "ứng dụng không đáp ứng" (ANR). Khi đó người dùng có thể quyết định tắt ứng dụng của bạn và gỡ nó nếu họ không hài lòng.

Ngoài ra, bộ công cụ Android UI *không* đảm bảo an toàn cho luồng. Vì vậy, bạn không được điều khiển các thành phần UI từ một luồng thợ – bạn phải thực

hiện tất cả việc điều khiển giao diện người dùng từ luồng UI. Do đó, có hai quy tắc đơn giản đối với mô hình đơn luồng của Android:

1. Không chặn luồng UI
2. Không truy cập bộ công cụ Android UI từ ngoài luồng UI.

Luồng Thợ

Vì với mô hình đơn luồng mô tả ở trên, điểm thiết yếu đối với khả năng đáp ứng của Giao diện Người dùng trong ứng dụng là bạn không được chặn luồng UI. Nếu bạn cần thực hiện các thao tác không xảy ra tức thời, bạn nên đảm bảo chạy chúng trong các luồng riêng (luồng "chạy ngầm" hay luồng "thợ")

Ví dụ, dưới đây là đoạn mã của một listener xử lý sự kiện nhấn nút (click) có nhiệm vụ tải một bức ảnh từ một luồng riêng và hiển thị nó trong ImageView:

```
public void onClick(View v) {
    new Thread (new Runnable() {
        public void run() {
          Bitmap b =
              loadImageFromNetwork("http://example.com/image.png");
          mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Thoáng nhìn, đoạn mã có vẻ chạy tốt vì đã tạo một luồng mới để xử lý thao tác về mạng. Tuy nhiên, đoạn mã trên vi phạm quy tắc thứ hai về mô hình đơn luồng: *không truy cập bộ công cụ Android UI từ bên ngoài luồng UI* – đoạn mã sửa đổi đối tượng ImageView từ luồng thợ thay vì luồng UI. Điều này có thể dẫn tới tình huống đối tượng bị null hoặc những hành vi không lường trước được, và sẽ rất khó khăn và tốn thời gian để điều tra.

Để xử lý vấn đề này, Android cung cấp một số cách để truy cập luồng UI từ các luồng khác. Đây là danh sách các phương thức có thể giúp bạn:

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable, long)

Ví dụ, bạn có thể sửa đoạn mã trên bằng cách sử dụng phương thức View.post(Runnable):

```
public void onClick(View v) {
   new Thread(new Runnable() {
      public void run() {
         final Bitmap bitmap =
             loadImageFromNetwork("http://example.com/image.png");
         mImageView.post (new Runnable() {
            public void run() {
               mImageView.setImageBitmap(bitmap);
            }
         });
      }
   }).start();
}
```

Đoạn mã này đảm bảo an toàn về luồng: thao tác mạng được thực hiện từ một luồng riêng trong khi đối tượng ImageView được điều khiển từ luồng UI.

Tuy vậy, khi độ phức tạp của tác vụ tăng lên, kiểu viết mã này có thể trở nên rắc rối và khó bảo trì. Để xử lý các tác vụ phức tạp hơn từ luồng thợ, bạn có thể cân nhắc dùng Handler trong luồng thợ để xử lý các thông điệp đến từ luồng UI. Tuy vậy, có thể giải pháp tốt nhất là kế thừa lớp AsyncTask vì lớp này đơn giản hóa việc thực thi các tác vụ từ luồng thợ mà cần tương tác với giao diện người dùng.

Sử dụng AsyncTask

AsyncTask cho phép bạn thực hiện tác vụ không đồng bộ trên giao diện người dùng. Nó thực hiện các thao tác gây chặn luồng trong một luồng thợ và sau đó đẩy kết quả lên luồng UI mà không cần bạn tự xử lý luồng hay Handler.

Để sử dụng AsyncTask, bạn phải kế thừa lớp AsyncTask và viết mã phương thức callback doInBackground() để chạy trong một bộ trữ (pool) luồng chạy ngầm. Để cập nhật giao diện người dùng, bạn cần viết mã cho phương thức onPostExecute() trong đó xử lý kết quả từ doInBackground() và chạy trong luồng UI, vì vậy bạn có thể cập nhật giao diện người dùng một cách an toàn. Lúc này bạn có thể chạy tác vụ bằng cách gọi execute() từ luồng UI.

Ví dụ, bạn có thể cài đặt ví dụ trước bằng AsyncTask như sau:

```
public void onClick(View v) {
   new DownloadImageTask().execute("http://example.com/image.png");}
private class DownloadImageTask extends AsyncTask<String, Void,
Bitmap> {
   /** Hệ thống gọi phương thức này để thực hiện công việc trong
     * một luồng thợ, truyền vào các tham số được cung cấp từ lời gọi
     * AsyncTask.execute()*/
   protected Bitmap doInBackground(String... urls) {
      return loadImageFromNetwork(urls[0]);
   }
   /** Hệ thống gọi phương thức này để thực hiện công việc trong
     * luồng UI và truyền vào kết quả của doInBackground() */

    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Giờ giao diện người dùng được đảm bảo và đoạn mã đơn giản hơn vì tách biệt phần việc nào nên thực hiện trong luồng thợ và phần việc nào nên được thực hiện trong luồng UI.

Bạn nên đọc tài liệu tham khảo về AsyncTask để hiểu hết cách sử dụng lớp này; sau đây là phần tóm tắt cách thức hoạt động của nó:

- Bạn có thể chỉ định kiểu tham số, các giá trị tiến độ thực hiện và giá trị cuối cùng của tác vụ theo các kiểu Generic.
- Phương thức doInBackground() tự động chạy trong một luồng thợ.
- onPreExecute(), onPostExecute() và onProgressUpdate() được gọi trong luồng UI.
- Giá trị trả về bởi doInBackground() được gửi tới onPostExecute().
- Bạn có thể gọi publishProgress() bất cứ lúc nào trong doInBackground() để chạy phương thức onProgressUpdate() trong luồng UI.
- Bạn có thể hủy tác vụ bất cứ lúc nào, từ bất cứ luồng nào.

> **Chú ý:** Một vấn đề khác bạn có thể gặp phải khi sử dụng luồng thợ là sự khởi động lại bất thường trong Activity của bạn vì sự thay đổi cấu hình lúc chạy (ví dụ khi người dùng thay đổi hướng màn hình), và hệ quả là luồng thợ của bạn bị hủy. Để tìm hiểu cách duy trì tác vụ trong khi xảy ra sự khởi động lại như vậy, và cách hủy tác vụ một cách hợp lý khi Activity bị hủy, bạn hãy xem mã nguồn của ứng dụng mẫu Shelves.

Các phương thức đảm bảo an toàn cho luồng

Trong một số tình huống, các phương thức bạn viết có thể được gọi từ nhiều hơn một luồng, và vì vậy phải được viết theo cách đảm bảo an toàn về luồng.

Đặc điểm này hầu như đúng đối với những phương thức được gọi từ xa – ví dụ như các phương thức trong một service liên kết (bound service). Khi lời gọi tới một phương thức của một đối tượng IBinder xuất phát trong cùng tiến trình của đối tượng IBinder, phương thức sẽ được chạy trong luồng của đối tượng gọi. Tuy nhiên, khi lời gọi xuất phát từ tiến trình khác, phương thức sẽ được chạy trong một luồng chọn từ bộ trữ luồng mà hệ thống duy trì trong cùng tiến trình của đối tượng IBinder (phương thức không được chạy trong luồng UI của tiến trình). Ví dụ, dù phương thức onBind() của một service được gọi từ luồng UI của tiến trình chứa service, các phương thức trong đối tượng mà phương thức onBind() trả về (ví dụ, một lớp con kế thừa các phương thức RPC) sẽ được gọi từ các luồng trong bộ trữ. Vì một service có thể có hơn một chương trình client (khách), hơn một luồng từ bộ trữ có thể gọi cùng một phương thức của đối tượng IBinder tại cùng một thời điểm. Vì vậy, các phương thức của đối tượng IBinder phải cung cấp cách thức đảm bảo an toàn về luồng.

Tương tự, một Content Provider có thể nhận yêu cầu lấy dữ liệu đến từ các tiến trình khác. Dù lớp ContentResolver và lớp ContentProvider ẩn đi chi tiết cách quản lý sự giao tiếp giữa các tiến trình, các phương thức của đối tượng ContentProvider trả lời các yêu cầu bên ngoài - các phương thức query(), insert(),delete(), update() và getType() - được gọi từ một bộ trữ luồng trong tiến trình của Content Provider, không phải từ luồng UI của tiến trình. Vì các phương thức này có thể được gọi từ nhiều luồng tại cùng thời điểm, chúng cũng phải cung cấp cách thức để đảm bảo an toàn về luồng.

Giao tiếp giữa các tiến trình

Android cung cấp một cơ chế cho việc giao tiếp giữa các tiến trình (IPC – Interprocess communication) bằng lời gọi thủ tục từ xa (RPC), trong đó một phương thức được gọi bởi một Activity hoặc một thành phần ứng dụng khác, nhưng được chạy từ xa (trong tiến trình khác) và kết quả được trả về cho đối tượng gọi. Cơ chế này bao hàm việc phân tách lời gọi phương thức và dữ liệu của nó tới mức độ hệ điều hành có thể hiểu được, truyền thông tin đó từ tiến trình cục bộ và không gian địa chỉ cục bộ tới tiến trình từ xa và không gian địa chỉ từ xa, sau đó lắp ráp lại và tiếp tục thi hành lời gọi ở đó. Sau đó, các giá trị trả về được truyền ngược trở lại. Android cung cấp tất cả đoạn mã để thực hiện

các giao dịch IPC này, nhờ đó bạn có thể tập trung vào việc định nghĩa và viết mã giao diện lập trình RPC.

Để thực hiện giao tiếp giữa các tiến trình, ứng dụng của bạn phải liên kết với một service bằng cách sử dụng bindService(). Để tìm hiểu thêm, bạn hãy xem hướng dẫn lập trình Service.

5. Tổng quan về Hoạt hình và Đồ Họa

Android cung cấp nhiều API mạnh mẽ hỗ trợ việc tạo hoạt hình cho các thành phần UI và vẽ đồ họa 2D và 3D thủ công. Các phần sau cung cấp thông tin tổng quan về những API này và khả năng có sẵn của hệ thống, đồng thời giúp bạn quyết định cách thức nào phù hợp nhất với nhu cầu của bạn.

Hoạt hình

Framework Android cung cấp hai hệ thống hoạt hình: property animation có từ Android 3.0 và view animation. Cả hai hệ thống hoạt hình này đều hoạt động tốt, nhưng hệ thống property animation nhìn chung được ưa chuộng hơn vì linh hoạt và nhiều tính năng hơn. Ngoài hai hệ thống này, bạn có thể sử dụng Drawable animation với khả năng nạp các tài nguyên dạng Drawable và hiển thị chúng theo từng khung hình (frame) nối tiếp nhau.

Property animation

Xuất hiện kể từ phiên bản Android 3.0 (API cấp 11), hệ thống property animation cho phép bạn tạo hoạt hình trên các thuộc tính của bất kỳ đối tượng nào, bao gồm những đối tượng không được kết xuất (render) ra màn hình. Hệ thống này có tính mở rộng và còn cho phép bạn tạo hoạt hình trên các thuộc tính của những kiểu tùy chỉnh.

View animation

View animation là hệ thống cũ hơn và chỉ có thể dùng cho các View. Hệ thống này khá dễ sử dụng và cung cấp đủ tính năng để đáp ứng nhu cầu của nhiều ứng dụng.

Drawable animation

Drawable animation liên quan đến việc hiển thị các tài nguyên Drawable nối tiếp nhau như một cuộn phim. Phương pháp tạo hoạt hình này sẽ hữu ích nếu bạn muốn tạo hoạt hình cho những thứ dễ biểu diễn với tài nguyên Drawable, chẳng hạn như một chuỗi hình ảnh.

Đồ họa 2D và 3D

Khi viết một ứng dụng, một yêu cầu quan trọng là dự tính chính xác các yêu cầu về đồ họa. Các tác vụ đồ họa khác nhau có thể được hoàn thành tốt nhất với những kỹ thuật khác nhau. Ví dụ, viết mã đồ họa và hoạt hình cho một ứng dụng tương đối tĩnh sẽ khác rất nhiều so với đồ họa và hoạt hình cho một ứng dụng trò chơi tương tác. Sau đây, chúng ta sẽ thảo luận về một số phương án vẽ đồ họa trên Android và những tác vụ phù hợp nhất với chúng.

Canvas và Drawable

Android cung cấp một bộ các View với các tính năng chung cho nhiều kiểu giao diện người dùng. Bạn cũng có thể kế thừa các View này để thay đổi hình thức hoặc hành vi của chúng. Ngoài ra, bạn có thể tùy chỉnh việc kết xuất đồ họa 2D bằng nhiều phương thức vẽ (drawing method) phong phú có trong lớp Canvas hoặc tạo các đối tượng Drawable cho những thứ như nút bấm với bề mặt có họa tiết (textured button) hoặc hoạt hình chạy theo từng khung hình (frame-by-frame animation)[3].

Tăng tốc đồ họa nhờ Phần cứng (Hardware Acceleration)

Bắt đầu từ Android 3.0, bạn có thể tăng tốc đồ họa nhờ phần cứng cho hầu hết các hình vẽ tạo bởi Canvas API để tăng hiệu suất hơn nữa.

OpenGL

Android hỗ trợ OpenGL ES 1.0 và 2.0 với các API trong framework Android cũng như với bộ công cụ phát triển thuần - Native Development Kit (NDK). Sử dụng các API của Android sẽ phù hợp khi bạn muốn thêm một vài cải tiến đồ họa cho ứng dụng mà Canvas API không hỗ trợ, hoặc nếu bạn muốn đạt sự độc lập về nền tảng và không yêu cầu hiệu suất cao. Việc sử dụng API của framework gây tổn thất về hiệu suất so với NDK, vì vậy với nhiều ứng dụng nặng về đồ họa như các trò chơi sử dụng NDK có lợi hơn. Một lưu ý quan trọng là bạn vẫn có thể đạt được hiệu suất khá tốt với bộ API của Android. Ví dụ, ứng dụng Google Body được phát triển hoàn toàn bằng API của framework Android. OpenGL với NDK cũng hữu ích nếu bạn có nhiều mã mà bạn muốn chuyển đổi sang Android. Để tìm hiểu thêm về cách sử dụng NDK, bạn có thể đọc thêm tài liệu trong thư mục docs/ của gói NDK.

Property Animation

NỘI DUNG BÀI HỌC

- Cách thức hoạt động của Property Animation
- Tạo hoạt hình với ValueAnimator
- Tạo hoạt hình với ObjectAnimator
- Dàn dựng nhiều hoạt hình với AnimatorSet
- Các listener cho Hoạt hình
- Sử dụng TypeEvaluator

[3] Tạo hoạt hình bằng cách hiển thị tuần tự một chuỗi các hình ảnh thông qua việc sử dụng một đối tượng AnimationDrawable.

- Sử dụng các Interpolator
- Chỉ định Khung hình chính
- Tạo hoạt hình cho các thay đổi về Layout trên ViewGroup
- Tạo hoạt hình cho View
  - o ViewPropertyAnimator
- Khai báo hoạt hình bằng XML

CÁC LỚP TRỌNG TÂM

- ValueAnimator
- ObjectAnimator
- TypeEvaluator

ỨNG DỤNG MẪU

- API Demos

Hệ thống property animation là một framework mạnh mẽ cho phép bạn tạo hoạt hình trên hầu hết mọi thứ. Bạn có thể định nghĩa một hoạt hình để thay đổi bất kỳ thuộc tính nào của một đối tượng theo thời gian, bất kể đối tượng có được vẽ lên màn hình hay không. Một property animation thay đổi giá trị của một thuộc tính (một trường trong một đối tượng) trong một khoảng thời gian cho trước. Để tạo hoạt hình cho đối tượng nào đó, bạn chỉ ra thuộc tính muốn biến đổi (ví dụ như vị trí của đối tượng trên màn hình), thời lượng hoạt hình diễn ra và những giá trị trong khoảng hoạt hình thay đổi tới.

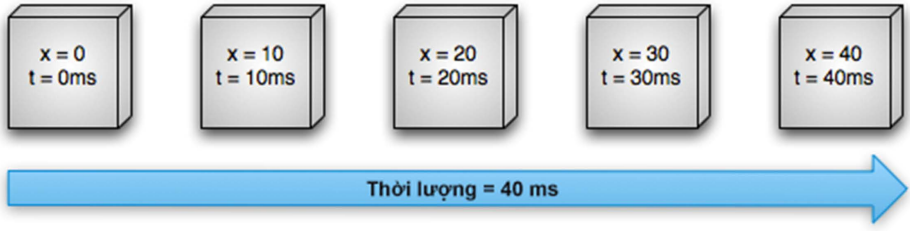
Hệ thống property animation cho phép bạn định nghĩa các đặc tính sau của một hoạt hình:

- Thời lượng: Bạn có thể chỉ định thời lượng của một hoạt hình. Thời lượng mặc định là 300 ms.
- Phép nội suy thời gian: Bạn có thể chỉ định cách tính toán các giá trị của thuộc tính dưới dạng một hàm số của thời gian mà hoạt hình đã trôi qua.
- Số lần lặp và cách thức lặp lại: Bạn có thể chỉ định việc liệu một hoạt hình có lặp lại hay không khi nó chạy hết một khoảng thời gian và số lần lặp lại hoạt hình đó. Bạn cũng có thể chỉ định bạn muốn hoạt hình chạy ngược lại hay không. Thiết đặt chạy ngược (reverse) sẽ phát hoạt hình theo chiều xuôi sau đó lặp lại theo chiều ngược cho đến khi hết số lần lặp lại.
- AnimatorSet: Bạn có thể nhóm các hoạt hình thành các bộ lôgic để chạy cùng nhau hoặc tuần tự hoặc sau các khoảng trễ cho trước.

- Độ trễ làm tươi khung hình: Bạn có thể chỉ định tần suất làm tươi các khung hình của hoạt hình. Giá trị mặc định là 10 ms nhưng tốc độ làm tươi trong ứng dụng của bạn cuối cùng phụ thuộc vào tải tổng thể của hệ thống và việc hệ thống có thể đáp ứng bộ hẹn giờ bên dưới nhanh đến đâu.

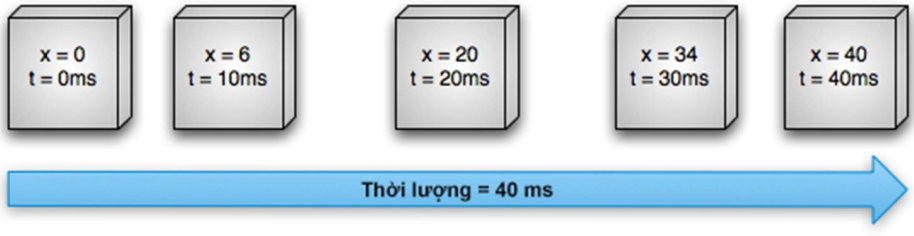
Cách thức hoạt động của Property Animation

Trước tiên, bạn hãy xem một hoạt hình hoạt động như thế nào với ví dụ đơn giản sau. Hình 5.1 mô tả một đối tượng giả định được tạo hoạt hình với thuộc tính x đại diện vị trí theo chiều ngang của đối tượng trên màn hình. Thời lượng hoạt hình được đặt là 40 ms và quãng đường di chuyển là 40 pixel. Cứ 10 ms, tức là bằng tốc độ làm tươi khung hình mặc định, đối tượng di chuyển theo chiều ngang 10 pixel. Đến hết 40 ms, hoạt hình dừng và đối tượng dừng ở vị trí 40 theo chiều ngang. Đây là ví dụ về một hoạt hình với phép nội suy tuyến tính, tức là đối tượng di chuyển với tốc độ không đổi.



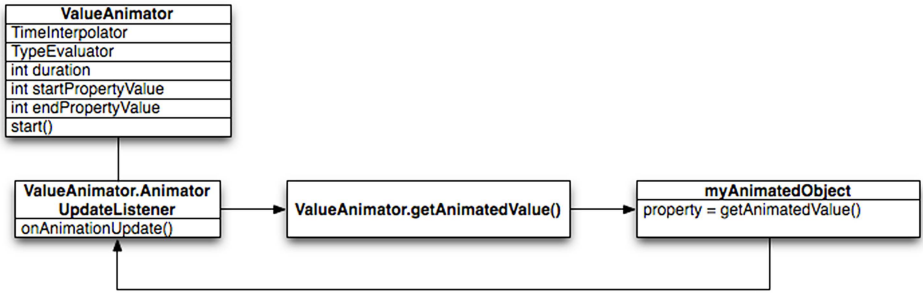
Hình 5.1. Ví dụ về một hoạt hình tuyến tính.

Bạn cũng có thể chỉ định phép nội suy phi tuyến tính cho các hoạt hình. Hình 5.2 minh hoạt một đối tượng giả định tăng tốc ở đầu hoạt hình và giảm tốc ở cuối hoạt hình. Đối tượng vẫn di chuyển 40 pixel trong 40 ms, nhưng theo cách phi tuyến tính. Lúc đầu, chuyển động tăng tốc lên tới điểm giữa, sau đó giảm tốc từ điểm giữa cho đến hết hoạt hình. Như trong Hình 5.2, quãng đường di chuyển ở đoạn đầu và đoạn cuối của hoạt hình ít hơn đoạn giữa.



Hình 5.2. Ví dụ về một hoạt hình phi tuyến tính

Hãy cùng xem xét kỹ lưỡng cách các thành phần quan trọng trong hệ thống property animation tính toán các hoạt hình giống những ví dụ ở trên. Hình 5.3 mô tả cách phối hợp hoạt động của các lớp chính.



Hình 5.3. Cách tính toán của các hoạt hình.

Đối tượng ValueAnimator theo dõi việc tính toán thời gian của hoạt hình như hoạt hình đã chạy bao lâu và giá trị hiện tại của thuộc tính đang được tạo hoạt hình.

ValueAnimator đóng gói một đối tượng TimeInterpolator có nhiệm vụ định nghĩa phép nội suy hoạt hình, và một đối tượng TypeEvaluator có nhiệm vụ định nghĩa cách tính các giá trị cho thuộc tính đang được tạo hoạt hình. Ví dụ, trong Hình 5.2, TimeInterpolator đã dùng là AccelerateDecelerateInterpolator và TypeEvaluator đã dùng là IntEvaluator.

Để tạo một hoạt hình, trước tiên bạn tạo một đối tượng ValueAnimator và đặt giá trị bắt đầu (startPropertyValue) và kết thúc (endPropertyValue) cho thuộc tính bạn muốn tạo hoạt hình cùng với thời lượng của hoạt hình. Khi bạn gọi phương thức start(), hoạt hình sẽ bắt đầu chạy. Trong suốt thời gian chạy của hoạt hình, đối tượng ValueAnimator tính toán *tỷ lệ thời gian chạy* (elapsed fraction) giữa 0 và 1, dựa trên tổng thời lượng của hoạt hình và thời lượng đã trôi qua. Tỷ lệ thời gian chạy biểu diễn tỷ lệ phần trăm thời gian mà hoạt hình đã hoàn thành, 0 nghĩa là 0% và 1 nghĩa là 100%. Ví dụ, trong Hình 5.1, tỷ lệ thời gian chạy tại t = 10 ms là 0.25 vì tổng thời lượng t = 40 ms.

Sau khi tính xong tỷ lệ thời gian chạy, đối tượng ValueAnimator gọi đối tượng TimeInterpolator hiện được thiết đặt để tính *tỷ lệ nội suy* (interpolated fraction). Tỷ lệ nội suy ánh xạ tỷ lệ thời gian chạy với một tỷ lệ mới có tính tới phép nội suy thời gian hiện được thiết lập. Ví dụ, trong Hình 5.2, vì hoạt hình tăng tốc từ từ, tại thời điểm t = 10 ms, tỷ lệ nội suy khoảng 0.15 là ít hơn tỷ lệ thời gian chạy 0.25. Trong Hình 5.1, tỷ lệ nội suy luôn bằng tỷ lệ thời gian chạy.

Khi đã có tỷ lệ nội suy, đối tượng ValueAnimator gọi đối tượng TypeEvaluator thích hợp để tính giá trị của thuộc tính bạn đang tạo hoạt hình, dựa trên tỷ lệ nội suy, giá trị bắt đầu và giá trị kết thúc của hoạt hình. Ví dụ, trong Hình 5.2,

tỷ lệ nội suy là 0.15 tại t = 10 ms, vì vậy giá trị của thuộc tính tại thời điểm đó là 0.15 X (40 - 0) = 6.

Gói com.example.android.apis.animation trong project mẫu API Demos cung cấp nhiều ví dụ về cách sử dụng hệ thống property animation.

Điểm khác biệt giữa Property animation và View animation

Hệ thống view animation chỉ cung cấp khả năng tạo hoạt hình cho các đối tượng View, vì vậy nếu muốn tạo hoạt hình cho các kiểu đối tượng khác, bạn phải có cách viết mã riêng. Hệ thống view animation cũng có giới hạn là chỉ cho phép tạo hoạt hình cho một vài đặc điểm của một đối tượng View như phóng to thu nhỏ và quay đối tượng, nhưng không thể tạo hoạt hình cho những thứ khác như màu nền.

Điểm yếu nữa của hệ thống view animation là chỉ thay đổi vị trí đối tượng View được vẽ, chứ không phải thay đổi bản thân đối tượng View. Ví dụ, nếu bạn tạo hoạt hình cho một nút bấm để nó di chuyển ngang màn hình, nút bấm được vẽ chính xác, nhưng vị trí thực nơi bạn có thể nhấn nút không thay đổi, vì vậy bạn phải tự viết mã lôgic để xử lý việc này.

Với hệ thống property animation, những hạn chế này được loại bỏ hoàn toàn, và bạn có thể tạo cho bất cứ thuộc tính nào của bất cứ đối tượng nào (View hay không phải View) và bản thân đối tượng được thực sự thay đổi. Hệ thống property animation cũng mạnh hơn trong cách thực hiện hoạt hình. Ở mức độ tổng quát, bạn gán các Animator[4] cho các thuộc tính muốn tạo hoạt hình, như màu sắc, vị trí, hoặc kích thước và chỉ định các đặc điểm của hoạt hình như phép nội suy và việc đồng bộ nhiều Animator.

Tuy vậy, hệ thống view animation cần ít thời gian cài đặt và viết ít mã hơn. Nếu view animation đáp ứng mọi thứ bạn cần, hoặc mã hiện tại đang hoạt động đúng như bạn muốn, bạn không cần thiết sử dụng hệ thống property animation. Cũng có những trường hợp cần đến cả hai hệ thống hoạt hình cho những tình huống khác nhau.

[4] *Từ gọi chung cho các đối tượng ValueAnimator, ObjectAnimator, AnimatorSet.*

Tổng quan về API

Bạn có thể tìm thấy hầu hết các API của hệ thống property animation trong android.animation. Vì hệ thống view animation định nghĩa sẵn nhiều Interpolator[5] trong android.view.animation, bạn cũng có thể sử dụng các Interpolator này trong hệ thống property animation. Các bảng dưới đây mô tả các thành phần chính của hệ thống property animation.

Lớp Animator cung cấp cấu trúc cơ bản cho việc tạo hoạt hình. Thông thường, bạn không sử dụng lớp này trực tiếp vì nó chỉ cung cấp tính năng tối thiểu phải được mở rộng để hỗ trợ đầy đủ cho các giá trị của hoạt hình. Các lớp sau kế thừa Animator:

[5] *Từ gọi chung cho các đối tượng định nghĩa các phép nội suy hoạt hình, chẳng hạn như TimeInterpolator.*

Bảng 1. Các Animator

Lớp	Mô tả
ValueAnimator	Bộ phận tính toán thời gian chính cho property animation, đồng thời tính toán các giá trị cho thuộc tính sẽ được tạo hoạt hình. Lớp này chứa tất cả tính năng lõi giúp tính toán các giá trị và chứa thông tin thời gian chi tiết của mỗi hoạt hình, thông tin về việc hoạt hình có lặp lại không, các listener lắng nghe các sự kiện cập nhật và khả năng thiết đặt các kiểu tùy chỉnh để ước tính giá trị. Hai công đoạn để tạo hoạt hình trên một thuộc tính là: tính toán các giá trị hoạt hình và gán các giá trị đó vào đối tượng và thuộc tính đang được tạo hoạt hình. ValueAnimator không thực hiện công đoạn thứ hai, vì vậy bạn phải lắng nghe các cập nhật cho các giá trị được tính bởi ValueAnimator và sửa đổi các đối tượng bạn muốn tạo hoạt hình bằng mã của mình. Bạn có thể đọc phần "Tạo hoạt hình với ValueAnimator" để tìm hiểu thêm.
ObjectAnimator	Một lớp con của ValueAnimator cho phép bạn thiết lập một đối tượng đích và một thuộc tính của đối tượng để tạo hoạt hình. Lớp này đồng thời cập nhật thuộc tính của đối tượng khi nó tính được một giá trị mới cho hoạt hình. Trong hầu hết các trường hợp, bạn sẽ sử dụng ObjectAnimator vì lớp này giúp đơn giản hóa quá trình tạo hoạt hình các giá trị trên các đối tượng đích. Tuy nhiên, đôi khi bạn sẽ muốn sử dụng trực tiếp ValueAnimator vì ObjectAnimator có thêm một số ràng buộc như đòi hỏi đối tượng đích có các phương thức truy cập (get/set - truy xuất/thiết lập giá trị thuộc tính) cụ thể.
AnimatorSet	Cung cấp cơ chế nhóm các hoạt hình để chúng có thể kết hợp chạy cùng nhau. Bạn có thể thiết đặt các hoạt hình chạy cùng nhau, chạy tuần tự, hoặc sau một khoảng trễ nhất định. Bạn có thể đọc phần "Dàn dựng nhiều hoạt hình với AnimatorSet" để tìm hiểu thêm.

Các Evaluator[6] báo cho hệ thống property animation biết cách tính toán các giá trị cho một thuộc tính nhất định. Evaluator nhận dữ liệu thời gian từ lớp Animator, giá trị bắt đầu và kết thúc của hoạt hình, và tính toán các giá trị hoạt hình của thuộc tính. Hệ thống property animation cung cấp các Evaluator sau:

[6] *Tên gọi chung cho các đối tượng TypeEvaluator, IntEvaluator …*

Bảng 2. Các Evaluator

Lớp/Giao diện	Mô tả
IntEvaluator	Evaluator mặc định để tính toán các giá trị cho các thuộc tính kiểu int.
FloatEvaluator	Evaluator mặc định để tính toán các giá trị cho các thuộc tính kiểu float.
ArgbEvaluator	Evaluator mặc định để tính toán các giá trị cho các thuộc tính liên quan đến màu sắc được biểu diễn dưới dạng giá trị theo hệ thập lục phân.
TypeEvaluator	Một giao diện cho phép bạn tự tạo Evaluator riêng. Nếu bạn đang tạo hoạt hình trên một thuộc tính đối tượng *không phải* kiểu int, float hoặc màu sắc, bạn phải kế thừa giao diện TypeEvaluator để chỉ định cách tính toán các giá trị hoạt hình cho thuộc tính đó. Bạn cũng có thể chỉ định một TypeEvaluator tùy chỉnh cho các giá trị kiểu int, float và màu sắc nếu muốn xử lý các kiểu này khác với hành vi mặc định. Bạn có thể đọc phần "Sử dụng TypeEvaluator" để tìm hiểu cách viết một Evaluator tùy chỉnh.

Một Interpolator chỉ định cách tính các giá trị cụ thể của một hoạt hình dưới dạng một hàm của thời gian. Ví dụ, bạn có thể chỉ định các hoạt hình xảy ra một cách tuyến tính, tức là hoạt hình di chuyển đều đặn trong toàn bộ thời gian, hoặc bạn có thể chỉ định các hoạt hình sử dụng thời gian phi tuyến tính, ví dụ như tăng tốc ở đoạn đầu và giảm tốc ở đoạn cuối của hoạt hình. Bảng 3 mô tả các Interpolator có trong android.view.animation. Nếu không Interpolator có sẵn nào đáp ứng nhu cầu của bạn, hãy kế thừa giao diện TimeInterpolator và tạo Interpolator của riêng bạn. Bạn có thể đọc phần "Sử dụng Interpolator" để tìm hiểu cách viết một Interpolator.

Bảng 3. Các Interpolator

Lớp/Giao diện	Mô tả
AccelerateDecelerateInterpolator	Một Interpolator với tốc độ thay đổi chậm khi bắt đầu và kết thúc, nhưng tăng lên ở đoạn giữa.
AccelerateInterpolator	Một Interpolator với tốc độ thay đổi chậm khi bắt đầu và sau đó tăng dần.
AnticipateInterpolator	Một Interpolator bắt đầu chuyển động theo chiều ngược sau đó chạy nhanh theo chiều xuôi.
AnticipateOvershootInterpolator	Một Interpolator bắt đầu chuyển động chiều ngược, chạy nhanh theo chiều xuôi, chạy vượt qua giá trị đích và cuối cùng trở về giá trị kết thúc.
BounceInterpolator	Một Interpolator với chuyển động nảy lên ở đoạn cuối.
CycleInterpolator	Một Interpolator với hoạt hình lặp lại một số lần nhất định.
DecelerateInterpolator	Một Interpolator với tốc độ thay đổi nhanh khi bắt đầu và sau đó giảm dần.
LinearInterpolator	Một Interpolator với tốc độ thay đổi một lượng không đổi.
OvershootInterpolator	Một Interpolator với sự thay đổi chạy nhanh theo chiều xuôi, vượt quá giá trị cuối cùng và quay trở lại.
TimeInterpolator	Một giao diện cho phép bạn kế thừa để tạo Interpolator riêng.

Tạo hoạt hình với ValueAnimator

Lớp ValueAnimator cho phép bạn tạo hoạt hình với các giá trị thuộc một số kiểu dữ liệu trong khoảng thời gian xảy ra hoạt hình bằng cách chỉ định một tập các giá trị kiểu int, float hoặc màu sắc. Bạn có thể có một đối tượng ValueAnimator bằng cách gọi một trong các phương thức factory của ValueAnimator: ofInt(), ofFloat() hoặc ofObject(). Ví dụ:

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);
animation.setDuration(1000);
animation.start();
```

Trong đoạn mã này, khi phương thức start() chạy, ValueAnimator bắt đầu tính toán các giá trị của hoạt hình, từ 0 đến 1, trong thời lượng 1000 ms.

Bạn cũng có thể dùng một kiểu tùy chỉnh để tạo hoạt hình bằng cách sau:

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(),
                          startPropertyValue, endPropertyValue);
animation.setDuration(1000);
animation.start();
```

Trong đoạn mã này, khi phương thức start() chạy, ValueAnimator bắt đầu tính toán các giá trị của hoạt hình, từ startPropertyValue đến endPropertyValue, với lôgic có trong MyTypeEvaluator trong thời lượng 1000 ms.

Tuy nhiên, đoạn mã trên không có tác động thực sự lên một đối tượng, vì ValueAnimator không thao tác trực tiếp trên các đối tượng hoặc thuộc tính. Trong hầu hết các trường hợp, điều bạn muốn là sửa đổi các đối tượng muốn tạo hoạt hình với các giá trị đã tính toán này. Bạn có thể làm điều đó bằng cách định nghĩa các listener trong ValueAnimator để xử lý các sự kiện quan trọng trong vòng đời của hoạt hình một cách thích hợp, ví dụ như các sự kiện cập nhật khung hình. Khi kế thừa các listener này, bạn có thể lấy giá trị đã tính toán cho lần làm tươi khung hình cụ thể đó bằng cách gọi getAnimatedValue(). Bạn có thể đọc phần “Các listener cho hoạt hình” để tìm hiểu thêm.

Tạo hoạt hình với ObjectAnimator

ObjectAnimator là lớp con của ValueAnimator (đã trình bày ở phần trước); kết hợp bộ phận tính toán thời gian và tính toán giá trị của ValueAnimator với khả năng tạo hoạt hình trên một thuộc tính được định danh trên đối tượng đích. Khả

năng này giúp đơn giản hóa việc tạo hoạt hình cho bất cứ đối tượng nào, vì bạn không còn cần kế thừa giao diện ValueAnimator.AnimatorUpdateListener, bởi thuộc tính được tạo hoạt hình sẽ tự động cập nhật.

Khởi tạo một đối tượng ObjectAnimator cũng tương tự như ValueAnimator, nhưng bạn cần chỉ định đối tượng và tên thuộc tính của đối tượng đó (dưới dạng một chuỗi) cùng với các giá trị bắt đầu và kết thúc của khoảng thay đổi.

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

Để ObjectAnimator cập nhật các thuộc tính thành công, bạn phải đảm bảo:

- Thuộc tính mà bạn đang tạo hoạt hình phải có một phương thức thiết lập giá trị thuộc tính (tên theo cú pháp lạc đà) có dạng set<TenThuocTinh>(). Vì ObjectAnimator tự động cập nhật thuộc tính trong quá trình diễn ra hoạt hình, ObjectAnimator phải truy cập được thuộc tính với phương thức set này. Ví dụ, nếu tên thuộc tính là foo, bạn cần có một phương thức setFoo(). Nếu phương thức set này không tồn tại, bạn có ba lựa chọn:
  - Thêm phương thức set vào lớp đó nếu bạn có quyền làm vậy.
  - Sử dụng một lớp mà bạn có quyền thay đổi và để lớp đó nhận giá trị với một phương thức set hợp lệ và chuyển tiếp giá trị đó tới đối tượng gốc.
  - Sử dụng lớp ValueAnimator.
- Nếu bạn chỉ chỉ định một giá trị cho tham số values... cho một trong các phương thức factory của ObjectAnimator, giá trị đó được coi là giá trị kết thúc của hoạt hình. Vì vậy, thuộc tính bạn đang tạo phải có một phương thức get dùng để lấy giá trị bắt đầu của hoạt hình. Phương thức get phải có dạng get<TenThuocTinh>(). Ví dụ, nếu tên thuộc tính là foo, bạn cần có một phương thức getFoo().
- Phương thức get (nếu cần) và phương thức set của thuộc tính bạn đang tạo hoạt hình phải có cùng kiểu với các giá trị bắt đầu và kết thúc bạn chỉ định cho ObjectAnimator. Ví dụ, bạn phải có targetObject.setPropName(float) và targetObject.getPropName(float) nếu bạn tạo đối tượng ObjectAnimator như sau:

```
ObjectAnimator.ofFloat(targetObject, "propName", 1f)
```

- Tùy thuộc vào thuộc tính hoặc đối tượng đang tạo hoạt hình, bạn có thể cần gọi phương thức invalidate() trên một đối tượng View để hủy hiệu lực của View và bắt buộc màn hình tự vẽ lại View với các giá trị hoạt

hình cập nhật. Bạn có thể làm điều này trong phương thức callback onAnimationUpdate(). Ví dụ, việc tạo hoạt hình trên thuộc tính màu sắc của một đối tượng Drawable chỉ khiến màn hình cập nhật khi đối tượng vẽ lại chính đối tượng. Tất cả các phương thức set cho thuộc tính trên View như setAlpha() và setTranslationX() đều tự hủy hiệu lực View vì vậy bạn không cần tự gọi invalidate() trên đối tượng View khi gọi các phương thức này với các giá trị mới. Bạn có thể đọc phần "Các listener cho hoạt hình" để tìm hiểu thêm.

Dàn dựng nhiều hoạt hình với AnimatorSet

Trong nhiều trường hợp, bạn muốn phát một hoạt hình dựa trên sự kiện bắt đầu hoặc kết thúc của một hoạt hình khác. Hệ thống Android cho phép bạn kết hợp nhiều hoạt hình với nhau trong một AnimatorSet, nhờ đó bạn có thể chỉ định có chạy các hoạt hình đồng thời, tuần tự hoặc sau một khoảng trễ nhất định hay không. Bạn cũng có thể lồng các đối tượng AnimatorSet trong nhau.

Đoạn mã mẫu dưới đây, được lấy từ ứng dụng mẫu Bouncing Balls tạo hoạt hình chuyển động nảy lên của quả bóng (đã được chỉnh sửa đơn giản hóa), phát các đối tượng Animator theo cách sau:

1. Phát bounceAnim.
2. Phát squashAnim1, squashAnim2, stretchAnim1 và stretchAnim2 cùng lúc.
3. Phát bounceBackAnim.
4. Phát fadeAnim.

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start();
```

Để xem ví dụ hoàn chỉnh về cách dùng các AnimatorSet, bạn có thể tham khảo ứng dụng mẫu Bouncing Balls trong bộ APIDemos.

Các listener cho hoạt hình

Bạn có thể lắng nghe các sự kiện quan trọng trong thời gian chạy của một hoạt hình với các listener dưới đây.

- Animator.AnimatorListener
  - onAnimationStart() - Được gọi khi hoạt hình bắt đầu.
  - onAnimationEnd() - Được gọi khi hoạt hình kết thúc.
  - onAnimationRepeat() – Được gọi khi hoạt hình lặp lại.
  - onAnimationCancel() – Được gọi khi hoạt hình bị hủy. Một hoạt hình bị hủy cũng gọi onAnimationEnd() bất kể nó bị kết thúc như thế nào.
- ValueAnimator.AnimatorUpdateListener
  - onAnimationUpdate() – được gọi trên mỗi khung hình của hoạt hình. Bạn lắng nghe sự kiện này để sử dụng những giá trị đã tính toán do ValueAnimator sinh ra trong quá trình chạy hoạt hình. Để sử dụng giá trị này, bạn truy vấn đối tượng ValueAnimator được truyền vào sự kiện để lấy giá trị hiện tại với phương thức getAnimatedValue(). Việc kế thừa listener này là bắt buộc nếu bạn sử dụng ValueAnimator.

    Tùy thuộc vào thuộc tính hoặc đối tượng đang tạo hoạt hình, bạn có thể cần gọi phương thức invalidate() trên một đối tượng View để bắt buộc màn hình tự vẽ lại View với các giá trị mới. Ví dụ, việc tạo hoạt hình cho thuộc tính màu sắc của một đối tượng Drawable chỉ khiến màn hình cập nhật khi đối tượng vẽ lại chính nó. Tất cả các phương thức set cho thuộc tính trên View như setAlpha() và setTranslationX() đều tự hủy hiệu lực View, vì vậy bạn không cần tự hủy hiệu lực View khi gọi các phương thức này với các giá trị mới.

Bạn có thể kế thừa lớp AnimatorListenerAdapter thay vì kế thừa giao diện Animator.AnimatorListener nếu bạn không muốn hiện thực hóa tất cả các phương thức của giao diện Animator.AnimatorListener. Lớp AnimatorListenerAdapter cung cấp sẵn một phần thân rỗng cho các phương thức và bạn có thể ghi đè một vài phương thức trong số này theo nhu cầu của mình.

Ví dụ, ứng dụng mẫu Bouncing Balls trong bộ API demos tạo một AnimatorListenerAdapter chỉ để sử dụng phương thức callback onAnimationEnd():

```
ValueAnimatorAnimator fadeAnim = ObjectAnimator.ofFloat(newBall,
                                                "alpha", 1f, 0f);
fadeAnim.setDuration(250);
fadeAnim.addListener(new AnimatorListenerAdapter() {
   public void onAnimationEnd(Animator animation) {
      balls.remove(((ObjectAnimator)animation).getTarget());
   }})
```

Tạo hoạt hình cho những thay đổi về Layout trên ViewGroup

Hệ thống property animation cung cấp khả năng tạo hoạt hình cho những thay đổi trên các đối tượng ViewGroup cũng như cung cấp cách thức thuận tiện để tạo hoạt hình cho bản thân các đối tượng View.

Bạn có thể tạo hoạt hình để thay đổi về layout trong một ViewGroup với lớp LayoutTransition. Các đối tượng View bên trong một ViewGroup có thể tạo hoạt hình xuất hiện và biến mất khi bạn thêm hoặc bớt chúng khỏi ViewGroup hoặc khi bạn gọi phương thức setVisibility() của một đối tượng View với giá trị VISIBLE, android.view.View#INVISIBLE} hoặc GONE. Các đối tượng View còn lại trong ViewGroup cũng có thể tạo hiệu ứng di chuyển vào vị trí mới khi bạn thêm hoặc bớt các đối tượng View. Bạn có thể định nghĩa các hoạt hình sau trong đối tượng LayoutTransition bằng cách gọi setAnimator() và truyền vào một đối tượng Animator với một trong các hằng số sau của lớp LayoutTransition:

- APPEARING – Cờ chỉ thị hoạt hình đối với các đối tượng đang xuất hiện trong đối tượng chứa.
- CHANGE_APPEARING – Cờ chỉ thị hoạt hình đối với các đối tượng đang thay đổi vì một đối tượng mới xuất hiện trong đối tượng chứa.
- DISAPPEARING – Cờ chỉ thị hoạt hình đối với các đối tượng đang biến mất khỏi đối tượng chứa.
- CHANGE_DISAPPEARING – Cờ chỉ thị hoạt hình đối với các đối tượng đang thay đổi vì một đối tượng đang biến mất khỏi đối tượng chứa.

Bạn có thể tự định nghĩa các hoạt hình tùy chỉnh cho bốn kiểu sự kiện này để tùy biến hình thức của các hiệu ứng chuyển tiếp layout, hoặc chỉ cần để hệ thống sử dụng các hoạt hình mặc định.

Ứng dụng mẫu LayoutAnimations trong bộ API Demos cho bạn thấy cách định

nghĩa các hoạt hình cho hiệu ứng chuyển tiếp layout và sau đó thiết lập các hoạt hình trên các đối tượng View mà bạn muốn làm động.

LayoutAnimationsByDefault và file tài nguyên layout tương ứng layout_animations_by_default.xml cho bạn thấy cách kích hoạt các hiệu ứng chuyển tiếp layout mặc định cho các ViewGroup bằng XML. Điều duy nhất bạn cần làm là đặt thuộc tính android:animateLayoutchanges của ViewGroup bằng true. Ví dụ:

```
<LinearLayout
   android:orientation="vertical"
   android:layout_width="wrap_content"
   android:layout_height="match_parent"
   android:id="@+id/verticalContainer"
   android:animateLayoutChanges="true" />
```

Việc thiết lập thuộc tính này bằng true sẽ tự động tạo hoạt hình cho các đối tượng View được thêm vào hoặc xóa đi trong ViewGroup cũng như các đối tượng View còn lại trong ViewGroup.

Sử dụng TypeEvaluator

Nếu bạn muốn tạo hoạt hình với một kiểu dữ liệu xa lạ với hệ thống Android, bạn có thể tự tạo Evaluator bằng cách kế thừa giao diện TypeEvaluator. Các kiểu mà hệ thống Android hiểu được là int, float hoặc kiểu dữ liệu về màu sắc được hỗ trợ bởi các Evaluator là IntEvaluator, FloatEvaluator và ArgbEvaluator.

Chỉ có một phương thức cần hiện thực hóa trong giao diện TypeEvaluator là phương thức evaluate(). Phương thức này cho phép Animator bạn đang dùng trả về một giá trị phù hợp cho thuộc tính đang được tạo hoạt hình tại thời điểm hiện tại của hoạt hình. Lớp FloatEvaluator là một ví dụ cho chức năng này:

```
public class FloatEvaluator implements TypeEvaluator {

   public Object evaluate(float fraction, Object startValue,
   Object endValue) {
      float startFloat = ((Number) startValue).floatValue();
      return startFloat + fraction * (((Number) endValue).floatValue()
                                       - startFloat);
   }
}
```

> **Lưu ý:** Khi ValueAnimator (hoặc ObjectAnimator) chạy, nó tính toán tỷ lệ thời gian chạy hiện tại của hoạt hình (một giá trị trong khoảng 0 và 1) và sau đó tính tỷ lệ nội suy dựa trên Interpolator bạn đang sử dụng. Tỷ lệ sau nội suy được truyền vào đối tượng TypeEvaluator thông qua tham số fraction, vì vậy bạn không phải tính đến Interpolator khi tính toán các giá trị làm động.

Sử dụng các Interpolator

Một Interpolator định nghĩa cách tính các giá trị cụ thể của một hoạt hình dưới dạng một hàm của thời gian. Ví dụ, bạn có thể chỉ định hoạt hình xảy ra một cách tuyến tính, tức là hoạt hình di chuyển đều đặn trong toàn bộ quãng thời gian hoặc bạn có thể chỉ định hoạt hình sử dụng thời gian phi tuyến tính, ví dụ như tăng tốc ở đoạn đầu và giảm tốc ở đoạn cuối của hoạt hình.

Các Interpolator trong hệ thống hoạt hình nhận một tỷ lệ biểu diễn thời gian đã chạy của hoạt hình từ các Animator. Interpolator sửa đổi tỷ lệ này cho phù hợp với kiểu hoạt hình cần sinh ra. Hệ thống Android cung cấp một bộ các Interpolator phổ biến trong gói android.view.animation. Nếu không Interpolator có sẵn nào đáp ứng nhu cầu của bạn, hãy kế thừa giao diện TimeInterpolator và tạo Interpolator của riêng bạn.

Bạn có thể xem ví dụ so sánh cách tính tỷ lệ nội suy của hai Interpolator mặc định AccelerateDecelerateInterpolator và LinearInterpolator dưới đây. LinearInterpolator không có tác động gì tới tỷ lệ thời gian chạy. AccelerateDecelerateInterpolator tăng tốc khi bắt đầu hoạt hình và giảm tốc khi kết thúc. Các phương thức sau định nghĩa lôgic cho hai Interpolator này:

AccelerateDecelerateInterpolator

```
public float getInterpolation(float input) {
    return (float)(Math.cos((input +1) * Math.PI) / 2.0f) + 0.5f;
}
```

LinearInterpolator

```
public float getInterpolation(float input) {
    return input;
}
```

Bảng sau biểu diễn các giá trị gần đúng được tính bởi hai Interpolator này cho một hoạt hình chạy trong 1000 ms:

Thời gian hoạt hình đã chạy (ms)	Tỷ lệ thời gian chạy / Tỷ lệ nội suy (Tuyến tính)	Tỷ lệ nội suy (Tăng tốc/Giảm tốc)
0	0	0
200	0.2	0.1
400	0.4	0.345
600	0.6	0.8
800	0.8	0.9
1000	1	1

Theo như bảng trên, LinearInterpolator thay đổi các giá trị với tốc độ không đổi, 0.2 với mỗi 200 ms trôi qua. AccelerateDecelerateInterpolator thay đổi các giá trị nhanh hơn LinearInterpolator trong khoảng 200 ms và 600 ms, và chậm hơn trong khoảng 600 ms và 1000 ms.

Chỉ định Khung hình Chính (Keyframe)

Một đối tượng Keyframe gồm một cặp thời gian/giá trị cho phép bạn định nghĩa một trạng thái cụ thể của hoạt hình tại một thời điểm cụ thể. Mỗi khung hình chính (keyframe) còn có thể có Interpolator riêng để kiểm soát hành vi của hoạt hình trong khoảng thời gian giữa khung hình chính trước và khung hình chính này.

Để khởi tạo một đối tượng Keyframe, bạn phải sử dụng một trong các phương thức factory của Keyframe: ofInt(), ofFloat() hoặc ofObject() để có kiểu Keyframe thích hợp. Sau đó, bạn có thể gọi phương thức factory ofKeyframe() để có một đối tượng PropertyValuesHolder. Lúc này, bạn có thể tạo một Animator bằng cách truyền đối tượng PropertyValuesHolder và đối tượng sẽ tạo hoạt hình. Bạn hãy xem minh họa trong đoạn mã sau:

```
Keyframe kf0 = Keyframe.ofFloat(0f, 0f);
Keyframe kf1 = Keyframe.ofFloat(.5f, 360f);
Keyframe kf2 = Keyframe.ofFloat(1f, 0f);
PropertyValuesHolder pvhRotation =
   PropertyValuesHolder.ofKeyframe("rotation", kf0, kf1, kf2);
ObjectAnimator rotationAnim =
   ObjectAnimator.ofPropertyValuesHolder(target, pvhRotation);
rotationAnim.setDuration(5000ms);
```

Để xem ví dụ hoàn chỉnh về cách dùng các khung hình chính, bạn có thể tham khảo ứng dụng mẫu MultiPropertyAnimation trong bộ APIDemos.

Tạo hoạt hình cho View

Hệ thống property animation cho phép tạo hoạt hình với các đối tượng View một cách tiện lợi và có một số ưu điểm so với hệ thống view animation. Hệ thống view animation biến đổi các đối tượng View bằng cách thay đổi cách vẽ chúng. Thao tác này được xử lý trong đối tượng chứa của mỗi View, vì bản thân đối tượng View không có thuộc tính để biến đổi. Điều này dẫn tới việc đối tượng View được tạo hoạt hình nhưng không gây ra thay đổi trong bản thân đối tượng View. Đặc điểm này dẫn tới những hành vi như một đối tượng vẫn tồn tại ở vị trí ban đầu, dù nó đã được vẽ ở một vị trí khác trên màn hình. Trong Android 3.0, các thuộc tính mới và các phương thức get và set tương ứng đã được thêm vào để loại bỏ nhược điểm này.

Hệ thống property animation có thể tạo hoạt hình cho các đối tượng View trên màn hình bằng cách thay đổi các thuộc tính thực trong đối tượng View. Ngoài ra, View cũng tự động gọi phương thức invalidate() để làm tươi màn hình bất cứ khi nào các thuộc tính của nó bị thay đổi. Các thuộc tính mới trong lớp View hỗ trợ cơ chế property animation là:

- translationX và translationY: Các thuộc tính này kiểm soát vị trí đặt đối tượng View nhờ vào khoảng cách từ tọa độ trái và tọa độ trên của View, hai khoảng cách này được quyết định bởi layout chứa View.
- rotation, rotationX và rotationY: Các thuộc tính này kiểm soát chuyển động xoay 2D (thuộc tính rotation) và 3D quanh điểm chốt.
- scaleX và scaleY: Các thuộc tính này kiểm soát việc thu phóng 2D của đối tượng View quanh điểm chốt của nó.
- pivotX và pivotY: Các thuộc tính này kiểm soát vị trí của điểm chốt mà chuyển động xoay và thu phóng xảy ra xung quanh điểm đó. Mặc định, điểm chốt xoay nằm ở trung tâm của đối tượng.
- x và y: Đây là các thuộc tính bổ trợ đơn giản để mô tả vị trí cuối cùng của đối tượng View trong đối tượng chứa nó, được tính bằng tổng của giá trị trái hoặc trên với giá trị translationX hoặc translationY.
- alpha: Thể hiện độ trong suốt của đối tượng View. Giá trị mặc định là 1 (mờ đục), còn giá trị 0 thể hiện độ trong suốt hoàn toàn (không hiển thị).

Để tạo hoạt hình cho một thuộc tính của một đối tượng View như màu hoặc độ xoay, tất cả việc bạn cần làm là tạo một property animator và chỉ định thuộc tính của View mà bạn muốn tạo hoạt hình. Ví dụ:

```
ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);
```

Để tìm hiểu thêm về cách tạo các Animator, bạn hãy xem phần tạo hoạt hình với ValueAnimator và ObjectAnimator.

Tạo hoạt hình với ViewPropertyAnimator

ViewPropertyAnimator cung cấp một cách đơn giản để tạo hoạt hình trên một số thuộc tính của một đối tượng View đồng thời chỉ sử dụng một đối tượng Animator cơ sở. ViewPropertyAnimator hoạt động rất giống một đối tượng ObjectAnimator vì nó thay đổi các giá trị thực của các thuộc tính của View, nhưng đạt hiệu quả hơn khi tạo hoạt hình cho nhiều thuộc tính cùng lúc. Ngoài ra, đoạn mã sử dụng ViewPropertyAnimator ngắn gọn và dễ đọc hơn nhiều. Đoạn mã sau cho thấy sự khác biệt giữa việc sử dụng nhiều đối tượng ObjectAnimator, một đối tượng ObjectAnimator và ViewPropertyAnimator khi tạo hoạt hình đồng thời cho thuộc tính x và y của một đối tượng View.

Nhiều đối tượng ObjectAnimator

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView, "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView, "y", 100f);
AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

Một ObjectAnimator

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvyY).start();
```

ViewPropertyAnimator

```
myView.animate().x(50f).y(100f);
```

Để tìm hiểu thêm về ViewPropertyAnimator, bạn có thể đọc bài blog này trên Android Developers blog.

Khai báo hoạt hình trong XML

Hệ thống property animation cho phép bạn khai báo các property animation bằng XML thay vì bằng mã lập trình. Bằng cách định nghĩa các hoạt hình trong XML, bạn có thể dễ dàng sử dụng lại các hoạt hình trong nhiều Activity và dễ dàng sửa đổi thứ tự các hoạt hình.

Để tách biệt các file hoạt hình sử dụng API property animation mới khỏi các file sử dụng framework view animation cũ, bắt đầu từ Android 3.1, bạn nên lưu các file XML định nghĩa các property animation trong thư mục res/animator/ (thay vì thư mục res/anim/). Việc sử dụng tên thư mục animator là không bắt buộc, nhưng là cần thiết nếu bạn muốn sử dụng các công cụ của trình xử lý layout trong Eclipse ADT plugin (ADT 11.0.0+) vì ADT chỉ tìm kiếm các tài nguyên property animation trong thư mục res/animator/.

Dưới đây là các lớp property animation có hỗ trợ việc khai báo bằng XML và các thẻ XML khai báo tương ứng:

- ValueAnimator - <animator>
- ObjectAnimator - <objectAnimator>
- AnimatorSet - <set>

Ví dụ sau phát tuần tự hai bộ hoạt hình, với bộ thứ nhất phát hai hoạt hình cùng nhau:

```
<set android:ordering="sequentially">
   <set>
      <objectAnimator
         android:propertyName="x"
         android:duration="500"
         android:valueTo="400"
         android:valueType="intType" />
      <objectAnimator
         android:propertyName="y"
         android:duration="500"
         android:valueTo="300"
         android:valueType="intType" />
   </set>
   <objectAnimator
      android:propertyName="alpha"
      android:duration="500"
      android:valueTo="1f" />
</set>
```

Để chạy hoạt hình này, bạn phải khai triển (inflate) các tài nguyên XML trong mã để tạo một đối tượng AnimatorSet, sau đó gán các đối tượng đích cho tất cả các hoạt hình trước khi bắt đầu chạy bộ hoạt hình. Việc gọi setTarget() sẽ gán một đối tượng đích duy nhất cho tất cả các con của đối tượng AnimatorSet một cách nhanh chóng. Ví dụ sau minh họa cách thực hiện:

```
AnimatorSet set =
(AnimatorSet) AnimatorInflater.loadAnimator(myContext,
                                            R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

Bạn có thể tìm hiểu thêm về cú pháp XML để định nghĩa các property animation trong phần "Các Tài nguyên Hoạt hình".

View Animation

Bạn có thể sử dụng hệ thống View animation để thực hiện các hoạt hình dạng tween (tween animation)[7] trên các View. Hoạt hình dạng tween tính toán hoạt hình với các thông tin như điểm bắt đầu, điểm kết thúc, kích thước, độ xoay và các đặc điểm chung khác của một hoạt hình.

Hoạt hình dạng tween có thể thực hiện một loạt các biến đổi đơn giản về vị trí, kích thước, độ xoay, và độ trong suốt với nội dung của một đối tượng View. Vì vậy nếu bạn có một đối tượng TextView, bạn có thể di chuyển, xoay, phóng to hoặc thu nhỏ dòng chữ đó. Nếu dòng chữ có ảnh nền, ảnh nền sẽ được biến đổi cùng với chữ. Gói android.view.animation cung cấp tất cả các lớp dùng trong một hoạt hình dạng tween.

Hoạt hình dạng tween được định nghĩa bằng một chuỗi các chỉ thị hoạt hình viết bằng XML hoặc mã Android. Giống việc định nghĩa layout, sử dụng file XML là cách được khuyên dùng vì dễ đọc, dễ tái sử dụng và dễ thay thế hơn sử dụng mã cứng. Trong ví dụ dưới đây, chúng ta sẽ sử dụng XML. (Để biết cách định nghĩa hoạt hình bằng mã lập trình trong ứng dụng thay vì bằng XML, bạn hãy tìm hiểu lớp AnimationSet và các lớp con của Animation.)

Các chỉ thị hoạt hình định nghĩa những biến đổi bạn muốn xảy ra, khi nào chúng xảy ra và thời gian chạy chúng. Các biến đổi có thể xảy ra tuần tự hoặc đồng thời – ví dụ, bạn có thể di chuyển nội dung của một TextView từ trái sang phải và sau đó xoay 180 độ, hoặc bạn có thể di chuyển và xoay nó đồng thời. Mỗi biến đổi nhận một bộ tham số đặc trưng cho biến đổi đó (kích thước bắt

[7] *Tạo hoạt hình bằng cách tạo một loạt các biến đổi trên một hình ảnh bằng cách sử dụng lớp trừu tượng Animation.*

đầu và kích thước kết thúc nếu là biến đổi về kích thước; góc bắt đầu và góc kết thúc nếu là biến đổi về chuyển động xoay …) và một bộ tham số chung (ví dụ, thời gian bắt đầu và thời lượng). Để một vài biến đổi xảy ra đồng thời, bạn hãy đặt cùng thời gian bắt đầu. Để chúng chạy theo thứ tự, bạn hãy tính thời gian bắt đầu cộng với thời lượng của biến đổi trước đó.

File XML định nghĩa hoạt hình nằm trong thư mục res/anim/ trong project Android của bạn. File này phải có một phần tử gốc duy nhất: một trong các phần tử <alpha>, <scale>, <translate>, <rotate>, phần tử interpolator hoặc phần tử <set> có khả năng chứa các nhóm gồm những phần tử này (cũng có thể chứa các phần tử <set> khác). Theo mặc định, tất cả các chỉ thị hoạt hình được áp dụng đồng thời. Để chúng chạy tuần tự, bạn phải chỉ định thuộc tính startOffset như trong ví dụ dưới đây.

Đoạn XML sau nằm trong một ứng dụng mẫu trong bộ Api Demos định nghĩa thao tác kéo dãn, sau đó quay tròn và xoay đồng thời một đối tượng View.

```
<set android:shareInterpolator="false">
   <scale
      android:interpolator="@android:anim/accelerate_decelerate_interpolator"
      android:fromXScale="1.0"
      android:toXScale="1.4"
      android:fromYScale="1.0"
      android:toYScale="0.6"
      android:pivotX="50%"
      android:pivotY="50%"
      android:fillAfter="false"
      android:duration="700" />
   <set android:interpolator="@android:anim/decelerate_interpolator">
      <scale
        android:fromXScale="1.4"
        android:toXScale="0.0"
        android:fromYScale="0.6"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="400"
        android:fillBefore="false" />
      <rotate
        android:fromDegrees="0"
        android:toDegrees="-45"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="400" />
   </set>
</set>
```

Các tọa độ trên màn hình (không được dùng trong ví dụ này) bắt đầu từ (0,0) ở góc trên bên trái và tăng dần khi bạn di chuyển xuống và sang phải.

Một số giá trị như pivotX có thể được chỉ định tương đối so với bản thân đối tượng hoặc tương đối với đối tượng cha. Hãy đảm bảo bạn sử dụng đúng định dạng mong muốn ("50" để chỉ 50% so với đối tượng cha hoặc "50%" để chỉ 50% so với bản thân đối tượng).

Bạn có thể quyết định biến đổi xảy ra như thế nào theo thời gian bằng cách gán một Interpolator. Android có sẵn một số lớp nội suy, lớp con của Interpolator, với các đồ thị tốc độ khác nhau: Ví dụ, AccelerateInterpolator chỉ thị việc biến đổi bắt đầu chậm và sau đó tăng tốc. Mỗi Interpolator có một giá trị thuộc tính tương ứng dùng để khai báo trong XML.

Với đoạn XML này lưu thành file hyperspace_jump.xml trong thư mục res/anim/ của project, đoạn mã sau sẽ tham chiếu tới file XML và sử dụng file này cho một đối tượng ImageView từ layout.

```
ImageView spaceshipImage = (ImageView) findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation =
      AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

Thay vì sử dụng startAnimation(), bạn có thể chỉ định thời gian bắt đầu cho hoạt hình với Animation.setStartTime() và sau đó gán hoạt hình cho đối tượng View với View.setAnimation().

Để tìm hiểu thêm về cú pháp XML, các thẻ và thuộc tính có sẵn, bạn hãy xem phần "Các tài nguyên Hoạt hình".

> **Lưu ý:** Bất kể hoạt hình di chuyển hay thay đổi kích thước như thế nào, đường biên của đối tượng View chứa hoạt hình sẽ không tự động điều chỉnh một cách phù hợp. Hoạt hình có thể được vẽ vượt bên ngoài đường biên của đối tượng View và hiện tượng clipping[8] *sẽ xảy ra* nếu hoạt hình vượt quá đường biên của đối tượng View cha.

Drawable Animation

Drawable animation cho phép bạn nạp một loạt các tài nguyên dạng Drawable theo tuần tự để tạo hoạt hình. Đây là kiểu hoạt hình truyền thống theo cách hiểu là nó được tạo ra với một chuỗi các hình ảnh khác nhau, phát theo thứ tự giống như một cuộn phim. Lớp AnimationDrawable là lớp cở sở của các Drawable animation.

Trong khi bạn có thể định nghĩa các khung hình cho một hoạt hình bằng mã lập trình với API của lớp AnimationDrawable, bạn có thể thực hiện điều đó với một file XML duy nhất liệt kê các khung hình cho một hoạt hình. File XML định nghĩa hoạt hình nằm trong thư mục res/drawable/ trong project Android của bạn. Trong trường hợp này, các chỉ thị gồm thứ tự và thời lượng mỗi khung hình của hoạt hình.

File XML này gồm một phần tử gốc <animation-list> và một loạt các nút con <item> định nghĩa từng khung hình: một tài nguyên dạng drawable cho khung hình và thời lượng khung hình. Đây là ví dụ một file XML dành cho một Drawable animation:

[8] *Không kết xuất và hiển thị những vùng hình ảnh không nằm trong khu vực hiển thị.*

```
<animation-list
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
       <item android:drawable="@drawable/rocket_thrust1"
             android:duration="200"/>
       <item android:drawable="@drawable/rocket_thrust2"
             android:duration="200"/>
       <item android:drawable="@drawable/rocket_thrust3"
             android:duration="200"/>
</animation-list>
```

Hoạt hình này chỉ chạy ba khung hình. Với thuộc tính android:oneshot của danh sách bằng *true*, hoạt hình sẽ chỉ chạy một vòng và dừng ở khung hình cuối cùng. Nếu bạn sử dụng *false* hoạt hình sẽ lặp đi lặp lại. Với đoạn mã XML này lưu thành file rocket_thrust.xml trong thư mục res/drawable/ của project, các drawable này có thể được dùng làm ảnh nền cho một đối tượng View và sau đó được gọi để chạy. Đây là ví dụ về một Activity trong đó hoạt hình trên được dùng cho một đối tượng ImageView và được chạy khi người dùng chạm vào màn hình:

```
AnimationDrawable rocketAnimation;

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

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_DOWN) {
    rocketAnimation.start();
    return true;
  }
  return super.onTouchEvent(event);
}
```

Một điểm quan trọng cần lưu ý là phương thức start() của đối tượng AnimationDrawable không thể được gọi trong phương thức onCreate() của Activity vì đối tượng AnimationDrawable chưa được gắn hoàn toàn vào cửa sổ (đối tượng window). Nếu bạn muốn phát hoạt hình ngay lập tức mà không cần tương tác, bạn có thể thực hiện lời gọi từ phương thức onWindowFocusChanged() trong Activity của bạn, phương thức này sẽ được gọi khi Android đưa cửa sổ của bạn thành cửa sổ hiện hành.

Để tìm hiểu thêm về cú pháp XML, các thẻ và thuộc tính có sẵn, bạn hãy xem phần “Các tài nguyên Hoạt hình”.

Canvas và Drawable

NỘI DUNG BÀI HỌC

- Vẽ với Canvas
  - Vẽ trên đối tượng View
  - Vẽ trên đối tượng SurfaceView
- Drawable
  - Tạo từ tài nguyên ảnh
  - Tạo từ tài nguyên XML
- ShapeDrawable
- Nine-patch

BÀI ĐỌC THÊM

- OpenGL với các API của Framework
- RenderScript

Framework Android cung cấp một bộ API vẽ 2D cho phép bạn kết xuất đồ họa tùy ý trên một canvas hoặc chỉnh sửa các đối tượng View có sẵn để tùy biến hình thức của chúng. Khi vẽ đồ họa 2D, bạn thường thực hiện theo một trong hai cách:

a. Vẽ hình ảnh hoặc hoạt hình vào một đối tượng View từ layout của bạn. Theo cách này, việc vẽ đồ họa được xử lý bởi quá trình vẽ cây phân cấp View thông thường của hệ thống – bạn có thể dễ dàng tích hợp hình ảnh vào trong View.
b. Vẽ hình ảnh trực tiếp lên Canvas. Theo cách này, bạn trực tiếp gọi phương thức onDraw() của lớp thích hợp (truyền đối tượng Canvas của bạn vào đó) hoặc một trong các phương thức draw...() của đối tượng Canvas (như drawPicture()). Như vậy, bạn có thể kiểm soát bất cứ hoạt hình nào.

Phương án "a", vẽ vào đối tượng View là lựa chọn tốt nhất khi bạn muốn vẽ những hình ảnh đơn giản mà không cần thay đổi liên tục và không phục vụ cho một trò chơi đòi hỏi hiệu suất cao. Ví dụ, bạn nên vẽ hình ảnh vào đối tượng View khi muốn hiển thị một hình ảnh tĩnh hoặc một hoạt hình được định trước trong một ứng dụng tương đối tĩnh. Bạn có thể đọc phần Drawable để tìm hiểu thêm.

Phương án "b", vẽ vào một đối tượng Canvas là lựa chọn tốt hơn trong trường hợp ứng dụng cần vẽ lại chính nó thường xuyên. Các ứng dụng như trò chơi video nên vẽ lên Canvas của chính nó. Tuy nhiên, có nhiều hơn một cách để làm điều này:

- Trong cùng luồng của Activity trên UI của bạn, bạn tạo một thành phần View tùy chỉnh trong layout, gọi invalidate() và sau đó xử lý phương thức callback onDraw().
- Hoặc trong một luồng riêng, bạn quản lý một đối tượng SurfaceView và thực hiện các thao tác vẽ lên Canvas nhanh hết khả năng của luồng (bạn không cần gọi invalidate()).

Vẽ với Canvas

Nếu bạn đang viết một ứng dụng trong đó bạn muốn thực thiện các thao tác vẽ chuyên dụng và/hoặc kiểm soát hoạt hình của các đối tượng đồ họa, bạn nên vẽ thông qua đối tượng Canvas. Canvas đóng vai trò vật thay thế hay giao diện cho bề mặt thực sự dùng để vẽ đồ họa – Canvas giữ lại tất các các lời gọi "draw". Thông qua Canvas, các thao tác vẽ thực sự được thực hiện trên một Bitmap bên dưới nằm trong cửa sổ.

Trong phương thức callback onDraw() vào thời điểm sự kiện vẽ diễn ra, bạn sẽ được cung cấp đối tượng Canvas và bạn chỉ cần dùng đối tượng này để thực hiện các lời gọi vẽ. Bạn cũng có thể lấy đối tượng Canvas từ SurfaceHolder.lockCanvas() khi làm việc với đối tượng SurfaceView. (Cả hai tình huống đều được trình bày trong các phần sau). Tuy nhiên, nếu bạn cần tạo một Canvas mới, bạn phải định nghĩa đối tượng Bitmap mà trên đó thao tác vẽ được thực sự thực hiện. Bitmap là điều kiện cần cho Canvas. Bạn có thể tạo một Canvas mới như sau:

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

Giờ Canvas của bạn sẽ thực hiện vẽ trên đối tượng Bitmap đã định nghĩa. Sau khi vẽ lên đó với Canvas, bạn có thể chuyển Bitmap sang Canvas khác với một trong các phương thức Canvas.drawBitmap(Bitmap,...). Bạn nên luôn vẽ các đối tượng đồ họa cuối cùng thông qua đối tượng Canvas được cung cấp bởi phương thức View.onDraw() hoặc SurfaceHolder.lockCanvas() (xem các phần sau).

Lớp Canvas có một bộ phương thức vẽ riêng như drawBitmap(...), drawRect(...), drawText(...) ... Bạn cũng có thể dùng một số lớp khác cũng có các phương thức draw(). Ví dụ, bạn có thể có một số đối tượng Drawable mà bạn muốn đặt lên Canvas. Các đối tượng Drawable có phương thức draw() riêng nhận Canvas làm đối số.

Vẽ trên đối tượng View

Nếu ứng dụng của bạn không đòi hỏi lượng xử lý lớn hoặc tốc độ khung hình cao (có thể là trò chơi cờ vua, rắn hoặc một ứng dụng với hoạt hình chậm), bạn có thể quyết định việc tự định nghĩa một thành phần View và vẽ lên đó bằng cách truyền Canvas vào phương thức View.onDraw(). Điểm tiện lợi nhất của cách làm này là framework Android sẽ cung cấp cho bạn một đối tượng Canvas định nghĩa sẵn mà có thể dùng để gọi các thao tác vẽ.

Trước tiên, bạn hãy kế thừa lớp View (hoặc một lớp khác là lớp con của View) và khai báo phương thức callback onDraw(). Phương thức này sẽ được gọi bởi framework Android để yêu cầu đối tượng View tự vẽ lại chính nó. Phương thức onDraw() là nơi bạn thực hiện tất cả các lời gọi vẽ thông qua đối tượng Canvas được truyền vào cho phương thức.

Framework Android sẽ chỉ gọi onDraw() khi cần thiết. Mỗi lần ứng dụng chuẩn bị vẽ, bạn phải gọi phương thức invalidate() để thông báo rằng bạn muốn vẽ lại đối tượng View và Android sẽ gọi phương thức onDraw() (dù không đảm bảo rằng phương thức callback sẽ được gọi lập tức).

Trong phương thức onDraw() của View, bạn sử dụng Canvas đã được cung cấp để thực hiện tất cả các thao tác vẽ với các phương thức Canvas.draw...(), hoặc các phương thức draw() khác nhận Canvas làm đối số. Khi phương thức onDraw() hoàn thành, framework Android sẽ sử dụng Canvas để vẽ một đối tượng Bitmap được hệ thống quản lý.

> **Lưu ý:** Để yêu cầu hủy hiệu lực View từ một luồng khác với luồng của Activity chính, bạn phải gọi postInvalidate().

Bạn có thể đọc bài viết về việc Xây dựng Thành phần Tùy chỉnh để tìm hiểu cách kế thừa lớp View.

Để tham khảo ứng dụng mẫu, bạn có thể xem trò chơi Snake trong thư mục samples của bộ SDK: <thư mục sdk của bạn>/samples/Snake/.

Vẽ trên đối tượng SurfaceView

SurfaceView là một lớp con đặc biệt của View với khả năng cung cấp một bề mặt vẽ chuyên dụng trong cây phân cấp View. Mục đích của lớp này là cung cấp bề mặt vẽ cho một luồng phụ của ứng dụng, nhờ đó ứng dụng không cần chờ cho đến khi cây phân cấp View của hệ thống sẵn sàng để được vẽ. Thay vào đó, luồng phụ có tham chiếu tới đối tượng SurfaceView có thể tự do vẽ lên Canvas của đối tượng SurfaceView.

Trước tiên, bạn cần tạo một lớp kế thừa SurfaceView. Lớp này cũng cần kế thừa giao diện SurfaceHolder.Callback. Lớp con này là một giao diện thông báo cho bạn những thông tin về đối tượng Surface bên dưới, như khi nào nó được tạo, thay đổi hoặc hủy. Các sự kiện này là cần thiết để bạn biết khi nào bạn có thể bắt đầu vẽ, bạn có cần điều chỉnh dựa trên các thuộc tính mới của đối tượng Surface không, khi nào dừng vẽ và có thể chấm dứt một số tác vụ. Bên trong lớp SurfaceView là nơi thích hợp để khai báo lớp Thread phụ sẽ thực hiện tất cả các thao tác vẽ với Canvas.

Thay vì làm việc trực tiếp với đối tượng Surface, bạn nên làm việc với đối tượng Surface thông qua một SurfaceHolder. Khi SurfaceView được khởi tạo, bạn có thể lấy đối tượng SurfaceHolder bằng cách gọi getHolder(). Tiếp đó, bạn cần thông báo với SurfaceHolder rằng bạn muốn nhận các lời gọi lại của SurfaceHolder (từ SurfaceHolder.Callback) bằng cách gọi addCallback() (và truyền *this* cho phương thức này). Sau đó bạn cần ghi đè từng phương thức trong SurfaceHolder.Callback trong lớp SurfaceView của bạn.

Để vẽ vào Canvas của đối tượng Surface từ luồng phụ, bạn phải truyền SurfaceHandler vào luồng đó và truy xuất Canvas với phương thức lockCanvas(). Giờ bạn có thể dùng Canvas có được từ SurfaceHolder và thực hiện các thao tác vẽ lên đó. Khi đã vẽ xong, bạn cần gọi unlockCanvasAndPost() và truyền cho phương thức này đối tượng Canvas của bạn. Lúc này đối tượng Surface sẽ kết xuất Canvas như bạn đã vẽ. Hãy thực hiện chuỗi các bước khóa và mở khóa canvas như trên mỗi lần bạn muốn vẽ lại.

> **Lưu ý:** Với mỗi lần bạn truy xuất Canvas từ SurfaceHolder, trạng thái trước đó của Canvas vẫn được lưu giữ. Để tạo hoạt hình đồ họa đúng, bạn phải vẽ lại toàn bộ Surface. Ví dụ, bạn có thể xóa trạng thái trước đó của Canvas bằng cách phủ màu lên với drawColor() hoặc đặt một bức ảnh nền với drawBitmap(). Trái lại, bạn sẽ thấy dấu vết của các hình đã vẽ trước đó.

Để tham khảo ứng dụng mẫu, bạn có thể xem trò chơi Luna Lander trong thư mục samples của bộ SDK: <thư mục sdk của bạn>/samples/LunarLander/. Hoặc bạn có thể xem mã nguồn trong phần Sample Code.

Drawable

Android cung cấp một thư viện đồ họa 2D riêng cho việc vẽ các hình và ảnh. Gói android.graphics.drawable là nơi bạn có thể tìm thấy các lớp chung dùng cho việc vẽ đồ họa hai chiều.

Bài học này trình bày kiến thức cơ bản về cách sử dụng các đối tượng Drawable để vẽ đồ họa và cách sử dụng một số lớp con của lớp Drawable. Bạn có thể đọc phần "Drawable Animation" để tìm hiểu cách dùng các đối tượng Drawable cho hoạt hình theo từng khung hình.

Drawable là lớp trừu tượng chung cho "những thứ có thể được vẽ ra". Bạn sẽ biết được rằng lớp Drawble mở rộng thành rất nhiều loại đồ họa có thể vẽ được như BitmapDrawable, ShapeDrawable, PictureDrawable, LayerDrawable và một số khác nữa. Dĩ nhiên, bạn cũng có thể kế thừa các lớp này để tự định nghĩa những đối tượng Drawable riêng của bạn với những hành vi độc nhất.

Có ba cách định nghĩa và khởi tạo một đối tượng Drawable: dùng một ảnh đã lưu trong tài nguyên project, dùng một file XML định nghĩa các thuộc tính của đối tượng Drawable; hoặc sử dụng các phương thức khởi tạo bình thường của lớp này. Sau đây, chúng ta sẽ xem xét hai kỹ thuật đầu tiên (dùng phương thức khởi tạo không có gì mới mẻ đối với một lập trình viên nhiều kinh nghiệm).

Tạo từ tài nguyên ảnh

Cách đơn giản để đưa đồ họa vào ứng dụng là tham chiếu một file ảnh từ tài nguyên của project. Các kiểu file được hỗ trợ gồm PNG (ưu tiên), JPG (chấp nhận được) và GIF (không được khuyên dùng). Kỹ thuật này dĩ nhiên được ưa dùng cho biểu tượng ứng dụng, logo hoặc các hình ảnh khác như các hình ảnh dùng trong một trò chơi.

Để sử dụng một tài nguyên ảnh, bạn chỉ cần thêm file ảnh vào thư mục res/drawable/ của project. Sau đó, bạn tham chiếu tới ảnh từ mã lập trình hoặc từ layout XML. Dù theo cách nào, tài nguyên ảnh đều được tham chiếu bằng resource ID (định danh tài nguyên), tức là tên file không tính đuôi mở rộng (ví dụ: tham chiếu tới file my_image.png bằng *my_image*).

> **Lưu ý:** Các tài nguyên ảnh đặt trong res/drawable/ có thể được tối ưu hóa tự động với thuật toán nén ảnh không gây tổn hao dữ liệu bằng công cụ aapt trong quá trình build. Ví dụ, một bức ảnh PNG màu-thực (true-color) không cần hơn 256 màu có thể được chuyển đổi thành một file PNG 8-bit bằng cách dùng một bảng màu. Tính năng này đem lại một file ảnh cùng chất lượng nhưng dùng ít bộ nhớ hơn. Vì vậy, bạn cần lưu ý rằng các file ảnh đặt trong thư mục này có thể thay đổi trong lúc build. Nếu bạn dự định đọc file ảnh dưới dạng bit stream (dòng bit) để chuyển đổi nó thành một bitmap, hãy đặt file ảnh vào thư mục res/raw/, trong thư mục này ảnh sẽ không bị tối ưu hóa.

Đoạn mã ví dụ

Đoạn mã sau minh họa cách tạo một ImageView sử dụng một file ảnh từ tài nguyên và đưa ImageView vào layout.

```
LinearLayout mLinearLayout;

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

 // Tạo một LinearLayout chứa ImageView
 mLinearLayout = new LinearLayout(this);

 // Khởi tạo một ImageView và thiết lập các thuộc tính của nó
 ImageView i = new ImageView(this);
 i.setImageResource(R.drawable.my_image);
 i.setAdjustViewBounds(true); // Thiết đặt các đường biên của
                              // ImageView khớp với kích thước của
                             // đối tượng Drawable
 i.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
 LayoutParams.WRAP_CONTENT));

 // Đưa ImageView vào layout và đặt layout làm nội dung của View
 mLinearLayout.addView(i);
 setContentView(mLinearLayout);
 }
```

Trong các trường hợp khác, bạn có thể muốn xử lý tài nguyên ảnh dưới dạng một đối tượng Drawable. Để làm điều đó, bạn có thể tạo đối tượng Drawable từ tài nguyên như sau:

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```

> **Lưu ý:** Mỗi tài nguyên trong project chỉ có thể duy trì một trạng thái duy nhất, bất kể có bao nhiêu đối tượng được bạn khởi tạo với nó. Ví dụ, nếu bạn khởi tạo hai đối tượng Drawable từ cùng một tài nguyên ảnh, sau đó thay đổi thuộc tính (ví dụ alpha) của một trong hai đối tượng Drawable, đối tượng còn lại cũng sẽ bị ảnh hưởng. Vì vậy, khi làm việc với nhiều thể hiện của một tài nguyên ảnh, thay vì trực tiếp biến đổi đối tượng Drawable, bạn nên thực hiện một hoạt hình dạng tween.

Đoạn mã XML ví dụ

Đoạn XML dưới đây minh họa cách đưa một tài nguyên Drawable vào một ImageView trong layout XML (với một chút ánh đỏ cho vui mắt qua thuộc tính android:tint).

```
<ImageView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:tint="#55ff0000"
     android:src="@drawable/my_image" />
```

Bạn có thể đọc bài viết về Các tài nguyên trong Ứng dụng để tìm hiểu thêm cách dùng các tài nguyên trong dự án.

Tạo từ tài nguyên XML

Lúc này, hẳn bạn đã quen thuộc với các nguyên tắc phát triển Giao diện Người dùng của Android. Vì vậy, bạn cũng đã hiểu sức mạnh và sự linh hoạt vốn có trong việc định nghĩa các đối tượng bằng XML. Triết lý này cũng ảnh hưởng tới khái niệm View và Drawable. Nếu có một đối tượng Drawable bạn muốn tạo và ban đầu nó không phụ thuộc vào các biến định nghĩa trong mã lập trình hoặc sự tương tác của người dùng, việc định nghĩa đối tượng Drawable đó bằng XML là một phương án tốt. Kể cả khi bạn dự kiến đối tượng Drawable đó sẽ thay đổi thuộc tính trong quá trình sử dụng của người dùng, bạn cũng nên cân nhắc việc định nghĩa đối tượng bằng XML vì bạn có thể sửa đổi thuộc tính tùy ý sau khi đối tượng được khởi tạo.

Khi bạn đã định nghĩa đối tượng Drawable trong XML, hãy lưu file vào thư mục res/drawable/ của project. Sau đó, bạn có thể truy xuất và khởi tạo đối tượng bằng cách gọi Resources.getDrawable() và truyền vào resource ID của file XML. (Xem ví dụ bên dưới).

Bất cứ lớp con nào của Drawable hỗ trợ phương thức inflate() cũng có thể được định nghĩa bằng XML và được khởi tạo bởi ứng dụng. Mỗi đối tượng Drawable có hỗ trợ khai triển XML sử dụng các thuộc tính XML riêng để định nghĩa thuộc tính đối tượng (hãy xem tài liệu tham khảo về lớp này để biết các

thuộc tính đó). Bạn cần xem tài liệu đặc tả của mỗi lớp con của Drawable để biết cách định nghĩa chúng bằng XML.

Ví dụ

Đây là một đoạn XML định nghĩa một đối tượng TransitionDrawable:

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/image_expand">
    <item android:drawable="@drawable/image_collapse">
    </transition>
```

Lưu đoạn XML này vào file res/drawable/expand_collapse.xml. Đoạn mã sau sẽ khởi tạo đối tượng TransitionDrawable và đặt đối tượng làm nội dung của một ImageView:

```
Resources res = mContext.getResources();
TransitionDrawable transition =
  (TransitionDrawable)res.getDrawable(R.drawable.expand_collapse);
ImageView image = (ImageView) findViewById(R.id.toggle_image);
image.setImageDrawable(transition);
```

Hiệu ứng chuyển tiếp này có thể chạy xuôi (trong 1 giây) với:

```
transition.startTransition(1000);
```

Bạn cần tham khảo các lớp Drawble liệt kê bên trên để tìm hiểu thông tin về các thuộc tính XML được hỗ trợ bởi mỗi lớp.

ShapeDrawable

Nếu bạn muốn vẽ một vài hình ảnh hai chiều tùy ý, đối tượng ShapeDrawable có thể sẽ đáp ứng nhu cầu của bạn. Với ShapeDrawable, bạn có thể dùng mã lập trình để vẽ các hình cơ bản và tạo kiểu cho chúng theo mọi cách có thể tưởng tượng.

ShapeDrawable là lớp mở rộng của Drawable, vì vậy bạn có thể sử dụng lớp này ở bất cứ nơi nào có thể sử dụng Drawable – có thể dùng làm hình nền của một View bằng cách sử dụng phương thức setBackgroundDrawable(). Dĩ nhiên, bạn cũng có thể vẽ các hình dưới dạng View tùy chỉnh và bạn có thể tùy ý đưa chúng vào các layout. Vì ShapeDrawable có phương thức draw() riêng, bạn có thể tạo một lớp con của View và vẽ đối tượng ShapeDrawble trong phương thức View.onDraw(). Ví dụ sau minh họa cách làm điều đó với một lớp kế thừa View và vẽ đối tượng ShapeDrawble dưới dạng View.

```
public class CustomDrawableView extends View {
private ShapeDrawable mDrawable;

public CustomDrawableView(Context context) {

   super(context);
   int x = 10;
   int y = 10;
   int width = 300;
   int height = 50;

   mDrawable = new ShapeDrawable(new OvalShape());
   mDrawable.getPaint().setColor(0xff74AC23);
   mDrawable.setBounds(x, y, x + width, y + height);

}


protected void onDraw(Canvas canvas) {

   mDrawable.draw(canvas);

}
}
```

Trong phương thức khởi tạo, đối tượng ShapeDrawable được định nghĩa dưới dạng OvalShape. Sau đó, đối tượng được thiết đặt màu và biên. Nếu bạn không thiết lập các biên, hình sẽ không được vẽ, trong khi nếu bạn không thiết lập màu, màu mặc định là màu đen.

Với đối tượng View tùy chỉnh được định nghĩa như đoạn mã trên, View có thể được vẽ theo bất kỳ cách nào bạn muốn. Với ví dụ trên, chúng ta có thể vẽ hình bằng cách viết mã trong một Activity:

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   mCustomDrawableView = new CustomDrawableView(this);

   setContentView(mCustomDrawableView);
}
```

Nếu muốn vẽ đối tượng Drawable này từ layout XML thay vì từ Activity, bạn phải ghi đè phương thức khởi tạo View(Context, AttributeSet) trong lớp CustomDrawableView vì phương thức này sẽ được gọi khi View được khởi tạo bằng cách khai triển XML. Sau đó, bạn có thể khai báo trong XML như sau:

```
<com.example.shapedrawable.CustomDrawableView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

Lớp ShapeDrawable (giống như nhiều kiểu Drawable khác trong gói android.graphics.drawable) cho phép bạn sử dụng nhiều thuộc tính thông qua các phương thức public. Một số thuộc tính bạn có thể muốn điều chỉnh gồm độ trong suốt alpha, bộ lọc màu (color filter), dither (tán sắc), độ mờ đục (opacity) và màu (color).

Bạn cũng có thể định nghĩa các hình cơ bản bằng XML. Để tìm hiểu thêm, bạn có thể đọc phần về ShapeDrawable trong tài liệu về Các tài nguyên Drawable.

Nine-patch

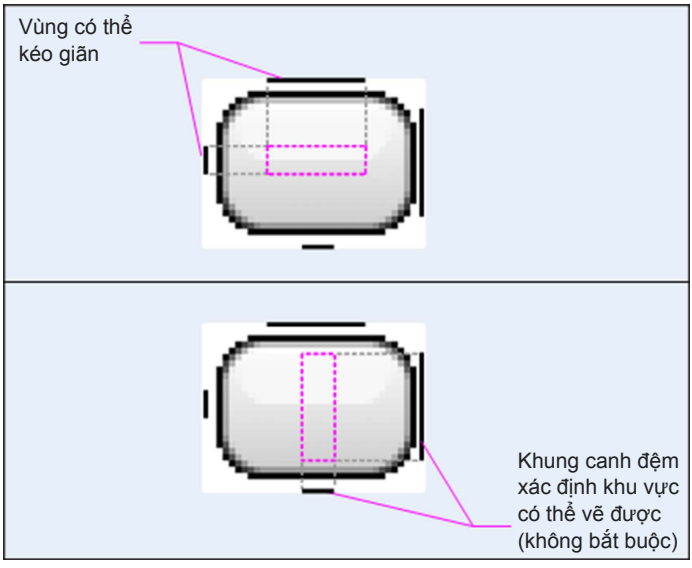
NinePatchDrawable là một dạng ảnh bitmap có thể kéo giãn được và Android sẽ tự động chỉnh kích thước của ảnh cho phù hợp với nội dung của View lấy ảnh này làm hình nền. Một ví dụ sử dụng NinePatch là hình nền của các nút bấm tiêu chuẩn của Android – các nút bấm phải giãn ra để chứa các chuỗi ký tự có độ dài khác nhau. Một tài nguyên drawable kiểu NinePatch là một bức ảnh PNG tiêu chuẩn chứa thêm một đường biên rộng 1 pixel. File ảnh này phải có đuôi mở rộng .9.png và lưu trong thư mục res/drawable/ của project.

Đường biên nói trên được dùng để xác định các vùng có thể kéo giãn và các vùng tĩnh của bức ảnh. Bạn chỉ định một khu vực kéo giãn bằng cách vẽ một (hoặc nhiều hơn một) đoạn thẳng màu đen rộng 1 pixel ở cạnh trái và cạnh trên của bức ảnh (các phần bên ngoài phải hoàn toàn trong suốt hoặc có màu trắng). Bạn có thể có bao nhiêu phần co giãn tùy ý: kích thước tương đối của chúng sẽ giữ nguyên, vì vậy phần lớn nhất sẽ luôn hiển thị là phần lớn nhất.

Dù không bắt buộc, bạn cũng có thể chỉ định phần có thể vẽ lên được của bức ảnh (đường canh đệm - padding) bằng cách vẽ một đoạn thẳng ở cạnh phải và cạnh dưới của bức ảnh. Nếu một đối tượng View đặt NinePatch làm hình nền, khi nội dung văn bản của View được thiết lập, View sẽ tự co giãn để tất cả nội dung văn bản nằm trong khu vực được chỉ định bởi đường kẻ bên dưới và bên phải (nếu có). Nếu không có các đường canh đệm, Android sử dụng các đường kẻ bên trái và bên trên để xác định khu vực có thể vẽ được.

Để tóm tắt sự khác biệt giữa các đường kẻ này, đường kẻ bên trái và bên trên xác định những điểm ảnh của bức ảnh được cho phép sao chép để kéo giãn bức ảnh. Đường kẻ bên dưới và bên phải xác định khu vực tương đối bên trong bức ảnh mà cho phép chứa nội dung của đối tượng View.

Đây là một file NinePatch mẫu được dùng để định nghĩa một nút bấm:



Bức ảnh NinePatch này định nghĩa khu vực co giãn với đường kẻ bên trái và bên trên, và khu vực có thể vẽ được với đường kẻ bên dưới và bên phải. Trong hình trên (hình đầu tiên), các đường kẻ màu xám đứt đoạn xác định các vùng trong bức ảnh sẽ được sao chép để kéo giãn bức ảnh. Hình chữ nhật màu hồng trong hình dưới xác định vùng cho phép chứa nội dung của View. Nếu nội dung không nằm vừa trong vùng này, bức ảnh sẽ được kéo giãn để đáp ứng.

Draw 9-patch là một công cụ cực kỳ hữu dụng để tạo các bức ảnh NinePatch với một trình xử lý đồ họa WYSIWYG. Công cụ này thậm chí còn đưa ra cảnh báo nếu vùng bạn chỉ định để kéo giãn có nguy cơ tạo các điểm ảnh lỗi do thao tác sao chép điểm ảnh.

Đoạn mã XML mẫu

Đây là một layout XML mẫu minh họa cách áp dụng một bức ảnh NinePatch cho hai nút bấm. (File ảnh NinePatch được lưu trong res/drawable/my_button_background.9.png)

```
<Button id=”@+id/tiny”
      android:layout_width=”wrap_content”
      android:layout_height=”wrap_content”
      android:layout_alignParentTop=”true”
      android:layout_centerInParent=”true”
      android:text=”Tiny”
      android:textSize=”8sp”
      android:background=”@drawable/my_button_background” />
```

```
<Button id="@+id/big"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerInParent="true"
      android:text="Biiiiiiig text!"
      android:textSize="30sp"
      android:background="@drawable/my_button_background" />
```

Lưu ý rằng chiều rộng và chiều cao được thiết đặt là “wrap_content” để nút bấm vừa khít với dòng chữ.

Dưới đây là hai nút bấm được kết xuất từ layout XML và bức ảnh NinePatch ở trên. Bạn hãy lưu ý chiều rộng và chiều cao của nút bấm thay đổi như thế nào với dòng chữ bên trong, và bức ảnh nền kéo giãn như thế nào để chứa nó.

OpenGL ES

NỘI DUNG BÀI HỌC

- Kiến thức cơ bản
  - o Các gói trong OpenGL ES
- Khai báo các yêu cầu về OpenGL
- Ánh xạ các Tọa độ cho những Đối tượng được vẽ
  - o Phép chiếu và máy quay trong ES 1.0
  - o Phép chiếu và máy quay trong ES 2.0 hoặc cao hơn
- Mặt của hình và Phép cuộn hình

- Các phiên bản OpenGL và Sự tương thích với Thiết bị
  - Hỗ trợ nén texture
  - Xác định các thành phần mở rộng OpenGL
  - Kiểm tra Phiên bản OpenGL ES
- Lựa chọn một Phiên bản OpenGL API

CÁC LỚP TRỌNG TÂM

- GLSurfaceView
- GLSurfaceView.Renderer

ỨNG DỤNG MẪU LIÊN QUAN

- GLSurfaceViewActivity
- GLES20Activity
- TouchRotateActivity
- Compressed Textures

BÀI ĐỌC THÊM

- Hiển thị Đồ họa với OpenGL ES
- OpenGL ES
- Chuẩn OpenGL ES 1.x
- Chuẩn OpenGL ES 2.x
- Chuẩn OpenGL ES 3.x

Android hỗ trợ đồ họa 2D và 3D hiệu suất cao với thư viện Open Graphics Library (OpenGL®), đặc biệt là OpenGL ES API. OpenGL là một API đồ họa đa nền tảng chỉ định một giao diện phần mềm chuẩn cho phần cứng xử lý đồ họa 3D. OpenGL ES là một nhánh của chuẩn OpenGL nhắm đến các thiết bị nhúng. Android hỗ trợ một số phiên bản OpenGL ES API:

- OpenGL ES 1.0 và 1.1 – Chuẩn API này được hỗ trợ bởi Android 1.0 hoặc cao hơn.
- OpenGL ES 2.0 – Chuẩn API này được hỗ trợ bởi Android 2.2 (API cấp 8) hoặc cao hơn.

- OpenGL ES 3.0 – Chuẩn API này được hỗ trợ bởi Android 4.3 (API cấp 18) hoặc cao hơn.

> **Cảnh báo:** Việc hỗ trợ OpenGL ES 3.0 API trên một thiết bị đòi hỏi một bản hiện thực hóa quy trình xử lý đồ họa (graphics pipeline) do nhà sản xuất thiết bị cung cấp. Một thiết bị chạy Android 4.3 hoặc cao hơn *có thể không hỗ trợ* OpenGL ES 3.0 API. Để tìm hiểu cách kiểm tra phiên bản OpenGL ES nào được hỗ trợ lúc chạy, bạn xem phần "Kiểm tra Phiên bản OpenGL ES".

> **Lưu ý:** API đặc thù cung cấp bởi framework Android tương tự như J2ME JSR239 OpenGL ES API, nhưng không giống hệt. Nếu bạn quen với chuẩn J2ME JSR239, hãy cảnh giác với các điểm khác biệt.

Kiến thức cơ bản

Android hỗ trợ OpenGL thông qua cả API của framework và NDK (Native Development Kit - Bộ công cụ phát triển thuần Android). Bài học này tập trung vào các giao diện trong framework Android. Để tìm hiểu về NDK, bạn có thể xem tài liệu về Android NDK.

Hai lớp cơ sở trong framework Android cho phép bạn tạo và điều khiển các đối tượng đồ họa với OpenGL ES API là: GLSurfaceView và GLSurfaceView.Renderer. Nếu bạn dự định dùng OpenGL trong ứng dụng Android, bước đầu tiên bạn cần làm là hiểu cách kế thừa các lớp này trong một Activity.

GLSurfaceView

Lớp này là một View trong đó bạn có thể vẽ và điều khiển các đối tượng bằng cách sử dụng các lời gọi của OpenGL API và lớp này có chức năng tương tự với SurfaceView. Bạn có thể sử dụng GLSurfaceView bằng cách tạo một thể hiện kiểu GLSurfaceView và gán đối tượng Renderer của bạn cho nó. Tuy nhiên, nếu bạn muốn bắt các sự kiện cảm ứng trên màn hình, bạn cần kế thừa lớp GLSurfaceView để hiện thực hóa các listener cho sự kiện cảm ứng, như hướng dẫn trong bài Responding to Touch Events (tạm dịch: Phản hồi các Sự kiện Cảm ứng).

GLSurfaceView.Renderer

Giao diện này định nghĩa các phương thức cần cho việc vẽ đồ họa trong một GLSurfaceView. Bạn phải viết một lớp riêng kế thừa giao diện này và gắn lớp này với thể hiện GLSurfaceView bằng phương thức GLSurfaceView.setRenderer().

Giao diện GLSurfaceView.Renderer đòi hỏi bạn hiện thực hóa các phương thức sau:

- onSurfaceCreated(): Hệ thống gọi phương thức này một lần khi tạo đối tượng GLSurfaceView. Bạn sử dụng phương thức này để thực hiện các hành động chỉ cần xảy ra một lần, ví dụ như thiết lập các tham số môi trường cho OpenGL hoặc khởi tạo các đối tượng đồ họa của OpenGL.
- onDrawFrame(): Hệ thống gọi phương thức này mỗi lần thực hiện vẽ lại trong GLSurfaceView. Bạn sử dụng phương thức này làm điểm thực thi chính cho việc vẽ (và vẽ lại) các đối tượng đồ họa.
- onSurfaceChanged(): Hệ thống gọi phương thức này khi đặc tính hình học của GLSurfaceView thay đổi, bao gồm việc thay đổi kích thước của GLSurfaceView hoặc thay đổi hướng của màn hình thiết bị. Ví dụ, hệ thống gọi phương thức này khi thiết bị đổi hướng từ dọc (portrait) sang ngang (landscape). Bạn sử dụng phương thức này để phản hồi lại những thay đổi trong đối tượng chứa GLSurfaceView.

Các gói trong OpenGL ES

Khi bạn đã thiết lập một đối tượng chứa (view) cho OpenGL ES bằng cách dùng GLSurfaceView và GLSurfaceView.Renderer, bạn có thể bắt đầu sử dụng các OpenGL API thông qua các lớp sau:

- Các gói trong OpenGL ES 1.0/1.1 API
  - android.opengl – Gói này cung cấp một giao diện static cho các lớp trong OpenGL ES 1.0/1.1 và hiệu suất tốt hơn các giao diện trong gói javax.microedition.khronos.
    - GLES10
    - GLES10Ext
    - GLES11
    - GLES11Ext
  - javax.microedition.khronos.opengles – Gói này cung cấp bản hiện thực hóa tiêu chuẩn của OpenGL ES 1.0/1.1.
    - GL10
    - GL10Ext

    - GL11
    - GL11Ext
    - GL11ExtensionPack
- OpenGL ES 2.0 API
    - android.opengl.GLES20 – Gói này cung cấp giao diện cho OpenGL ES 2.0 và được đưa vào từ bản Android 2.2 (API cấp 8).
- OpenGL ES 3.0 API
    - android.opengl.GLES30 – Gói này cung cấp giao diện cho OpenGL ES 3.0 và được đưa vào từ bản Android 4.3 (API cấp 18).

Nếu bạn muốn bắt đầu xây dựng ngay một ứng dụng với OpenGL ES, bạn hãy làm theo bài viết Displaying Graphics with OpenGL ES (tạm dịch: Hiển thị Đồ họa với OpenGL ES).

Khai báo các yêu cầu về OpenGL

Nếu ứng dụng của bạn sử dụng những tính năng của OpenGL không có sẵn trên tất cả các thiết bị, bạn phải đưa các yêu cầu này vào file AndroidManifest.xml của ứng dụng. Dưới đây là các khai báo OpenGL thường dùng nhất:

- **Các yêu cầu về phiên bản OpenGL ES** – Nếu ứng dụng chỉ hỗ trợ OpenGL ES 2.0, bạn phải khai báo yêu cầu này bằng cách thêm đoạn sau vào file kê khai:

```
<!—Báo cho hệ thống biết ứng dụng này yêu cầu OpenGL ES 2.0. -->
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

Khai báo này khiến Google Play ngăn chặn việc cài ứng dụng trên những thiết bị không hỗ trợ OpenGL ES 2.0. Nếu ứng dụng chỉ dành riêng cho các thiết bị có hỗ trợ OpenGL ES 3.0, bạn dùng khai báo sau trong file kê khai:

```
<!-- Báo cho hệ thống biết ứng dụng này yêu cầu OpenGL ES 3.0. -->
<uses-feature android:glEsVersion="0x00030000" android:required="true" />
```

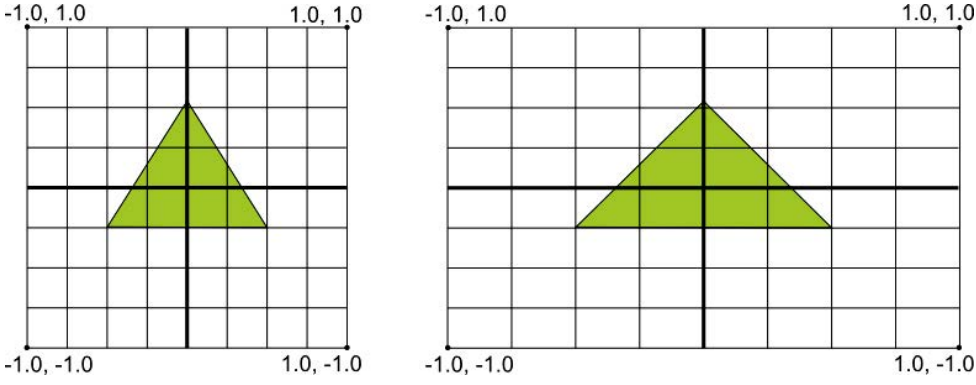
> **Lưu ý:** OpenGL ES 3.0 API tương thích ngược với 2.0 API, nghĩa là bạn có thể linh hoạt hơn khi sử dụng OpenGL ES trong ứng dụng. Với khai báo yêu cầu OpenGL ES 2.0 API trong file kê khai, bạn có thể sử dụng API phiên bản đó làm phiên bản mặc định, kiểm tra sự tồn tại của 3.0 API lúc chạy và sau đó sử dụng những tính năng của OpenGL ES 3.0 nếu thiết bị có hỗ trợ. Để tìm hiểu về cách kiểm tra phiên bản OpenGL ES nào được thiết bị hỗ trợ, bạn hãy xem phần “Kiểm tra Phiên bản OpenGL ES”.

- **Các yêu cầu về nén texture** – Nếu ứng dụng sử dụng các định dạng nén texture, bạn phải khai báo các định dạng được ứng dụng hỗ trợ trong file kê khai bằng phần tử <supports-gl-texture>. Để tìm hiểu về các định dạng nén texture hiện có, bạn hãy xem phần "Hỗ trợ nén texture".

  Việc khai báo các yêu cầu về nén texture trong file kê khai giúp ẩn ứng dụng khỏi những người dùng có thiết bị không hỗ trợ ít nhất một trong các kiểu nén bạn đã khai báo. Để tìm hiểu cách Google Play lọc các kiểu nén texture, bạn hãy xem bài viết Google Play and texture compression filtering (tạm dịch: Google Play và cách lọc kiểu nén texture) trong tài liệu đặc tả <supports-gl-texture>.

Ánh xạ các Tọa độ cho những Đối tượng được vẽ

Một trong những vấn đề cơ bản trong việc hiển thị đồ họa trên các thiết bị Android là sự khác nhau về kích thước và hình dạng của màn hình. OpenGL giả định có một hệ tọa độ ô vuông đồng đều và sẵn sàng vẽ các tọa độ đó lên màn hình thường không phải hình vuông của bạn như thể đó là hình vuông hoàn hảo.



Hình 5.4. Hệ tọa độ mặc định của OpenGL (bên trái) được ánh xạ sang một màn hình điển hình của Android (bên phải).

Hình minh họa trên cho thấy hệ tọa độ đồng đều mặc định cho một khung hình OpenGL ở bên trái và cách các tọa độ này được thực sự ánh xạ sang một màn hình thiết bị điển hình trong chế độ xoay ngang ở bên phải. Để giải quyết vấn đề này, bạn có thể áp dụng các chế độ phép chiếu (projection mode) và góc máy quay (camera view) của OpenGL để biến đổi các tọa độ, từ đó các đối tượng đồ họa sẽ có tỷ lệ đúng trên bất cứ màn hình nào.

Để áp dụng phép chiếu và góc máy quay, bạn tạo một ma trận phép chiếu và một ma trận góc máy quay, sau đó áp dụng chúng cho quy trình xử lý đồ họa của OpenGL. Ma trận phép chiếu tính toán lại tọa độ các đối tượng đồ họa để chúng được ánh xạ đúng tới các cỡ màn hình thiết bị của Android. Ma trận góc máy quay tạo một phép biến đổi khiến các đối tượng được kết xuất từ một vị trí nhìn cụ thể.

Phép chiếu và góc máy quay trong OpenGL ES 1.0

Trong ES 1.0 API, bạn sử dụng phép chiếu và góc máy quay bằng cách tạo từng ma trận và sau đó đưa chúng vào môi trường OpenGL.

1. **Ma trận phép chiếu** – được tạo dựa trên đặc điểm hình học của màn hình thiết bị, nhằm tính toán lại tọa độ các đối tượng, nhờ đó chúng được vẽ với tỷ lệ đúng. Đoạn mã ví dụ sau minh họa cách viết phương thức onSurfaceChanged() của một lớp kế thừa giao diện GLSurfaceView.Renderer để tạo ma trận phép chiếu dựa trên tỷ lệ màn hình và áp dụng lớp này vào môi trường kết xuất của OpenGL.

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);

    // điều chỉnh theo tỷ lệ màn hình
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);  // thiết lập chế độ phép chiếu
    gl.glLoadIdentity();  // thiết lập lại trạng thái
                          // mặc định cho ma trận
    gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);  // áp dụng ma trận
                                                // phép chiếu
}
```

2. **Ma trận biến đổi máy quay** - Khi bạn đã điều chỉnh hệ thống tọa độ bằng ma trận phép chiếu, bạn phải áp dụng một góc máy quay. Đoạn mã sau minh họa cách viết phương thức onDrawFrame() của một lớp kế thừa giao diện GLSurfaceView.Renderer để thiết lập ma trận biến đổi bằng GL_MODELVIEW và sử dụng GLU.gluLookAt() để tạo một phép biến đổi góc nhìn có tác dụng giả lập vị trí máy quay.

```
public void onDrawFrame(GL10 gl) {
    ...
    // Thiết lập chế độ biến đổi bằng GL_MODELVIEW
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    // thiết lập lại trạng thái mặc định cho ma trận

    // Khi sử dụng GL_MODELVIEW, bạn phải thiết lập góc máy quay
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
    ...
}
```

Phép chiếu và góc máy quay trong OpenGL ES 2.0 hoặc cao hơn

Trong API của ES 2.0 và 3.0, bạn áp dụng phép chiếu và góc máy quay bằng cách thêm một ma trận vào các vertex shader (bộ đổ bóng đỉnh)[9] của các đối

[9] *Có nhiệm vụ dựng ảnh dựa trên dữ liệu đỉnh*

tượng đồ họa. Với ma trận đã thêm, bạn có thể tạo, áp dụng phép chiếu và các ma trận góc máy quay trên các đối tượng của bạn.

1. **Thêm ma trận vào các vertex shader** - Tạo một biến chứa ma trận phép chiếu và dùng biến làm hệ số nhân cho vị trí của shader. Trong đoạn mã ví dụ sau về vertex shader, biến uMVPMatrix được đưa vào cho phép bạn áp dụng phép chiếu và các ma trận góc máy quay trên tọa độ của các đối tượng sử dụng shader này.

```
private final String vertexShaderCode =

    // Biến thành viên của ma trận này cung cấp điểm móc (hook)
    // để điều khiển các tọa độ của các đối tượng sử dụng vertex
    //shader này.
    "uniform mat4 uMVPMatrix;   \n"+

    "attribute vec4 vPosition;  \n" +
    "void main(){              \n" +
    // Ma trận phải nằm trong gl_Position
    // Lưu ý rằng thừa số uMVPMatrix *phải đứng đầu tiên* để có
    // kết quả nhân ma trận đúng.
    " gl_Position = uMVPMatrix * vPosition; \n" +

    "}  \n";
```

> **Lưu ý:** Ví dụ trên định nghĩa một biến thành viên của ma trận biến đổi duy nhất trong vertex shader mà bạn dùng để áp dụng một ma trận kết hợp của ma trận phép chiếu và ma trận góc máy quay. Tùy thuộc vào yêu cầu của ứng dụng, bạn có thể muốn định nghĩa riêng ma trận phép chiếu và ma trận góc máy quay trong vertex shader vì như vậy bạn có thể thay đổi chúng một cách độc lập.

2. **Truy cập ma trận shader** – Sau khi tạo điểm móc trong các vertex shader để áp dụng phép chiếu và góc máy quay, bạn có thể truy cập biến đó để gán ma trận phép chiếu và ma trận góc máy quay. Đoạn mã sau cho thấy cách viết phương thức onSurfaceCreated() của một lớp kế thừa giao diện GLSurfaceView.Renderer để truy cập biến ma trận đã định nghĩa trong vertex shader bên trên.

```
public void onSurfaceCreated(GL10 unused, EGLConfig config) {
    ...
    muMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram,
                                                    "uMVPMatrix");
    ...
}
```

3. **Tạo ma trận phép chiếu và góc máy quay** - Tạo các ma trận phép chiếu và góc máy quay để áp dụng cho các đối tượng đồ họa. Ví dụ sau cho thấy cách viết phương thức onSurfaceCreated() và onSurfaceChanged() của một lớp kế thừa giao diện GLSurfaceView.Renderer để tạo ma trận góc máy quay và ma trận phép chiếu dựa trên tỷ lệ màn hình của thiết bị.

```
public void onSurfaceCreated(GL10 unused,EGLConfig config) {
    ...
    // Tạo một ma trận góc máy quay
    Matrix.setLookAtM(mVMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
}

public void onSurfaceChanged(GL10 unused, int width, int height) {
    GLES20.glViewport(0, 0, width, height);

    float ratio = (float) width / height;

    // tạo một ma trận phép chiếu từ đặc điểm hình học của màn hình
    // thiết bị
    Matrix.frustumM(mProjMatrix, 0, -ratio, ratio, -1, 1, 3, 7);
}
```

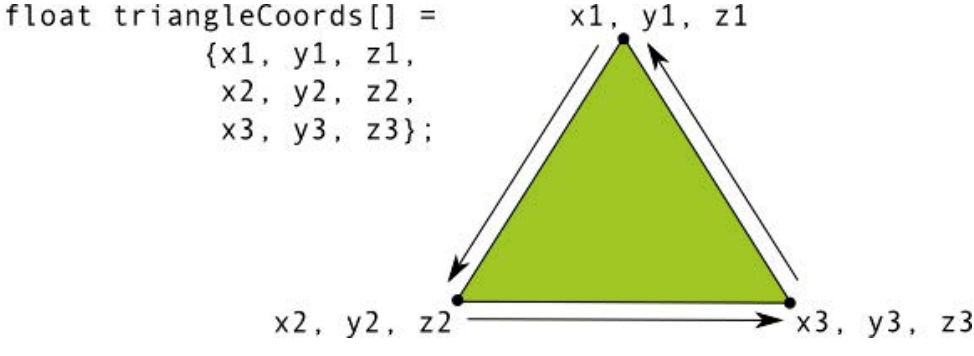
4. **Áp dụng ma trận phép chiếu và góc máy quay** – Để áp dụng phép biến đổi từ phép chiếu và góc máy quay, bạn nhân hai ma trận với nhau, sau đó gán chúng vào vertex shader. Đoạn mã sau cho thấy cách viết phương thức onDrawFrame() của một lớp kế thừa giao diện GLSurfaceView.Renderer để kết hợp ma trận phép chiếu và ma trận góc máy quay được tạo trong đoạn mã bên trên và áp dụng nó cho các đối tượng đồ họa sẽ được sinh ra bởi OpenGL.

```
public void onDrawFrame(GL10 unused) {
    ...
    // Kết hợp ma trận phép chiếu và ma trận góc máy quay
    Matrix.multiplyMM(mMVPMatrix, 0, mProjMatrix, 0, mVMatrix, 0);

    // Áp dụng phép biến đổi kết hợp phép chiếu và góc máy quay
    GLES20.glUniformMatrix4fv(muMVPMatrixHandle, 1, false,
                              mMVPMatrix, 0);

    // Vẽ các đối tượng
    ...
}
```

Để tham khảo ví dụ hoàn chỉnh về cách áp dụng phép chiếu và góc máy quay với OpenGL ES 2.0, bạn hãy xem bài viết Displaying Graphics with OpenGL ES (tạm dịch: Hiển thị Đồ họa với OpenGL ES).

Mặt của hình và Phép cuộn hình

Trong OpenGL, mặt của một hình là bề mặt được định nghĩa bởi ba điểm hoặc nhiều hơn trong không gian ba chiều. Một bộ ba hay nhiều hơn ba điểm trong không gian ba chiều (gọi là các đỉnh trong OpenGL) có một mặt trước và một mặt sau. Làm sao để biết mặt nào là mặt trước và mặt nào là mặt sau? Câu trả lời nằm trong phép cuộn hình (winding), hay nói cách khác là chiều bạn dùng để định nghĩa các điểm của một hình.



Hình 5.5. Hình minh họa một danh sách tọa độ được vẽ theo thứ tự ngược chiều kim đồng hồ.

Trong ví dụ này, các điểm của hình tam giác được định nghĩa theo thứ tự để chúng được vẽ ngược chiều kim đồng hồ. Thứ tự vẽ các tọa độ này định nghĩa chiều cuộn của hình. Mặc định, trong OpenGL, mặt được vẽ ngược chiều kim đồng hồ là mặt trước. Hình tam giác trong Hình 5.5 được định nghĩa để bạn nhìn

vào mặt trước của nó (như cách hiểu của OpenGL) và mặt còn lại là mặt sau.

Vì sao việc biết mặt nào của hình là mặt trước lại quan trọng? Câu trả lời liên quan đến một tính năng thường dùng của OpenGL gọi là chọn mặt (face culling). Chọn mặt là một tính năng tùy chọn cho môi trường OpenGL, cho phép quy trình xử lý đồ họa bỏ qua (không tính toán hay vẽ) mặt sau của một hình, từ đó tiết kiệm thời gian, bộ nhớ và công xử lý.

```
// kích hoạt tính năng chọn mặt
gl.glEnable(GL10.GL_CULL_FACE);
// chỉ định mặt nào sẽ không được vẽ
gl.glCullFace(GL10.GL_BACK);
```

Nếu bạn cố dùng tính năng chọn mặt mà không biết mặt nào của hình là mặt trước hay mặt sau, các đối tượng đồ họa OpenGL của bạn nhìn sẽ hơi mỏng, hoặc có thể sẽ không hề hiển thị. Vì vậy, hãy luôn định nghĩa các tọa độ cho các hình OpenGL theo thứ tự vẽ ngược chiều kim đồng hồ.

> **Lưu ý:** Bạn có thể thiết đặt môi trường OpenGL để coi mặt theo chiều kim đồng hồ là mặt trước nhưng làm vậy sẽ cần viết nhiều mã hơn và có thể gây nhầm lẫn cho các lập trình viên OpenGL kinh nghiệm khi bạn cần họ giúp đỡ. Vì vậy, bạn không nên làm điều đó.

Các phiên bản OpenGL và Sự tương thích với Thiết bị

Chuẩn OpenGL ES 1.0 và 1.1 được hỗ trợ từ Android 1.0. Từ Android 2.2 (API cấp 8), Android hỗ trợ chuẩn OpenGL ES 2.0. OpenGL ES 2.0 được hỗ trợ bởi hầu hết các thiết bị Android và được khuyên dùng cho những ứng dụng mới phát triển dùng OpenGL. OpenGL ES 3.0 được hỗ trợ ở Android 4.3 (API cấp 18) và phiên bản cao hơn, trên các thiết bị có cung cấp bản hiện thực hóa OpenGL ES 3.0 API. Để tìm hiểu về số hiệu phiên bản tương đối của các thiết bị Android có hỗ trợ một phiên bản nào đó của OpenGL ES, bạn hãy xem bài viết OpenGL ES Version Dashboard (tạm dịch: Bảng tổng kết các Phiên bản OpenGL ES).

Lập trình đồ họa với OpenGL ES 1.0/1.1 API khác nhiều so với lập trình với phiên bản 2.0 hoặc cao hơn. Phiên bản 1.x của OpenGL có nhiều phương thức tiện ích hơn và có một quy trình xử lý đồ họa cố định, trong khi OpenGL ES 2.0 và 3.0 API cung cấp quyền kiểm soát trực tiếp nhiều hơn với quy trình xử lý đồ họa thông qua OpenGL shader. Bạn nên cân nhắc kỹ các yêu cầu đồ họa và chọn phiên bản API phù hợp nhất cho ứng dụng của mình. Bạn có thể đọc phần “Lựa chọn một Phiên bản OpenGL API” để tìm hiểu thêm.

OpenGL ES 3.0 API bổ sung các tính năng và cải thiện hiệu suất so với phiên bản 2.0, đồng thời hỗ trợ tương thích ngược. Điều này nghĩa là bạn có thể viết ứng dụng hướng tới OpenGL ES 2.0 và tùy điều kiện dùng các tính năng đồ họa của OpenGL ES 3.0 nếu chúng sẵn có. Để tìm hiểu cách kiểm tra sự tồn tại của phiên bản 3.0, bạn hãy xem phần "Kiểm tra Phiên bản OpenGL ES".

Hỗ trợ nén texture

Nén texture có thể tăng đáng kể hiệu suất của ứng dụng OpenGL bằng cách giảm yêu cầu về bộ nhớ và tận dụng băng thông bộ nhớ hiệu quả hơn. Framework Android hỗ trợ định dạng nén ETC1 như một tính năng tiêu chuẩn, trong đó có lớp tiện ích ETC1Util và công cụ nén etc1tool (nằm trong Android SDK tại thư mục <sdk>/tools/). Để tham khảo một ứng dụng Android mẫu sử dụng kỹ thuật nén texture, bạn có thể xem mã của CompressedTextureActivity trong Android SDK (<sdk>/samples/<version>/ApiDemos/src/com/example/android/apis/graphics/).

> **Cảnh báo:** Định dạng ETC1 được hầu hết các thiết bị Android hỗ trợ, nhưng điều này không được đảm bảo hoàn toàn. Để kiểm tra xem định dạng ETC1 có được thiết bị hỗ trợ hay không, bạn hãy gọi phương thức ETC1Util.isETC1Supported().

> **Lưu ý:** Định dạng nén texture ETC1 không hỗ trợ texture có đặc tính trong suốt (kênh alpha). Nếu ứng dụng của bạn cần sử dụng texture có đặc tính trong suốt, bạn nên nghiên cứu các định dạng nén texture khác có trên các thiết bị đích của bạn.

Các định dạng ETC2/EAC được đảm bảo có sẵn khi bạn sử dụng OpenGL ES 3.0 API. Định dạng texture này cung cấp tỷ lệ nén ưu việt với chất lượng hình ảnh cao và cũng hỗ trợ đặc tính trong suốt (kênh alpha).

Ngoài các định dạng ETC, các thiết bị Android cung cấp sự hỗ trợ khác nhau đối với kỹ thuật nén texture tùy thuộc vào các GPU chipset và bản hiện thực hóa OpenGL. Bạn nên tìm hiểu về việc hỗ trợ nén texture trên các thiết bị đang nhắm tới để quyết định những kiểu nén nào ứng dụng của bạn nên hỗ trợ. Để quyết định các định dạng texture nào được hỗ trợ trên một thiết bị cho trước, bạn phải truy vấn thông tin của thiết bị và kiểm tra *tên các thành phần mở rộng OpenGL* vì chúng cho biết các định dạng nén texture (và các tính năng khác của OpenGL) được thiết bị hỗ trợ. Một số định dạng nén texture được hỗ trợ phổ biến là:

- **ATITC (ATC)** – Định dạng nén texture ATI (ATITC hoặc ATC) có trên rất nhiều thiết bị. Định dạng này hỗ trợ nén tốc độ cố định cho các texture RGB có hoặc không có kênh alpha. Định dạng này có thể được đại diện bằng tên một số thành phần mở rộng OpenGL như:

  - GL_AMD_compressed_ATC_texture
  - GL_ATI_texture_compression_atitc

- **PVRTC** – Định dạng nén texture PowerVR (PVRTC) có trên rất nhiều thiết bị và hỗ trợ texture dạng 2-bit và 4-bit trên một điểm ảnh có hoặc không có kênh alpha. Định dạng này được đại diện bằng tên thành phần mở rộng OpenGL sau:
  - GL_IMG_texture_compression_pvrtc
- **S3TC (DXT*n*/DXTC)** – Định dạng nén texture S3 (S3TC) có một số định dạng biến thể (DXT1 tới DXT5) và ít được hỗ trợ hơn. Định dạng này hỗ trợ texture RGB có các kênh alpha 4-bit hoặc 8-bit. Định dạng này có thể được đại diện bằng tên một số thành phần mở rộng OpenGL như sau:
  - GL_OES_texture_compression_S3TC
  - GL_EXT_texture_compression_s3tc
  - GL_EXT_texture_compression_dxt1
  - GL_EXT_texture_compression_dxt3
  - GL_EXT_texture_compression_dxt5
- **3DC** – Định dạng nén texture 3DC (3DC) là một định dạng ít thấy hơn, hỗ trợ texture RGB có một kênh alpha. Định dạng này được đại diện bởi tên thành phần mở rộng OpenGL sau:
  - GL_AMD_compressed_3DC_texture

> **Cảnh báo:** Các định dạng nén texture này *không được hỗ trợ* trên tất cả các thiết bị. Sự hỗ trợ các định dạng này khác nhau tùy theo từng nhà sản xuất và từng thiết bị. Cách xác định các định dạng nén texture có trên một thiết bị nào đó hay không được trình bày trong phần tiếp theo.

> **Lưu ý:** Khi bạn đã quyết định được các định dạng nén texture nào sẽ được hỗ trợ trong ứng dụng, hãy nhớ khai báo chúng trong file kê khai bằng phần tử <supports-gl-texture>. Khai báo này kích hoạt việc lọc ứng dụng ở các dịch vụ bên ngoài như Google Play, nhờ đó ứng dụng của bạn chỉ cài được trên những thiết bị có hỗ trợ các định dạng yêu cầu trong ứng dụng. Nội dung chi tiết có trong bài viết OpenGL manifest declarations (tạm dịch: Khai báo OpenGL trong file kê khai).

Xác định các thành phần mở rộng OpenGL

Các bản hiện thực hóa OpenGL ở các thiết bị Android khác nhau về các thành phần mở rộng OpenGL ES API được hỗ trợ. Các thành phần mở rộng này bao gồm các kiểu nén texture, nhưng cũng thường bao gồm cả các thành phần mở

rộng khác cho bộ tính năng của OpenGL.

Để xác định các định dạng nén texture và những thành phần mở rộng OpenGL khác được hỗ trợ trên một thiết bị cụ thể:

1. Bạn chạy đoạn mã sau trên các thiết bị mục tiêu để xác định các định dạng nén texture được hỗ trợ:

```
String extensions = javax.microedition.khronos.opengles.GL10.glGetString(
GL10.GL_EXTENSIONS);
```

> **Lưu ý:** Kết quả của lời gọi này *khác nhau trên mỗi đời (model) thiết bị!* Bạn phải chạy lời gọi này trên một vài thiết bị mục tiêu để xác định những kiểu nén nào được hỗ trợ phổ biến.

2. Bạn kiểm tra kết quả của phương thức này để xác định các thành phần mở rộng OpenGL nào được hỗ trợ trên thiết bị.

Kiểm tra Phiên bản OpenGL ES

Có một số phiên bản OpenGL ES được hỗ trợ trên các thiết bị Android. Bạn có thể chỉ định phiên bản API tối thiểu mà ứng dụng cần trong file kê khai, nhưng đồng thời bạn cũng có thể muốn tận dụng các tính năng trong các API mới hơn. Ví dụ, OpenGL ES 3.0 tương thích ngược với phiên bản 2.0, vì vậy bạn có thể viết ứng dụng để dùng các tính năng của OpenGL ES 3.0 nhưng dự phòng dùng bản 2.0 nếu OpenGL ES 3.0 không được thiết bị hỗ trợ.

Trước khi sử dụng các tính năng từ một phiên bản OpenGL ES cao hơn yêu cầu tối thiểu đã khai báo trong file kê khai của ứng dụng, ứng dụng cần kiểm tra phiên bản API đó có sẵn trên thiết bị hay không. Bạn có thể kiểm tra bằng một trong hai cách sau:

1. Thử tạo một OpenGL ES Context (EGLContext) của phiên bản cao hơn và kiểm tra kết quả.
2. Tạo một OpenGL ES Context của phiên bản tối thiểu được hỗ trợ, sau đó kiểm tra giá trị phiên bản.

Đoạn mã sau minh họa cách kiểm tra phiên bản OpenGL ES hiện có bằng cách tạo một đối tượng EGLContext và kiểm tra kết quả. Ví dụ này kiểm tra sự hỗ trợ phiên bản OpenGL ES 3.0:

```
private static double glVersion = 3.0;

private static class ContextFactory implements
GLSurfaceView.EGLContextFactory {

  private static int EGL_CONTEXT_CLIENT_VERSION = 0x3098;

  public EGLContext createContext(
        EGL10 egl, EGLDisplay display, EGLConfig eglConfig) {

     Log.w(TAG, "creating OpenGL ES " + glVersion + " context");
     int[] attrib_list = {EGL_CONTEXT_CLIENT_VERSION,
                          (int) glVersion, EGL10.EGL_NONE };
     // thử tạo đối tượng EGLContext của OpenGL ES 3.0
     EGLContext context = egl.eglCreateContext(
           display, eglConfig, EGL10.EGL_NO_CONTEXT, attrib_list);
     return context; // trả về null nếu phiên bản 3.0
                     // không được hỗ trợ;
  }
}
```

Nếu phương thức createContext() bên trên trả về null, bạn nên tạo một OpenGL ES Context của phiên bản 2.0 để thay thế và quay trở lại dùng API 2.0.

Đoạn mã tiếp theo minh họa cách kiểm tra phiên bản OpenGL ES bằng cách tạo một OpenGL Context của phiên bản tối thiểu được hỗ trợ, sau đó kiểm tra chuỗi chứa tên phiên bản:

```
// Tạo OpenGL ES Context của phiên bản tối thiểu được hỗ trợ, sau đó
// kiểm tra:
String version =
javax.microedition.khronos.opengles.GL10.glGetString(GL10.GL_VERSION);
Log.w(TAG, "Version: " + version );
// Định dạng của chuỗi version là:
// "OpenGL ES <Số phiên bản chính>.<Số phiên bản phụ>"
// theo sau là thông tin tùy chọn được bản hiện thực hóa OpenGL ES
// trên thiết bị cung cấp.
```

Với phương pháp này, nếu bạn phát hiện rằng thiết bị hỗ trợ phiên bản API cấp cao hơn, bạn phải hủy OpenGL ES Context tối thiểu và tạo một OpenGL ES Context mới với phiên bản API cao hơn hiện có.

Lựa chọn một Phiên bản OpenGL API

Phiên bản OpenGL ES 1.0 (và phần mở rộng 1.1), phiên bản 2.0 và phiên bản 3.0 đều cung cấp các giao diện đồ họa hiệu suất cao dành cho việc tạo các trò chơi 3D, các hình ảnh và giao diện người dùng. Lập trình đồ họa cho OpenGL ES 2.0 và ES 3.0 phần lớn giống nhau vì phiên bản 3.0 là tập cha của phiên bản 2.0 với những tính năng bổ sung. Lập trình cho OpenGL ES 1.0/1.1 API khác biệt khá lớn với OpenGL ES 2.0 và 3.0, vì vậy lập trình viên cần cân nhắc kỹ các nhân tố sau trước khi bắt đầu phát triển với các API này:

- **Hiệu suất** – Nhìn chung, OpenGL ES 2.0 và 3.0 đem lại hiệu suất đồ họa nhanh hơn các ES 1.0/1.1 API. Tuy nhiên, sự khác biệt về hiệu suất có thể thay đổi phụ thuộc vào thiết bị Android mà đang chạy ứng dụng OpenGL của bạn do có sự khác biệt về bản hiện thực hóa quy trình xử lý đồ họa OpenGL ES của các nhà sản xuất phần cứng.
- **Sự tương thích Thiết bị** - Lập trình viên nên tính đến các kiểu thiết bị, các phiên bản Android và các phiên bản OpenGL ES mà khách hàng của họ có. Để tìm hiểu về sự tương thích OpenGL trên các thiết bị, bạn hãy xem phần "Các phiên bản OpenGL và Sự tương thích Thiết bị".
- **Tiện ích Lập trình** - OpenGL ES 1.0/1.1 API cung cấp một quy trình xử lý đồ họa với chức năng cố định và các hàm tiện ích không có trong API của OpenGL ES 2.0 hay 3.0. Những lập trình viên mới làm quen với OpenGL ES có thể thấy việc lập trình cho phiên bản 1.0/1.1 nhanh hơn và tiện lợi hơn.
- **Kiểm soát Đồ họa** – OpenGL ES 2.0 API và OpenGL ES 3.0 API đem lại mức độ kiểm soát cao hơn bằng cách cung cấp một quy trình xử lý đồ họa hoàn toàn có thể lập trình được thông qua việc sử dụng shader. Với sự kiểm soát quy trình xử lý đồ họa một cách trực tiếp hơn, lập trình viên có thể tạo các hiệu ứng rất khó có thể thực hiện bằng API 1.0/1.1.
- **Hỗ trợ Texture** - OpenGL ES 3.0 có sự hỗ trợ tốt nhất cho kỹ thuật nén texture vì đảm bảo tính sẵn dùng của định dạng nén ETC2 trong đó hỗ trợ đặc tính trong suốt. Các bản hiện thực hóa API 1.x và 2.0 thường hỗ trợ ETC1, tuy nhiên định dạng texture này không hỗ trợ đặc tính trong suốt, và vì vậy bạn thường phải cung cấp các tài nguyên theo những định dạng nén khác được các thiết bị đang nhắm tới hỗ trợ. Bạn có thể đọc phần "Hỗ trợ nén texture" để tìm hiểu thêm.

Dù hiệu suất, tính tương thích, sự tiện lợi, sự kiểm soát và các nhân tố khác có thể tác động đến quyết định của bạn, bạn nên chọn phiên bản OpenGL API mà bạn nghĩ sẽ đem lại trải nghiệm tốt nhất cho người dùng.

Tăng tốc đồ họa nhờ phần cứng

NỘI DUNG BÀI HỌC

- Kiểm soát tính năng Tăng tốc đồ họa nhờ phần cứng
- Xác định xem View có được tăng tốc đồ họa nhờ phần cứng hay không
- Các mô hình vẽ của Android
  - Mô hình vẽ dựa trên phần mềm
  - Mô hình vẽ dựa trên tính năng tăng tốc đồ họa nhờ phần cứng
- Các thao tác vẽ không được hỗ trợ
- Các layer của View
  - Layer của View và Hoạt hình
- Thủ thuật

BÀI ĐỌC THÊM

- OpenGL with the Framework APIs (tạm dịch: OpenGL với các API của Framework)
- Renderscript

Bắt đầu từ Android 3.0 (API cấp 11), quy trình xử lý đồ họa 2D của Android hỗ trợ tính năng tăng tốc đồ họa nhờ phần cứng (hardware acceleration), tức là tất cả các thao tác vẽ được thực hiện trên canvas của View đều sử dụng GPU. Vì cần thêm tài nguyên để kích hoạt tính năng tăng tốc đồ họa phần cứng, ứng dụng của bạn sẽ sử dụng nhiều RAM hơn.

Mặc định, tính năng tăng tốc đồ họa nhờ phần cứng được kích hoạt nếu cấp API đích của bạn >= 14, nhưng tính năng này cũng có thể được kích hoạt thủ công. Nếu ứng dụng chỉ sử dụng các đối tượng View và Drawable tiêu chuẩn, việc bật tính năng tăng tốc đồ họa nhờ phần cứng ở mức toàn cục sẽ không gây vấn đề đồ họa nào. Tuy nhiên, vì tăng tốc đồ họa nhờ phần cứng không được hỗ trợ đối với tất cả các thao tác vẽ 2D, nên việc sử dụng tính năng này có thể ảnh hưởng đến một số View tùy chỉnh hoặc một số thao tác vẽ. Vấn đề thường biểu hiện bằng các phần tử không hiển thị, xảy ra ngoại lệ, hoặc các điểm ảnh kết xuất không đúng. Để xử lý vấn đề này, Android cho phép bạn chọn kích hoạt hay vô hiệu hóa tính năng tăng tốc đồ họa nhờ phần cứng ở nhiều mức. Nội dung chi tiết được trình bày trong phần “Kiểm soát tính năng Tăng tốc đồ họa nhờ phần cứng”.

Nếu ứng dụng của bạn thực hiện thao tác vẽ tùy chỉnh, hãy kiểm thử ứng dụng trên các thiết bị phần cứng thực với tùy chọn tăng tốc đồ họa nhờ phần cứng được bật để kiểm tra xem có lỗi gì không. Phần "Các thao tác vẽ không được hỗ trợ" trình bày các vấn đề đã biết của việc dùng tính năng tăng tốc đồ họa nhờ phần cứng và cách khắc phục.

Kiểm soát tính năng tăng tốc đồ họa nhờ phần cứng

Bạn có thể điều khiển tính năng tăng tốc đồ họa nhờ phần cứng ở các mức sau:

- Ứng dụng
- Activity
- Cửa sổ (Window)
- View

Mức Ứng dụng

Trong file kê khai Android, bạn hãy thêm thuộc tính sau vào thẻ <application> để kích hoạt tính năng tăng tốc đồ họa nhờ phần cứng cho toàn bộ ứng dụng.

```
<application android:hardwareAccelerated="true" ...>
```

Mức Activity

Nếu ứng dụng không hoạt động đúng khi kích hoạt tùy chọn tăng tốc đồ họa nhờ phần cứng ở mức toàn cục, bạn có thể điều chỉnh tính năng này cho từng Activity. Để kích hoạt hoặc vô hiệu hóa tính năng ở mức Activity, bạn có thể sử dụng thuộc tính android:hardwareAccelerated cho phần tử <activity>. Ví dụ sau kích hoạt tăng tốc đồ họa nhờ phần cứng cho toàn bộ ứng dụng nhưng vô hiệu hóa nó trong một Activity:

```
<application android:hardwareAccelerated="true">
    <activity ... />
    <activity android:hardwareAccelerated="false" />
</application>
```

Mức cửa sổ

Nếu bạn cần kiểm soát chặt hơn, bạn có thể kích hoạt tính năng tăng tốc đồ họa nhờ phần cứng cho từng cửa sổ (window) với đoạn mã sau:

```
getWindow().setFlags(
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
```

> **Lưu ý:** Hiện nay bạn không thể vô hiệu hóa tính năng tăng tốc đồ họa nhờ phần cứng ở mức cửa sổ.

Mức View

Bạn có thể vô hiệu hóa tính năng tăng tốc đồ họa nhờ phần cứng cho từng View lúc chạy với đoạn mã sau:

```
myView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

> **Lưu ý:** Hiện nay bạn không thể kích hoạt tính năng tăng tốc đồ họa nhờ phần cứng ở mức View. Ngoài khả năng vô hiệu hóa tính năng này, các layer của View còn hỗ trợ các chức năng khác. Bạn có thể đọc phần "Các layer của View" để tìm hiểu thêm.

Xác định xem View có được tăng tốc đồ họa nhờ phần cứng hay không

Đôi khi việc biết một ứng dụng có đang được tăng tốc đồ họa nhờ phần cứng hay không cũng khá quan trọng, đặc biệt là với những đối tượng như View tùy chỉnh. Và đặc biệt hữu ích nếu ứng dụng thực hiện nhiều thao tác vẽ tùy chỉnh và không phải tất cả các thao tác đều được hỗ trợ bởi quy trình xử lý đồ họa mới.

Có hai cách khác nhau để kiểm tra xem ứng dụng có đang được tăng tốc đồ họa nhờ phần cứng hay không:

- View.isHardwareAccelerated() trả về true nếu đối tượng View được gắn với một cửa sổ (window) được tăng tốc đồ họa nhờ phần cứng.
- Canvas.isHardwareAccelerated() trả về true nếu đối tượng Canvas được tăng tốc đồ họa nhờ phần cứng.

Nếu bạn phải kiểm tra trong đoạn mã thực hiện thao tác vẽ, hãy sử dụng Canvas.isHardwareAccelerated() thay vì View.isHardwareAccelerated() khi có thể. Khi một đối tượng View gắn với một cửa sổ được tăng tốc đồ họa nhờ phần cứng, đối tượng này vẫn có thể được vẽ bằng một Canvas không tăng tốc đồ họa nhờ phần cứng. Ví dụ, tình huống này xảy ra khi bạn vẽ một đối tượng View vào một bitmap vì mục đích lưu đệm (cache).

Các mô hình vẽ của Android

Khi kích hoạt tính năng tăng tốc đồ họa nhờ phần cứng, framework Android sử dụng một mô hình vẽ (drawing model) mới tận dụng các *danh sách hiển thị* (display list) để kết xuất ứng dụng lên màn hình. Để tìm hiểu về danh sách hiển thị và sự ảnh hưởng của chúng tới ứng dụng, chúng ta cần hiểu cách Android vẽ các đối tượng View khi không dùng tính năng tăng tốc đồ họa nhờ phần cứng. Những phần sau mô tả các mô hình vẽ dựa trên phần mềm và dựa trên tính năng tăng tốc đồ họa nhờ phần cứng.

Mô hình vẽ dựa trên phần mềm

Trong mô hình vẽ phần mềm, các View được vẽ với hai bước sau:

1. Hủy hiệu lực của cây phân cấp
2. Vẽ cây phân cấp

Mỗi khi ứng dụng cần cập nhật một phần UI, ứng dụng gọi invalidate() (hoặc một trong những phương thức biến thể của nó) trên bất kỳ View nào có nội dung bị thay đổi. Các thông điệp hủy hiệu lực (invalidate) được truyền lên tận cùng cây phân cấp View để tính toán các vùng màn hình cần được vẽ lại (vùng "bẩn"). Sau đó hệ thống Android vẽ bất kỳ View nào trong cây phân cấp giao cắt với vùng bẩn. Thật không may, mô hình vẽ này có hai điểm hạn chế.

- Thứ nhất, mô hình này đòi hỏi chạy rất nhiều mã với mỗi lần vẽ. Ví dụ, nếu ứng dụng gọi invalidate() trên một nút bấm (button) và nút bấm này nằm ở đầu một View khác, hệ thống Android sẽ vẽ lại View đó dù nó không có thay đổi gì.
- Hạn chế thứ hai là mô hình vẽ này có thể ẩn đi lỗi trong ứng dụng của bạn. Vì hệ thống Android vẽ lại các View khi chúng giao cắt với vùng "bẩn", một đối tượng View có nội dung thay đổi có thể được vẽ lại dù phương thức invalidate() của nó không được gọi. Khi đó, bạn phụ thuộc vào việc một View khác bị hủy hiệu lực để có được hành vi đúng. Hành vi này có thể thay đổi mỗi lần bạn sửa đổi ứng dụng. Vì vậy, bạn nên luôn gọi invalidate() trên các đối tượng View tùy chỉnh bất cứ khi nào bạn sửa đổi dữ liệu hoặc trạng thái có ảnh hưởng tới phần mã vẽ View.

> **Lưu ý:** Các đối tượng View trong Android tự động gọi invalidate() khi các thuộc tính của chúng thay đổi, ví dụ như màu nền hoặc nội dung văn bản trong một TextView.

Mô hình vẽ dựa trên tính năng tăng tốc đồ họa nhờ phần cứng

Hệ thống Android vẫn sử dụng invalidate() và draw() để yêu cầu cập nhật màn hình và để kết xuất các View, nhưng cách xử lý thao tác vẽ thực tế khác đi. Thay vì thi hành các câu lệnh vẽ ngay lập tức, hệ thống Android ghi lại chúng trong các danh sách hiển thị, danh sách này chứa đầu ra của đoạn mã vẽ cây phân cấp View. Một điểm tối ưu khác là hệ thống Android chỉ cần ghi lại và cập nhật các danh sách hiển thị cho các View được đánh dấu là bẩn bởi lời gọi invalidate(). Các View chưa bị hủy hiệu lực có thể được vẽ lại đơn giản bằng cách dùng lại danh sách hiển thị đã ghi lại trước đó. Mô hình vẽ mới gồm ba công đoạn:

1. Hủy hiệu lực của cây phân cấp
2. Ghi lại và cập nhật các danh sách hiển thị
3. Vẽ các danh sách hiển thị

Với mô hình này, bạn không thể phụ thuộc vào View giao cắt với vùng bẩn để gọi tới phương thức draw() của View. Để đảm bảo hệ thống Android ghi lại một danh sách hiển thị của một View, bạn phải gọi invalidate(). Nếu bạn quên thao tác đó, đối tượng View nhìn sẽ vẫn như cũ sau khi nó bị thay đổi.

Sử dụng các danh sách hiển thị cũng có lợi cho hiệu suất hoạt hình vì việc thiết lập một số thuộc tính cụ thể (ví dụ alpha hoặc rotation) không yêu cầu phải hủy hiệu lực đối tượng View đích (việc này được thực hiện tự động). Điểm tối ưu này cũng áp dụng cho các View sử dụng các danh sách hiển thị (toàn bộ View khi ứng dụng được tăng tốc đồ họa nhờ phần cứng). Ví dụ, giả sử có một LinearLayout chứa một ListView nằm phía trên một Button. Danh sách hiển thị cho đối tượng LinearLayout sẽ như sau:

- DrawDisplayList(ListView)
- DrawDisplayList(Button)

Giả sử giờ bạn muốn thay đổi độ mờ đục (opacity) của ListView. Sau khi gọi setAlpha(0.5f) trên ListView, danh sách hiển thị sẽ bao gồm:

- SaveLayerAlpha(0.5)
- DrawDisplayList(ListView)
- Restore
- DrawDisplayList(Button)

Đoạn mã vẽ phức tạp của ListView đã không được chạy. Thay vào đó, hệ thống chỉ cập nhật danh sách hiển thị của đối tượng LinearLayout đơn giản hơn nhiều. Trong một ứng dụng không dùng tính năng tăng tốc đồ họa nhờ phần cứng, mã vẽ của cả danh sách và đối tượng cha của nó sẽ được chạy lại.

Các thao tác vẽ không được hỗ trợ

Khi dùng tính năng tăng tốc đồ họa nhờ phần cứng, quy trình xử lý đồ họa 2D hỗ trợ các thao tác vẽ Canvas thường dùng nhất cũng như nhiều thao tác ít dùng khác. Tất cả các thao tác vẽ được dùng để kết xuất các ứng dụng cài sẵn cùng hệ điều hành Android, các widget và layout mặc định, và các hiệu ứng hình ảnh nâng cao như phản chiếu và texture dạng ô đều được hỗ trợ.

Bảng sau mô tả mức độ hỗ trợ đối với nhiều thao tác khác nhau ở các cấp API:

	Cấp API			
	< 16	16	17	18
Canvas				
drawBitmapMesh() (mảng màu sắc)	✗	✗	✗	✓
drawPicture()	✗	✗	✗	✗
drawPosText()	✗	✓	✓	✓
drawTextOnPath()	✗	✓	✓	✓
drawVertices()	✗	✗	✗	✗
setDrawFilter()	✗	✓	✓	✓
clipPath()	✗	✗	✗	✓
clipRegion()	✗	✗	✗	✓
clipRect(Region.Op.XOR)	✗	✗	✗	✓
clipRect(Region.Op.Difference)	✗	✗	✗	✓
clipRect(Region.Op.ReverseDifference)	✗	✗	✗	✓
clipRect() with rotation/perspective	✗	✗	✗	✓
Paint				
setAntiAlias() (cho văn bản)	✗	✗	✗	✓
setAntiAlias() (cho nét vẽ)	✗	✓	✓	✓
setFilterBitmap()	✗	✗	✓	✓

setLinearText()	✗	✗	✗	✗
setMaskFilter()	✗	✗	✗	✗
setPathEffect() (cho nét vẽ)	✗	✗	✗	✗
setRasterizer()	✗	✗	✗	✗
setShadowLayer() (ngoại trừ văn bản)	✗	✗	✗	✗
setStrokeCap() (cho nét vẽ)	✗	✗	✗	✓
setStrokeCap() (cho các điểm)	✗	✗	✗	✗
setSubpixelText()	✗	✗	✗	✗
Xfermode				
AvoidXfermode	✗	✗	✗	✗
PixelXorXfermode	✗	✗	✗	✗
PorterDuff.Mode.DARKEN (framebuffer)	✗	✗	✗	✗
PorterDuff.Mode.LIGHTEN (framebuffer)	✗	✗	✗	✗
PorterDuff.Mode.OVERLAY (framebuffer)	✗	✗	✗	✗
Shader				
ComposeShader bên trong ComposeShader	✗	✗	✗	✗
Các shader cùng kiểu bên trong ComposeShader	✗	✗	✗	✗
Ma trận cục bộ trên ComposeShader	✗	✗	✗	✓

Thu phóng Canvas

Quy trình xử lý đồ họa 2D hỗ trợ tăng tốc đồ họa nhờ phần cứng ban đầu được xây dựng để hỗ trợ vẽ với tỷ lệ gốc, trong đó một số thao tác vẽ gây giảm chất lượng đáng kể ở các giá trị tỷ lệ lớn hơn. Các thao tác này được viết mã dưới dạng các texture vẽ ở tỷ lệ 1.0 và được biến đổi bởi GPU. Ở cấp API < 17, sử dụng các thao tác này sẽ tạo ra các đối tượng thu phóng tăng theo tỷ lệ.

Bảng sau trình bày những phần đã có sự thay đổi để xử lý đúng các tỷ lệ lớn:

	Cấp API		
	< 17	17	18
Hỗ trợ các hệ số tỷ lệ lớn			
drawText()	✗	✗	✓
drawPosText()	✗	✗	✗
drawTextOnPath()	✗	✗	✗
Các hình Đơn giản*	✗	✓	✓
Các hình Phức tạp*	✗	✗	✗
drawPath()	✗	✗	✗
Shadow layer	✗	✗	✗

> **Lưu ý:** Các hình 'Đơn giản' gồm các lệnh drawRect(), drawCircle(), drawOval(), drawRoundRect() và drawArc() (với useCenter=false) từ một đối tượng Paint không có PathEffect, và không chứa các điểm nối (join) ngoài mặc định (thông qua setStrokeJoin() / setStrokeMiter()). Các trường hợp sử dụng khác của các lệnh vẽ này nằm trong mục Các hình 'Phức tạp' trong bảng trên.

Nếu ứng dụng của bạn bị ảnh hưởng vì sự thiếu hụt hay hạn chế của các tính năng này, bạn có thể tắt tính năng tăng tốc đồ họa nhờ phần cứng cho phần bị ảnh hưởng trong ứng dụng bằng cách gọi setLayerType(View.LAYER_TYPE_SOFTWARE, null). Như vậy, bạn vẫn có thể tận dụng tính năng này ở các phần khác. Bạn có thể xem phần "Kiểm soát việc tăng tốc đồ họa nhờ phần cứng" để tìm hiểu cách kích hoạt và vô hiệu hóa tính năng này ở các mức khác nhau trong ứng dụng của bạn.

Các layer của View

Trong tất cả phiên bản Android, các đối tượng View có khả năng kết xuất vào trong các bộ nhớ đệm ngoài màn hình (off-screen buffer) bằng cách dùng cache vẽ của View hoặc dùng Canvas.saveLayer(). Các bộ nhớ đệm ngoài màn hình, còn gọi là các layer của View, cung cấp một số chức năng. Bạn có thể sử dụng chúng để tăng hiệu suất khi tạo hoạt hình trên các View phức tạp hoặc khi thực hiện các hiệu ứng tổng hợp. Ví dụ, bạn có thể thực thi các hiệu ứng mờ dần (fade)

bằng Canvas.saveLayer() để tạm thời kết xuất View vào một layer và sau đó ghép layer trở lại màn hình với một hệ số mờ đục.

Bắt đầu từ Android 3.0 (API cấp 11), bạn có nhiều sự kiểm soát hơn đối với cách thức và thời điểm sử dụng các layer của View với phương thức View.setLayerType(). Phương thức này nhận hai tham số: kiểu layer bạn muốn sử dụng và một đối tượng Paint (không bắt buộc) để mô tả cách ghép layer đó. Bạn có thể sử dụng tham số Paint để áp dụng các bộ lọc màu (color filter), chế độ hòa trộn (blending mode) đặc biệt hoặc độ mờ đục (opacity) cho một layer. Một View có thể sử dụng một trong ba kiểu layer:

- LAYER_TYPE_NONE: View được kết xuất bình thường và không được hỗ trợ bởi bộ nhớ đệm ngoài màn hình nào. Đây là kiểu layer mặc định.
- LAYER_TYPE_HARDWARE: View được kết xuất bằng phần cứng vào một texture phần cứng (hardware texture) nếu ứng dụng được tăng tốc đồ họa nhờ phần cứng. Nếu ứng dụng không được tăng tốc đồ họa nhờ phần cứng, kiểu layer này hoạt động giống như LAYER_TYPE_SOFTWARE.
- LAYER_TYPE_SOFTWARE: View được kết xuất bằng phần mềm vào một ảnh bitmap.

Việc sử dụng kiểu layer nào phụ thuộc vào mục đích của bạn:

- **Hiệu suất:** Bạn hãy sử dụng layer phần cứng (LAYER_TYPE_HARDWARE) để kết xuất View vào một texture phần cứng. Khi View được kết xuất vào một layer, mã thực hiện thao tác vẽ của view không phải chạy cho đến khi View gọi invalidate(). Một số hiệu ứng hoạt hình, ví dụ hoạt hình alpha, có thể được áp dụng trực tiếp lên layer và GPU có thể thực hiện với hiệu suất cao.
- **Hiệu ứng hình ảnh:** Bạn hãy sử dụng layer phần cứng hoặc phần mềm (LAYER_TYPE_SOFTWARE) và một đối tượng Paint để áp dụng các hiệu ứng hình ảnh đặc biệt cho View. Ví dụ, bạn có thể vẽ một View theo dạng đen trắng bằng cách sử dụng ColorMatrixColorFilter.
- **Sự tương thích:** Bạn hãy sử dụng kiểu layer phần mềm để bắt buộc View kết xuất bằng phần mềm. Nếu một View đang được tăng tốc đồ họa nhờ phần cứng (ví dụ, nếu toàn bộ ứng dụng của bạn được tăng tốc đồ họa nhờ phần cứng) gặp vấn đề khi kết xuất, đây là cách dễ dàng để tránh các hạn chế của quy trình xử lý đồ họa bằng phần cứng.

Layer của View và hoạt hình

Các layer phần cứng có thể tạo hoạt hình nhanh và mượt hơn nếu ứng dụng

của bạn được tăng tốc đồ họa nhờ phần cứng. Chạy một hoạt hình ở tốc độ 60 khung hình trên giây không phải lúc nào cũng khả thi khi đối tượng là các View phức tạp với rất nhiều thao tác vẽ. Trở ngại này được giảm bớt nhờ việc sử dụng các layer phần cứng để kết xuất View vào một texture phần cứng. Texture phần cứng này có thể được dùng sau đó để tạo hiệu ứng hoạt hình trên View, loại bỏ nhu cầu tự vẽ lại liên tục của View trong quá trình chạy hoạt hình. View không được vẽ lại trừ khi bạn thay đổi các thuộc tính của view khiến hệ thống gọi invalidate(), hoặc khi bạn tự gọi invalidate(). Nếu bạn đang chạy một hoạt hình trong ứng dụng và kết quả không mượt mà như mong đợi, hãy nghĩ đến việc kích hoạt các layer phần cứng trên các đối tượng View đang được tạo hoạt hình.

Khi View có hỗ trợ layer phần cứng, một số thuộc tính của view được xử lý theo cách ghép layer vào màn hình. Việc thiết lập giá trị cho các thuộc tính này sẽ đạt hiệu suất cao vì chúng không yêu cầu hủy hiệu lực và vẽ lại View. Các thuộc tính trong danh sách sau ảnh hưởng tới cách ghép layer. Gọi phương thức set của bất kỳ thuộc tính nào dưới đây sẽ mang lại cơ chế hủy hiệu lực View tối ưu và đối tượng View đích không bị vẽ lại.

- alpha: Thay đổi độ mờ đục của layer.
- x, y, translationX, translationY: Thay đổi vị trí của layer.
- scaleX, scaleY: Thay đổi kích thước của layer.
- rotation, rotationX, rotationY: Thay đổi hướng của layer trong không gian ba chiều.
- pivotX, pivotY: Thay đổi điểm chốt của layer.

Tên của các thuộc tính này được dùng khi tạo hoạt hình cho View với ObjectAnimator. Nếu bạn muốn truy cập các thuộc tính này, hãy gọi phương thức set hoặc get thích hợp. Ví dụ, để thay đổi thuộc tính alpha, bạn dùng setAlpha(). Đoạn mã sau minh họa cách hữu hiệu nhất để xoay một View trong không gian ba chiều quanh trục Y:

```
view.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator.ofFloat(view, "rotationY", 180).start();
```

Vì các layer phần cứng sử dụng bộ nhớ card màn hình (video memory), một lưu ý quan trọng là bạn chỉ nên sử dụng chúng trong thời gian chạy hoạt hình và phải vô hiệu hóa chúng sau khi hoạt hình kết thúc. Bạn có thể làm điều đó với các animation listener:

```
View.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotationY",180);
animator.addListener(new  AnimatorListenerAdapter() {
   @Override
   public void onAnimationEnd (Animator animation) {
      view.setLayerType(View.LAYER_TYPE_NONE, null);
   }
});
animator.start();
```

Để tìm hiểu thêm về property animation, bạn hãy xem phần "Property Animation".

Thủ thuật

Chuyển sang chế độ đồ họa 2D hỗ trợ tăng tốc đồ họa nhờ phần cứng có thể lập tức tăng hiệu suất, nhưng bạn vẫn cần thiết kế ứng dụng của bạn để sử dụng GPU hữu hiệu bằng cách tuân theo những chỉ dẫn sau:

Giảm số lượng View trong ứng dụng

Càng phải vẽ nhiều View, hệ thống càng chậm. Điều này cũng đúng đối với quy trình xử lý đồ họa bằng phần mềm. Giảm số lượng View là một trong các cách dễ dàng nhất để tối ưu hóa Giao diện người dùng.

Tránh vẽ quá nhiều

Đừng vẽ quá nhiều layer chồng lên nhau. Hãy loại bỏ bất kỳ View nào bị che khuất hoàn toàn bởi các View không trong suốt khác nằm trên nó. Nếu bạn cần vẽ một số layer chồng lên nhau, hãy xem xét khả năng kết hợp chúng thành một layer. Một quy tắc hữu ích với phần cứng hiện nay là không vẽ quá 2,5 lần số lượng điểm ảnh trên màn hình trong mỗi khung hình (tính cả các điểm ảnh trong suốt trong ảnh bitmap).

Đừng tạo các đối tượng kết xuất trong các phương thức vẽ

Một lỗi thường gặp là tạo mới một đối tượng Paint hoặc Path mỗi lần có phương thức kết xuất được gọi. Điều này khiến bộ dọn rác chạy nhiều hơn đồng thời còn bỏ qua cache và các tối ưu hóa trong quy trình xử lý đồ họa bằng phần cứng.

Đừng thay đổi hình quá thường xuyên

Các hình và đường vẽ phức tạp, ví dụ như hình tròn, được kết xuất nhờ các mặt nạ texture (texture mask). Mỗi lần bạn tạo hoặc thay đổi một đường vẽ, quy trình xử lý đồ họa bằng phần cứng tạo một mặt nạ mới, và thao tác này có thể tốn tài nguyên.

Đừng thay đổi các bitmap quá thường xuyên

Mỗi lần bạn thay đổi nội dung một bitmap, bitmap được tải lại dưới dạng GPU texture vào lần tiếp theo bạn vẽ nó.

Cẩn trọng khi sử dụng alpha

Khi bạn thiết lập một View bán trong suốt bằng setAlpha(), AlphaAnimation, hoặc ObjectAnimator, View được kết xuất vào một bộ đệm ngoài màn hình khiến nhân đôi tỷ lệ phủ cần thiết. Khi sử dụng alpha cho các View cỡ lớn, hãy cân nhắc thiết lập kiểu layer cho View là LAYER_TYPE_HARDWARE.

6. Thực hiện các thao tác Mạng

Bài học này giải thích các tác vụ cơ bản cần cho việc kết nối mạng, theo dõi kết nối mạng (bao gồm các thay đổi về kết nối) và cho phép người dùng kiểm soát cách sử dụng mạng của một ứng dụng. Ngoài ra, bạn cũng sẽ học được cách phân tách và sử dụng dữ liệu XML.

Trong bài học có một ứng dụng mẫu minh họa cách thực hiện các thao tác mạng thường gặp. Bạn có thể tải mã nguồn và sử dụng chúng làm nguồn để tái sử dụng trong ứng dụng của chính bạn.

Sau bài học này, bạn sẽ có kiến thức về các thành phần nền tảng để tạo các ứng dụng Android có khả năng tải nội dung và phân tách dữ liệu hiệu quả, đồng thời giảm thiểu lưu lượng mạng.

Kết nối Mạng

Bài học này giới thiệu cách viết một ứng dụng đơn giản thực hiện kết nối với mạng. Trong đó có một số kinh nghiệm thực tiễn tốt nhất bạn nên tuân theo khi tạo một ứng dụng kết nối mạng dù là nhỏ nhất.

Lưu ý rằng để thực hiện các thao tác mạng trong bài học này, file kê khai của ứng dụng phải có các quyền sau:

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

Lựa chọn HTTP Client

Hầu hết các ứng dụng Android có kết nối mạng sử dụng HTTP để gửi và nhận dữ liệu. Android cung cấp hai HTTP client: HttpURLConnection và Apache HttpClient. Cả hai đều hỗ trợ HTTPS, tải lên và tải xuống theo dòng (stream), giới hạn trễ cho phép tùy chỉnh, IPv6 và bộ trữ kết nối (connection pool). Chúng tôi khuyên dùng HttpURLConnection cho các ứng dụng chạy trên Gingerbread hoặc cao hơn. Để thảo luận thêm về chủ đề này, bạn có thể xem bài đăng trên blog Android's HTTP Client (tạm dịch: Các HTTP Client của Android).

Kiểm tra Kết nối Mạng

Trước khi kết nối mạng, ứng dụng nên kiểm tra liệu có kết nối mạng nào đang hoạt động hay không bằng phương thức getActiveNetworkInfo() và isConnected(). Bạn cần nhớ rằng thiết bị có thể nằm ngoài phạm vi mạng, hoặc người dùng có thể đã tắt cả Wi-Fi và dịch vụ truy cập dữ liệu mạng di động (mobile data access). Để thảo luận thêm về chủ đề này, bạn có thể xem phần "Quản lý Cách sử dụng Mạng".

```
public void myClickHandler(View view) {
    ...
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // truy xuất dữ liệu
    } else {
        // hiển thị lỗi
    }
    ...
}
```

Thực hiện thao tác Mạng trong luồng riêng

Các thao tác mạng có thể ẩn chứa các khoảng trễ ngoài dự tính. Để đặc điểm này không ảnh hưởng tới trải nghiệm người dùng, hãy luôn thực hiện các thao tác mạng trong luồng riêng ngoài luồng UI. Lớp AsyncTask cung cấp các cách đơn giản nhất để kích hoạt một tác vụ mới ngoài luồng UI. Để thảo luận thêm về chủ đề này, bạn có thể xem bài đăng trên blog Multithreading For Performance (tạm dịch: Đa luồng để nâng Hiệu suất).

Trong đoạn mã sau, phương thức myClickHandler() gọi new DownloadWebpageTask().execute(stringUrl). DownloadWebpageTask là lớp con của AsyncTask. DownloadWebpageTask hiện thực hóa các phương thức sau của AsyncTask:

- doInBackground() thực thi phương thức downloadUrl(). Nó truyền URL của trang web làm tham số. Phương thức downloadUrl() truy xuất và xử lý nội dung trang web. Khi hoàn thành, nó trả lại kết quả là một chuỗi.
- onPostExecute() nhận chuỗi trả về bên trên và hiển thị lên Giao diện người dùng.

```
public class HttpExampleActivity extends Activity {
   private static final String DEBUG_TAG = "HttpExample";
   private EditText urlText;
   private TextView textView;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      urlText = (EditText) findViewById(R.id.myUrl);
      textView = (TextView) findViewById(R.id.myText);
   }
   // Khi người dùng nhấn vào nút bấm, gọi AsyncTask.
   // Trước khi thử truy xuất URL, hãy đảm bảo có kết nối mạng.
   public void myClickHandler(View view) {
      // Lấy URL từ trường văn bản trên UI
      String stringUrl = urlText.getText().toString();
      ConnectivityManager connMgr = (ConnectivityManager)
         getSystemService(Context.CONNECTIVITY_SERVICE);
      NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
      if (networkInfo != null && networkInfo.isConnected()) {
         new DownloadWebpageTask().execute(stringUrl);
      } else {
         textView.setText("Hiện nay không có kết nối mạng nào.");
      }
   }
    // Sử dụng AsyncTask để tạo một tác vụ mới ngoài luồng UI chính.
    // Tác vụ này nhận vào một chuỗi URL và sử dụng chuỗi để tạo
    // HttpUrlConnection. Khi kết nối đã được thiết lập, tác vụ
    // AsyncTask tải nội dung trang web dưới dạng một InputStream.
    // Cuối cùng, InputStream này được chuyển đổi thành chuỗi và
    // được hiển thị trên UI bằng phương thức onPostExecute của AsyncTask.
    private class DownloadWebpageTask extends AsyncTask<String,
    Void, String> {
      @Override
      protected String doInBackground(String... urls) {

         // các tham số đến từ lời gọi execute(): tham số đầu
         // tiên là URL.
         try {
            return downloadUrl(urls[0]);
         } catch (IOException e) {
          // trả về chuỗi cho biết không thể truy xuất trang web.
          // URL có thể không đúng.
```

```
            return "Unable to retrieve web page. URL may be invalid.";
        }
    }
    // onPostExecute hiển thị các kết quả của AsyncTask.
    @Override
    protected void onPostExecute(String result) {
        textView.setText(result);
    }
  }
  ...
}
```

Chuỗi các sự kiện trong đoạn mã trên xảy ra như sau:

1. Khi người dùng nhấn vào nút bấm sẽ kích hoạt myClickHandler(), ứng dụng truyền URL đã được chỉ định tới DownloadWebpageTask, một lớp con của lớp AsyncTask.
2. Phương thức doInBackground() của AsyncTask gọi phương thức downloadUrl().
3. Phương thức downloadUrl() nhận một chuỗi URL làm tham số và sử dụng chuỗi này để tạo một đối tượng URL.
4. Đối tượng URL được dùng để thiết lập một HttpURLConnection.
5. Ngay khi kết nối được thiết lập, đối tượng HttpURLConnection lấy về nội dung trang web dưới dạng một InputStream (Dòng đầu vào).
6. Đối tượng InputStream được truyền tới phương thức readIt(), phương thức này chuyển đổi dòng thành một chuỗi.
7. Cuối cùng, phương thức onPostExecute() của AsyncTask hiển thị chuỗi đó lên Giao diện Người dùng trong Activity chính.

Kết nối và Tải Dữ liệu

Trong luồng mà bạn dùng cho các giao dịch mạng, bạn có thể sử dụng HttpURLConnection để thực hiện một yêu cầu GET và tải về dữ liệu. Sau khi gọi connect(), bạn có thể lấy một InputStream chứa dữ liệu bằng cách gọi phương thức getInputStream().

Trong đoạn mã tiếp theo, phương thức doInBackground() gọi phương thức downloadUrl(). Phương thức downloadUrl() nhận vào một chuỗi URL và sử dụng chuỗi này để kết nối mạng thông qua đối tượng HttpURLConnection. Ngay

khi kết nối được thiết lập, ứng dụng sử dụng phương thức getInputStream() để lấy dữ liệu dưới dạng một InputStream.

```
// Với URL được cung cấp, thiết lập một HttpUrlConnection và truy xuất
// nội dung trang web dưới dạng một InputStream, sau đó trả về dưới
// dạng chuỗi
private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
    // Chỉ hiển thị 500 ký tự đầu tiên của nội dung trang web nhận được
    int len = 500;

   try {
       URL url = new URL(myurl);
       HttpURLConnection conn =
         (HttpURLConnection) url.openConnection();
       conn.setReadTimeout(10000 /* mili giây */);
       conn.setConnectTimeout(15000 /* mili giây */);
       conn.setRequestMethod("GET");
       conn.setDoInput(true);
       // Bắt đầu truy vấn
       conn.connect();
       int response = conn.getResponseCode();
       Log.d(DEBUG_TAG,"The response is: "+ response);
       is = conn.getInputStream();

       // Chuyển đổi InputStream thành chuỗi
       String contentAsString = readIt(is, len);
       return contentAsString;

    // Đảm bảo đóng InputStream sau khi ứng dụng đã dùng xong
    } finally {
       if (is!= null) {
          is.close();
       }
    }
}
```

> Lưu ý rằng phương thức getResponseCode() trả về mã trang thái (status code) của kết nối. Đây là cách hữu ích để lấy thêm thông tin về kết nối. Mã trạng thái 200 cho biết kết nối thành công.

Chuyển đổi InputStream thành Chuỗi

InputStream là nguồn dạng byte có thể đọc được. Khi có một InputStream, thông thường bạn cần phải giải mã hoặc chuyển đổi nó thành một kiểu dữ liệu đích. Ví dụ, nếu bạn đang tải về dữ liệu ảnh, bạn có thể giải mã và hiển thị InputStream như sau:

```
InputStream is = null;
...
Bitmap bitmap = BitmapFactory.decodeStream(is);
ImageView imageView = (ImageView) findViewById(R.id.image_view);
imageView.setImageBitmap(bitmap);
```

Trong ví dụ trước đó, InputStream biểu diễn nội dung văn bản của một trang web. Đây là ví dụ cách chuyển đổi InputStream thành một chuỗi để Activity có thể hiển thị trên Giao diện người dùng:

```
// Đọc một InputStream và chuyển đổi nó thành một Chuỗi
public String readIt(InputStream stream, int len) throws
IOException, UnsupportedEncodingException {
    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

Quản lý Cách sử dụng Mạng

NỘI DUNG BÀI HỌC

- Kiểm tra Kết nối Mạng của Thiết bị
- Quản lý Cách sử dụng Mạng
- Viết một Activity tùy chỉnh cá nhân
- Phản hồi lại các thay đổi của một tùy chỉnh
- Phát hiện những thay đổi về Kết nối

BÀI ĐỌC THÊM

Bài học này mô tả cách viết ứng dụng có khả năng kiểm soát chặt chẽ cách sử dụng các tài nguyên mạng. Nếu ứng dụng của bạn thực hiện rất nhiều thao tác mạng, bạn nên cung cấp các thiết lập người dùng cho phép họ kiểm soát thói quen sử dụng dữ liệu của ứng dụng, ví dụ như ứng dụng đồng bộ dữ liệu bao lâu một lần, ứng dụng có thực hiện tải lên/tải xuống chỉ khi dùng Wi-Fi hay không, ứng dụng có sử dụng dữ liệu khi chuyển vùng (roaming) hay không… Với các tùy chọn kiểm soát này, ít xảy ra khả năng người dùng tắt tính năng truy xuất dữ liệu ngầm của ứng dụng của bạn khi họ gần tới giới hạn sử dụng dữ liệu, vì thay vào đó họ có thể điều khiển chính xác lượng dữ liệu ứng dụng sử dụng.

Để tìm hiểu những hướng dẫn chung về cách viết ứng dụng để giảm thiểu tác động tới thời lượng pin do thao tác tải xuống và kết nối mạng, bạn hãy xem bài viết Optimizing Battery Life (tạm dịch: Tối ưu Thời lượng Pin) và Transferring Data Without Draining the Battery (tạm dịch: Truyền tải Dữ liệu mà không hút cạn Pin).

Kiểm tra Kết nối Mạng của Thiết bị

Một thiết bị có thể có nhiều kiểu kết nối mạng. Bài học này tập trung vào việc sử dụng kết nối Wi-Fi hoặc kết nối mạng di động. Để biết danh sách đầy đủ của các kiểu kết nối mạng, bạn hãy xem ConnectivityManager.

Wi-Fi có tốc độ cao hơn. Hơn nữa, dữ liệu mạng di động thường được đo đếm và có thể tốn nhiều tiền. Chiến lược phổ biến cho các ứng dụng là chỉ truy xuất dữ liệu lớn nếu có kết nối Wi-Fi.

Trước khi thực hiện các thao tác mạng, một thói quen tốt là kiểm tra trạng thái của kết nối mạng. Điều này giúp ứng dụng tránh việc vô tình sử dụng sai kết nối. Nếu không có một kết nối mạng khả dụng, ứng dụng nên có cách phản hồi thích hợp. Để kiểm tra kết nối mạng, bạn thường cần dùng các lớp sau:

- ConnectivityManager: Trả lời các truy vấn về trạng thái của kết nối mạng. Nó cũng thông báo cho các ứng dụng khi kết nối mạng thay đổi.
- NetworkInfo: Mô tả trạng thái của một giao diện mạng thuộc một kiểu xác định (hiện nay là kiểu Mạng di động hoặc Wi-Fi).

Đoạn mã sau kiểm tra kết nối mạng Wi-Fi và mạng di động. Đoạn mã xác định sự tồn tại của các giao diện mạng này (tức là kiểu kết nối mạng có hỗ trợ trên thiết bị hay không), và/hoặc tình trạng kết nối (tức là kết nối mạng có hoạt động hay không và có khả năng thiết lập socket và truyền dữ liệu hay không):

```
private static final String DEBUG_TAG = "NetworkStatusExample";
...
ConnectivityManager connMgr = (ConnectivityManager)
      getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo =
      connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

> Lưu ý rằng bạn không nên quyết định dựa trên cơ sở một mạng nào đó "khả dụng" hay không. Bạn nên luôn kiểm tra isConnected() trước khi thực hiện các thao tác mạng, vì isConnected() xử lý các trường hợp như tín hiệu mạng di động chập chờn, chế độ máy bay và chế độ hạn chế dữ liệu chạy ngầm.

Một cách khác ngắn gọn hơn để kiểm tra có giao diện mạng nào khả dụng hay không như sau: Phương thức getActiveNetworkInfo() trả về một thể hiện NetworkInfo đại diện cho giao diện mạng đã kết nối đầu tiên mà nó tìm được, hoặc null nếu không giao diện mạng nào được kết nối (tức là không có kết nối Internet nào tồn tại):

```
public boolean isOnline() {
   ConnectivityManager connMgr = (ConnectivityManager)
         getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
   return (networkInfo != null && networkInfo.isConnected());
}
```

Để truy vấn trạng thái cụ thể hơn bạn có thể sử dụng NetworkInfo.DetailedState, nhưng nhu cầu này hiếm khi cần thiết.

Quản lý cách sử dụng Mạng

Bạn có thể viết một PreferenceActivity (Activity cung cấp tùy chỉnh cá nhân cho Người dùng) cho phép người dùng kiểm soát cách sử dụng các tài nguyên mạng của ứng dụng. Ví dụ:

- Bạn có thể cho phép người dùng đăng tải video chỉ khi thiết bị đang kết nối với mạng Wi-Fi.

- Bạn có thể thực hiện đồng bộ (hoặc không) dựa trên các tiêu chí cụ thể như tính khả dụng của mạng, tần suất …

Để viết một ứng dụng hỗ trợ truy xuất mạng và quản lý cách sử dụng mạng, file kê khai ứng dụng phải có các quyền và bộ lọc Intent thích hợp.

- Đoạn trích file kê khai bên dưới chứa các quyền sau:
  - android.permission.INTERNET—Cho phép ứng dụng mở các socket mạng.
  - android.permission.ACCESS_NETWORK_STATE—Cho phép ứng dụng truy xuất thông tin về các mạng.
- Bạn có thể khai báo bộ lọc Intent cho action ACTION_MANAGE_NETWORK_USAGE (được giới thiệu trong Android 4.0) để thông báo rằng ứng dụng có một Activity cho phép người dùng điều chỉnh cách sử dụng dữ liệu. ACTION_MANAGE_NETWORK_USAGE hiển thị các thiết lập liên quan đến việc quản lý việc sử dụng dữ liệu mạng của một ứng dụng cụ thể. Khi ứng dụng có một Activity cho phép người dùng điều chỉnh cách sử dụng mạng, bạn nên khai báo bộ lọc Intent này cho Activity đó. Trong ứng dụng mẫu, action này được xử lý bởi lớp SettingsActivity, lớp này hiển thị một Giao diện Người dùng dạng tùy chỉnh cá nhân để người dùng quyết định khi nào tải một dữ liệu feed.

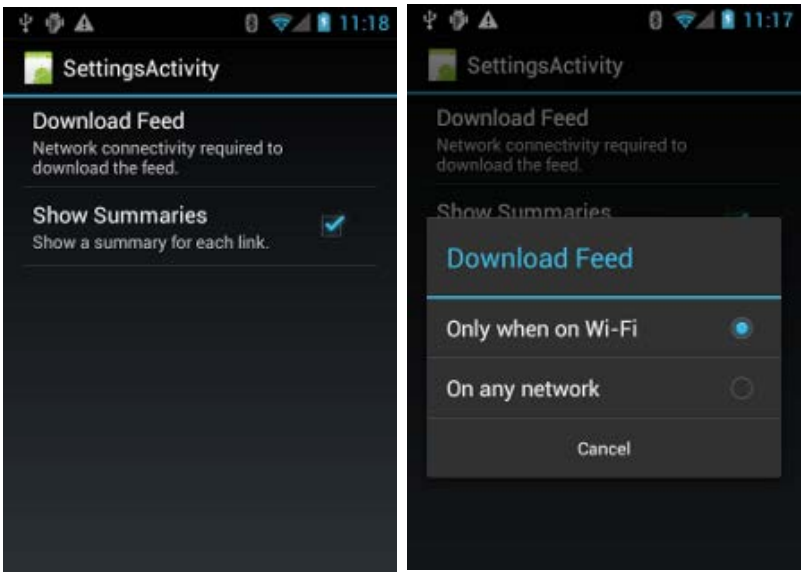
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.networkusage"
   ...>

   <uses-sdk android:minSdkVersion="4"
        android:targetSdkVersion="14" />

   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
   <application
      ...>
      ...
      <activity android:label="SettingsActivity"
                android:name=".SettingsActivity">
         <intent-filter>
           <action android:name=
                  "android.intent.action.MANAGE_NETWORK_USAGE" />
```

```
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
      </activity>
    </application>
</manifest>
```

Viết một Activity tùy chỉnh cá nhân

Như bạn có thể thấy trong đoạn trích file kê khai ở trên, SettingsActivity có một bộ lọc Intent cho action ACTION_MANAGE_NETWORK_USAGE. SettingsActivity là lớp con của PreferenceActivity. Activity này hiển thị một màn hình tùy chỉnh cá nhân (như trong Hình 6.1) cho phép người dùng lựa chọn:

- Có hiển thị phần tóm tắt (summaries) cho mỗi mục feed XML hay không, hay chỉ hiển thị liên kết đến mỗi mục.
- Có tải feed XML với bất cứ kết nối mạng nào khả dụng hay không, hay chỉ với kết nối Wi-Fi.



Hình 6.1. Activity tùy chỉnh cá nhân.

Sau đây là lớp SettingsActivity. Lưu ý rằng lớp này kế thừa OnSharedPreferenceChangeListener. Khi người dùng thay đổi một tùy chọn, onSharedPreferenceChanged() sẽ được gọi và gán refreshDisplay bằng true. Thao tác này khiến màn hình nạp lại dữ liệu khi người dùng trở lại Activity chính:

```
public class SettingsActivity extends PreferenceActivity
implements OnSharedPreferenceChangeListener {

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

      // Nạp file XML chứa các tùy chỉnh cá nhân
      addPreferencesFromResource(R.xml.preferences);
   }
   @Override
   protected void onResume() {
      super.onResume();

      // Đăng ký một listener cho sự kiện một tùy chỉnh bất kỳ
      // thay đổi
      getPreferenceScreen().getSharedPreferences().registerOnShar
edPreferenceChangeListener(this);
   }

   @Override
   protected void onPause() {
      super.onPause();

     // Hủy đăng ký listener đã tạo trong onResume().
     // Hủy đăng ký listener là một thói quen tốt khi ứng dụng của
     // bạn không sử dụng chúng nhằm giảm chi phí hệ thống không cần
     // thiết. Bạn có thể làm việc này trong OnPause().
     getPreferenceScreen().getSharedPreferences().unregisterOnSh
aredPreferenceChangeListener(this);
   }


   // Khi người dùng thay đổi một lựa chọn tùy chỉnh,
   // onSharedPreferenceChanged() khởi động lại Activity chính dưới
   // dạng một tác vụ mới. Gán refreshDisplay bằng "true" để
   // thông báo rằng Activity chính nên cập nhật dữ liệu hiển thị.
   // Activity sẽ truy xuất PreferenceManager để lấy về các
   // thiết lập mới nhất.

   @Override
   public void onSharedPreferenceChanged(SharedPreferences
   sharedPreferences, String key) {
      // Đặt refreshDisplay bằng true để khi người dùng trở lại
      // Activity chính màn hình cập nhật lại dữ liệu để hiển thị
      // các thiết lập mới.
      NetworkActivity.refreshDisplay = true;
   }
}
```

Phản hồi các thay đổi của một tùy chỉnh

Khi người dùng thay đổi các tùy chỉnh ở màn hình cài đặt (settings), thường sẽ có các tác động lên hành vi của ứng dụng. Trong đoạn mã sau, ứng dụng kiểm tra các thiết lập tùy chỉnh trong onStart(). Nếu có sự trùng khớp giữa thông tin thiết lập và kết nối mạng của thiết bị (ví dụ, nếu thiết lập là "Wi-Fi" và thiết bị đang có kết nối Wi-Fi), ứng dụng sẽ tải dữ liệu feed và làm mới màn hình hiển thị.

```
public class NetworkActivity extends Activity {
    public static final String WIFI = "Wi-Fi";
    public static final String ANY = "Any";
    private static final String URL =
      "http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest";

    // Trạng thái kết nối Wi-Fi
    private static boolean wifiConnected = false;
    // Trạng thái kết nối Mạng di động
    private static boolean mobileConnected = false;
    // Màn hình có nên được cập nhật hay không
    public static boolean refreshDisplay = true;

    // Thiết lập hiện tại cho tùy chỉnh mạng của người dùng
    public static String sPref = null;

    // BroadcastReceiver với nhiệm vụ theo dõi các thay đổi kết
   // nối mạng
    private NetworkReceiver receiver = new NetworkReceiver();

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

        // Đăng ký BroadcastReceiver để theo dõi các thay đổi kết
        // nối mạng.
        IntentFilter filter =
          new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
        receiver = new NetworkReceiver();
        this.registerReceiver(receiver, filter);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Hủy đăng ký BroadcastReceiver khi ứng dụng bị hủy.
```

```
        if (receiver != null) {
            this.unregisterReceiver(receiver);
        }
    }

    // Cập nhật màn hình hiển thị nếu kết nối mạng và
    // các thiết lập tùy chỉnh của người dùng cho phép.

    @Override
    public void onStart () {
        super.onStart();

        // Lấy các thiết lập về tùy chỉnh mạng của người dùng
        SharedPreferences sharedPrefs =
          PreferenceManager.getDefaultSharedPreferences(this);

        // Truy xuất một giá trị dạng chuỗi của các tùy chỉnh. Tham
        // số thứ hai là giá trị mặc định để sử dụng nếu không tìm
        // thấy giá trị của tùy chỉnh.

        sPref = sharedPrefs.getString("listPref", "Wi-Fi");
        updateConnectedFlags();
        if (refreshDisplay) {
            loadPage();
        }
    }

    // Kiểm tra kết nối mạng và tùy theo đó gán giá trị biến
    // wifiConnected và mobileConnected
    public void updateConnectedFlags() {
        ConnectivityManager connMgr = (ConnectivityManager)
              getSystemService(Context.CONNECTIVITY_SERVICE);

        NetworkInfo activeInfo = connMgr.getActiveNetworkInfo();
        if (activeInfo != null && activeInfo.isConnected()) {
            wifiConnected = activeInfo.getType() ==
                                    ConnectivityManager.TYPE_WIFI;
            mobileConnected = activeInfo.getType() ==
                                    ConnectivityManager.TYPE_MOBILE;
        } else {
            wifiConnected = false;
            mobileConnected = false;
```

```
        }
    }
    // Sử dụng một lớp con của AsyncTask để tải dữ liệu feed XML từ
    // stackoverflow.com.
    public void loadPage() {
        if (((sPref.equals(ANY)) && (wifiConnected || mobileConnected))
                || ((sPref.equals(WIFI)) && (wifiConnected))) {
            // Lớp con của AsyncTask
            new DownloadXmlTask().execute(URL);
        } else {
            showErrorPage();
        }
    }
...

}
```

Phát hiện những thay đổi về Kết nối

Mảnh ghép cuối cùng của bài toán là NetworkReceiver, một lớp con của BroadcastReceiver. Khi kết nối mạng của thiết bị thay đổi, NetworkReceiver chặn action CONNECTIVITY_ACTION xác định trạng thái kết nối mạng, rồi theo đó gán cờ wifiConnected và mobileConnected là true hoặc false. Kết quả cuối cùng là khi lần tới người dùng trở lại ứng dụng, ứng dụng sẽ chỉ tải dữ liệu feed mới nhất và cập nhật màn hình nếu NetworkActivity.refreshDisplay bằng true.

Thiết lập một BroadcastReceiver để nhận lời gọi một cách không cần thiết có thể làm cạn tài nguyên hệ thống. Ứng dụng mẫu đăng ký NetworkReceiver trong onCreate() và hủy đăng ký trong onDestroy(). Cách làm này gọn nhẹ hơn cách khai báo một đối tượng <receiver> trong file kê khai. Khi bạn khai báo <receiver> trong file kê khai, hệ thống có thể đánh thức ứng dụng của bạn bất cứ lúc nào, ngay cả khi bạn đã không dùng nó trong nhiều tuần. Bằng cách đăng ký và hủy đăng ký NetworkReceiver trong Activity chính, bạn đảm bảo rằng ứng dụng sẽ không bị đánh thức sau khi người dùng rời khỏi ứng dụng. Nếu bạn khai báo một <receiver> trong file kê khai và biết chính xác bạn cần gì, bạn có thể dùng setComponentEnabledSetting() để kích hoạt và vô hiệu hóa receiver khi thích hợp.

Đây là phần mã của lớp NetworkReceiver:

```
public class NetworkReceiver extends BroadcastReceiver {

@Override
public void onReceive(Context context, Intent intent) {
   ConnectivityManager conn = (ConnectivityManager)
      context.getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo networkInfo = conn.getActiveNetworkInfo();

   // Kiểm tra các tùy chỉnh cá nhân của người dùng và kết nối
   // mạng. Dựa trên kết quả này, quyết định cập nhật màn hình hiển
   // thị hoặc giữ nguyên. Nếu tùy chỉnh của người dùng là chỉ sử
   // dụng Wi-Fi, kiểm tra xem thiết bị có kết nối Wi-Fi hay không.
   if (WIFI.equals(sPref) && networkInfo != null
      && networkInfo.getType()== ConnectivityManager.TYPE_WIFI) {
      // Nếu thiết bị có kết nối Wi-Fi, gán refreshDisplay
      // bằng true. Biến này chỉ thị màn hình sẽ được cập nhật khi
      // người dùng trở lại ứng dụng.
      refreshDisplay = true;
      Toast.makeText(context, R.string.wifi_connected,
                                 Toast.LENGTH_SHORT).show();

   // Nếu thiết lập đã chọn là bất cứ mạng nào (ANY) và hiện có
   // một kết nối mạng (theo phép loại trừ sẽ là mạng di động), gán
   // refreshDisplay bằng true.
   } else if (ANY.equals(sPref) && networkInfo != null) {
      refreshDisplay = true;

   // Trái lại, ứng dụng sẽ không tải nội dung – vì không có kết nối
   // mạng nào (mạng di động hoặc Wi-Fi), hoặc vì thiết lập là WIFI,
   // trong khi không có kết nối Wi-Fi. Gán refreshDisplay bằng false.
   } else {
      refreshDisplay = false;
      Toast.makeText(context, R.string.lost_connection,
                                  Toast.LENGTH_SHORT).show();
   }
}
```

Phân tách dữ liệu XML

NỘI DUNG BÀI HỌC

- Lựa chọn Bộ phân tách
- Phân tích Dữ liệu feed

- Khởi tạo Bộ phân tách
- Đọc Dữ liệu feed
- Phân tách XML
- Bỏ qua các Thẻ Bạn Không Dùng đến
- Sử dụng Dữ liệu XML

Ngôn ngữ Đánh dấu Mở rộng (Extensible Markup Language - XML) là một bộ các quy tắc mã hóa văn bản theo dạng máy có thể đọc được. XML là định dạng phổ biến để chia sẻ dữ liệu trên Internet. Các website thường xuyên cập nhật nội dung như các trang tin tức hoặc blog thường cung cấp dữ liệu feed dạng XML để các chương trình bên ngoài có thể theo dõi được những thay đổi về nội dung. Việc tải lên và phân tách dữ liệu XML là tác vụ thường gặp với các ứng dụng kết nối mạng. Bài học này giải thích cách phân tách các tài liệu XML và sử dụng dữ liệu trong đó.

Lựa chọn Bộ phân tách

Chúng tôi khuyên dùng XmlPullParser vì đây là một giải pháp hiệu quả và dễ bảo trì cho việc phân tách XML trên Android. Cho đến này Android có hai lựa chọn kế thừa giao diện này:

- KXmlParser thông qua XmlPullParserFactory.newPullParser().
- ExpatPullParser thông qua Xml.newPullParser().

Cả hai lựa chọn đều tốt. Ví dụ trong phần này sử dụng ExpatPullParser thông qua Xml.newPullParser().

Phân tích Dữ liệu feed

Bước đầu tiên trong việc phân tách một dữ liệu feed là quyết định những trường thông tin nào bạn quan tâm. Bộ phân tách sẽ trích xuất dữ liệu từ các trường đó và bỏ qua những thông tin còn lại.

Đây là một trích đoạn từ dữ liệu feed sẽ được phân tách trong ứng dụng mẫu. Mỗi bài viết gửi lên StackOverflow.com xuất hiện trong dữ liệu feed dưới dạng một thẻ entry lồng ngoài một số thẻ khác:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
xmlns:creativeCommons="http://backend.userland.com/creativeCommonsRssModule"
...">
<title type="text">newest questions tagged android - Stack Overflow
</title>
...
   <entry>
   ...
   </entry>
   <entry>
      <id>http://stackoverflow.com/q/9439999</id>
      <re:rank scheme="http://stackoverflow.com">0</re:rank>
      <title type="text">Where is my data file?</title>
      <category scheme="http://stackoverflow.com/feeds/tag?tagnames
=android&sort=newest/tags"term="android" />
      <category scheme="http://stackoverflow.com/feeds/tag?tagnames
=android&sort=newest/tags"term="file" />
      <author>
         <name>cliff2310</name>
         <uri>http://stackoverflow.com/users/1128925</uri>
      </author>
      <link rel="alternate"
    href=http://stackoverflow.com/questions/9439999/where-is-my-data-file />
      <published>2012-02-25T00:30:54Z</published>
      <updated>2012-02-25T00:30:54Z</updated>
      <summary type="html">
         <p>I have an Application that requires a data file...</p>
      </summary>
   </entry>
   <entry>
   ...
   </entry>
...
</feed>
```

Ứng dụng mẫu trích xuất dữ liệu từ thẻ entry (bản ghi) và một số thẻ nằm lồng trong đó gồm title (tiêu đề), link (liên kết) và summary (tóm tắt).

Khởi tạo Bộ phân tách

Bước tiếp theo là khởi tạo một bộ phân tách và bắt đầu quá trình phân tách. Trong đoạn mã mẫu này, bộ phân tách được khởi tạo và được chỉ thị không

xử lý các không gian tên (namespace) và nhận đối tượng InputStream làm đầu vào. Quá trình phân tách bắt đầu bằng lời gọi phương thức nextTag(), sau đó kích hoạt phương thức readFeed() trích xuất và xử lý dữ liệu ứng dụng muốn có:

```
public class StackOverflowXmlParser  {
   // Ứng dụng không sử dụng không gian tên
   private static final String ns = null;

   public List parse(InputStream in) throws
                  XmlPullParserException, IOException  {
      try {
          XmlPullParser parser = Xml.newPullParser();
          parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,
                                                              false);
          parser.setInput(in, null);
          parser.nextTag();
          return readFeed(parser);
       } finally {
          in.close();
       }
   }
 ...
}
```

Đọc Dữ liệu feed

Phương thức readFeed() không thực sự làm công việc xử lý dữ liệu feed, readFeed() tìm các phần tử được gán thẻ "entry" làm điểm khởi đầu để xử lý đệ quy dữ liệu feed. Nếu gặp một thẻ không phải entry, phương thức này sẽ bỏ qua thẻ đó. Khi toàn bộ dữ liệu feed đã được xử lý đệ quy, readFeed() trả về một List chứa các phần tử entry (bao gồm những thành phần dữ liệu nằm bên trong đó) được trích xuất từ dữ liệu feed. Sau đó đối tượng List này được trả về bởi bộ phân tách ở trên.

```
private List readFeed(XmlPullParser parser) throws
XmlPullParserException, IOException {
   List entries = new ArrayList();

   parser.require(XmlPullParser.START_TAG, ns, "feed");
   while (parser.next() != XmlPullParser.END_TAG) {
      if (parser.getEventType() != XmlPullParser.START_TAG) {
         continue;
      }
```

```
        String name = parser.getName();
        // Bắt đầu bằng việc tìm thẻ entry
        if (name.equals("entry")) {
            entries.add(readEntry(parser));
        } else {
            skip(parser);
        }
    }
    return entries;
}
```

Phân tách XML

Các bước để phân tách một dữ liệu feed dạng XML như sau:

1. Như trình bày trong phần "Phân tích Dữ liệu feed", bạn cần xác định các thẻ muốn dùng cho ứng dụng. Ví dụ này trích xuất dữ liệu từ thẻ entry và một số thẻ nằm lồng trong đó gồm title, link và summary.
2. Tạo các phương thức sau:
    - Phương thức "read" cho mỗi thẻ bạn quan tâm. Ví dụ, readEntry(), readTitle() ... Bộ phân tách đọc các thẻ từ dòng dữ liệu đầu vào (input stream). Khi gặp một thẻ có tên entry, title, link hoặc summary, bộ phân tách gọi các phương thức tương ứng cho thẻ đó. Trái lại, thẻ đó sẽ bị bỏ qua.
    - Các phương thức để trích xuất dữ liệu cho mỗi kiểu thẻ khác nhau và để đẩy bộ phân tách tới thẻ kế tiếp. Ví dụ:
        - Với thẻ title và summary, bộ phân tách gọi readText(). Phương thức này trích xuất dữ liệu cho các thẻ này bằng cách gọi parser.getText().
        - Với thẻ link, bộ phân tách trích xuất dữ liệu cho các liên kết bằng cách trước tiên, xác định liên kết có thuộc kiểu ứng dụng quan tâm không. Sau đó, bộ phân tách sử dụng parser.getAttributeValue() để trích xuất giá trị của liên kết.
        - Với thẻ entry, bộ phân tách gọi readEntry(). Phương thức này phân tách các thẻ lồng trong thẻ entry và trả về một đối tượng Entry với các phần tử dữ liệu title, link và summary.
    - Phương thức bổ trợ skip() có tính đệ quy. Để thảo luận thêm về chủ đề này, bạn có thể xem phần "Bỏ qua các Thẻ Bạn Không Dùng đến".

Đoạn mã sau cho bạn thấy cách bộ phân tách làm việc với các phần tử entry, title, link và summary.

```
public static class Entry {
    public final String title;
    public final String link;
    public final String summary;

    private Entry(String title, String summary, String link) {
        this.title = title;
        this.summary = summary;
        this.link = link;
    }
}

// Phân tách nội dung của một đối tượng entry. Nếu gặp thẻ title,
// summary, hoặc link, chuyển thẻ đó tới phương thức "read" tương
// ứng để xử lý. Trái lại, bỏ qua thẻ đó.
private Entry readEntry(XmlPullParser parser) throws
XmlPullParserException, IOException {

    parser.require(XmlPullParser.START_TAG, ns, "entry");
    String title = null;
    String summary = null;
    String link = null;
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        if (name.equals("title")) {
            title = readTitle(parser);
        } else if (name.equals("summary")) {
            summary = readSummary(parser);
        } else if (name.equals("link")) {
            link = readLink(parser);
        } else {
            skip(parser);
        }
    }
    return new Entry(title, summary, link);
}
```

```
// Xử lý các thẻ title trong dữ liệu feed
private String readTitle(XmlPullParser parser) throws IOException,
XmlPullParserException {
    parser.require(XmlPullParser.START_TAG, ns, "title");
    String title = readText(parser);
    parser.require(XmlPullParser.END_TAG, ns, "title");
    return title;
}
// Xử lý các thẻ link trong dữ liệu feed
private String readLink(XmlPullParser parser) throws IOException,
XmlPullParserException {
    String link ="";
    parser.require(XmlPullParser.START_TAG, ns, "link");
    String tag = parser.getName();
    String relType = parser.getAttributeValue(null, "rel");
    if (tag.equals("link")) {
        if (relType.equals("alternate")) {
            link = parser.getAttributeValue(null, "href");
            parser.nextTag();
        }
    }
    parser.require(XmlPullParser.END_TAG, ns, "link");
    return link;
}
// Xử lý các thẻ summary trong dữ liệu feed
private String readSummary(XmlPullParser parser) throws
IOException, XmlPullParserException {
    parser.require(XmlPullParser.START_TAG, ns, "summary");
    String summary = readText(parser);
    parser.require(XmlPullParser.END_TAG, ns, "summary");
    return summary;
}
// Với các thẻ title và summary, trích xuất giá trị văn bản của chúng
private String readText(XmlPullParser parser) throws IOException,
XmlPullParserException {
    String result = "";
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.getText();
        parser.nextTag();
    }
    return result;
}
  ...
}
```

Bỏ qua các Thẻ Bạn Không Dùng đến

Một trong những bước phân tách XML đã trình bày ở trên đó là bộ phân tách bỏ qua các thẻ ứng dụng không quan tâm tới. Đoạn mã sau minh họa phương thức skip() của bộ phân tách:

```
private void skip(XmlPullParser parser) throws
XmlPullParserException, IOException {
   if (parser.getEventType() != XmlPullParser.START_TAG) {
      throw new IllegalStateException();
   }
   int depth =1;
   while (depth !=0) {
      switch (parser.next()) {
      case XmlPullParser.END_TAG:
         depth--;
         break;
      case XmlPullParser.START_TAG:
         depth++;
         break;
      }
   }
 }
```

Phương thức skip() hoạt động cụ thể như sau:

- Ném ra một ngoại lệ nếu sự kiện hiện tại không phải là START_TAG (thẻ mở).
- Xử lý sự kiện START_TAG và tất cả các sự kiện lồng trong cho đến sự kiện END_TAG (Thẻ đóng) tương ứng.
- Để đảm bảo phương thức dừng ở thẻ END_TAG đúng và không phải ở thẻ đầu tiên gặp phải sau thẻ START_TAG gốc, phương thức theo dõi độ sâu các tầng lồng nhau.

Nhờ đó, nếu phần tử hiện tại có các các phần tử lồng trong, giá trị biến depth sẽ không phải là 0 cho đến khi bộ phân tách đã xử lý tất cả các sự kiện nằm giữa START_TAG gốc và END_TAG tương ứng. Hãy xem ví dụ về cách bộ phân tách bỏ qua phần tử <author> với hai phần tử lồng trong là <name> và <uri>:

- Trong lần lặp đầu tiên của vòng lặp while, thẻ tiếp theo bộ phân tách gặp phải sau thẻ <author> là thẻ START_TAG của phần tử <name>. Giá trị depth được tăng lên thành 2.

- Trong lần lặp thứ hai, thẻ tiếp theo bộ phân tách gặp là thẻ END_TAG </name>. Giá trị depth giảm còn 1.
- Trong lần lặp thứ ba, thẻ tiếp theo bộ phân tách gặp là thẻ START_TAG <uri>. Giá trị depth được tăng lên thành 2.
- Trong lần lặp thứ tư, thẻ tiếp theo bộ phân tách gặp là thẻ END_TAG </uri>. Giá trị depth giảm còn 1.
- Trong lần lặp thứ năm (lần cuối), thẻ tiếp theo bộ phân tách gặp là thẻ END_TAG </author>. Giá trị depth giảm còn 0, cho biết phần tử <author> đã được bỏ qua hoàn toàn.

Sử dụng Dữ liệu XML

Ứng dụng mẫu truy xuất và phân tách dữ liệu feed XML trong một tác vụ không đồng bộ AsyncTask. Điều đó giúp tách việc xử lý ra khỏi luồng UI. Khi việc xử lý hoàn tất, ứng dụng cập nhật Giao diện Người dùng trong Activity chính (NetworkActivity).

Trong đoạn trích dưới đây, phương thức loadPage() làm những việc sau:

- Khởi tạo một biến dạng chuỗi với URL của dữ liệu feed XML.
- Nếu những thiết lập của người dùng và kết nối mạng cho phép, phương thức này sẽ gọi new DownloadXmlTask().execute(url). Thao tác này khởi tạo một đối tượng DownloadXmlTask mới (lớp con của AsyncTask) và chạy phương thức execute() của lớp này, phương thức execute() tải và phân tách dữ liệu feed và trả về kết quả dạng chuỗi để hiển thị lên Giao diện Người dùng.

```
public class NetworkActivity extends Activity {
    public static final String WIFI = "Wi-Fi";
    public static final String ANY = "Any";
    private static final String URL =
    "http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest";

    // Trạng thái kết nối Wifi.
    private static boolean wifiConnected = false;
    // Trạng thái kết nối Mạng di động.
    private static boolean mobileConnected = false;
    // Có nên cập nhật lại màn hình hiển thị hay không
    public static boolean refreshDisplay = true;
    public static String sPref = null;
```

```
...

// Sử dụng AsyncTask để tải dữ liệu feed XML từ stackoverflow.com.
public void loadPage() {

   if ((sPref.equals(ANY)) && (wifiConnected || mobileConnected)) {
      new DownloadXmlTask().execute(URL);

   }
   else if ((sPref.equals(WIFI)) && (wifiConnected)) {
      new DownloadXmlTask().execute(URL);
   } else {
      // hiển thị lỗi
   }
}
```

Lớp con DownloadXmlTask của AsyncTask được minh họa bên dưới, lớp DownloadXmlTask hiện thực hóa các phương thức sau của lớp AsyncTask:

- doInBackground() chạy phương thức loadXmlFromNetwork(). Phương thức doInBackground() truyền URL của dữ liệu feed làm tham số. Phương thức loadXmlFromNetwork() truy xuất và xử lý dữ liệu feed. Khi hoàn thành, loadXmlFromNetwork() trả lại kết quả là một chuỗi.
- onPostExecute() nhận chuỗi trả về bên trên và hiển thị lên Giao diện người dùng.

```
// Phần mã kế thừa lớp AsyncTask dùng để tải dữ liệu feed XML từ
// stackoverflow.com
private class DownloadXmlTask extends AsyncTask<String, Void, String> {
   @Override
   protected String doInBackground(String... urls) {
      try {
         return loadXmlFromNetwork(urls[0]);
      } catch (IOException e) {
         return getResources().getString(R.string.connection_error);
      } catch (XmlPullParserException e) {
         return getResources().getString(R.string.xml_error);
      }
   }

   @Override
   protected void onPostExecute(String result) {
      setContentView(R.layout.main);
      // Hiển thị chuỗi HTML lên Giao diện Người dùng thông qua
      // đối tượng WebView
```

```
        WebView myWebView = (WebView) findViewById(R.id.webview);
        myWebView.loadData(result, "text/html", null);
    }
}
```

Dưới đây là phương thức loadXmlFromNetwork() được gọi từ DownloadXmlTask. Nhiệm vụ của phương thức này như sau:

1. Khởi tạo một đối tượng StackOverflowXmlParser. Tạo các biến gồm: một đối tượng List (entries) chứa các đối tượng Entry; title, url và summary chứa các giá trị trích xuất từ dữ liệu feed XML tương ứng với các trường đó.
2. Gọi downloadUrl() để truy xuất dữ liệu feed và trả về một đối tượng InputStream.
3. Sử dụng StackOverflowXmlParser để phân tách InputStream. StackOverflowXmlParser lưu dữ liệu vào một đối tượng List chứa các Entry (entries) với dữ liệu từ dữ liệu feed.
4. Xử lý đối tượng List (entries) chứa các Entry, và kết hợp dữ liệu feed với mã HTML.
5. Trả lại một chuỗi HTML để sau đó hiển thị trên Giao diện Người dùng trong Activity chính bởi phương thức onPostExecute() của lớp AsyncTask.

```
// Tải dữ liệu XML từ stackoverflow.com, phân tách kết quả, và kết
// hợp với mã HTML. Trả về chuỗi HTML.
private String loadXmlFromNetwork(String urlString) throws
XmlPullParserException, IOException {
    InputStream stream = null;
    // Khởi tạo bộ phân tách
    StackOverflowXmlParser stackOverflowXmlParser =
                                new StackOverflowXmlParser();
    List<Entry> entries = null;
    String title = null;
    String url = null;
    String summary = null;
    Calendar rightNow = Calendar.getInstance();
    DateFormat formatter = new SimpleDateFormat("MMM dd h:mmaa");

    // Kiểm tra xem người dùng có thiết lập tùy chỉnh lấy thông tin
    // phần tóm tắt hay không
    SharedPreferences sharedPrefs =
            PreferenceManager.getDefaultSharedPreferences(this);
    boolean pref = sharedPrefs.getBoolean("summaryPref", false);
    StringBuilder htmlString = new StringBuilder();
    htmlString.append("<h3>" +
      getResources().getString(R.string.page_title) + "</h3>");
```

```
    htmlString.append("<em>" +
      getResources().getString(R.string.updated) + " " +
          formatter.format(rightNow.getTime()) + "</em>");

    try {
       stream = downloadUrl(urlString);
       entries = stackOverflowXmlParser.parse(stream);
    // Đảm bảo là InputStream được đóng sau khi ứng dụng sử dụng
    // xong nó
    } finally {
       if (stream != null) {
          stream.close();
       }
     }

    // StackOverflowXmlParser trả về một đối tượng List (có tên
    // "entries") chứa các đối tượng Entry. Mỗi đối tượng Entry đại
    // diện một bài viết trong dữ liệu feed XML. Phần này xử lý danh
    // sách các Entry để kết hợp mỗi Entry với mã HTML. Mỗi Entry
    // được hiển thị trên Giao diện Người dùng dưới dạng đường liên
    // kết với phần tóm tắt (không bắt buộc).
    for (Entry entry : entries) {
       htmlString.append("<p><a href='");
       htmlString.append(entry.link);
       htmlString.append("'>" + entry.title + "</a></p>");
       // Nếu người dùng thiết lập tùy chỉnh hiển thị phần tóm tắt,
       // thêm tóm tắt vào chuỗi HTML để hiển thị.
       if (pref) {
          htmlString.append(entry.summary);
       }
    }
    return htmlString.toString();
}

// Với một chuỗi dạng URL, thiết lập một kết nối và lấy về
// một dòng đầu vào
private InputStream downloadUrl(String urlString) throws
IOException {
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* mili giây */);
    conn.setConnectTimeout(15000 /* mili giây */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Bắt đầu câu truy vấn
    conn.connect();
    return conn.getInputStream();
}
```

7. Đồng bộ dữ liệu lên Đám mây

Với bộ API mạnh cho việc kết nối Internet, framework Android giúp bạn xây dựng những ứng dụng giàu tính năng hỗ trợ đám mây với khả năng đồng bộ dữ liệu tới một dịch vụ web từ xa, đảm bảo tất cả các thiết bị của bạn luôn được đồng bộ, và những dữ liệu đáng giá luôn được sao lưu trên đám mây.

Bài học này trình bày những chiến lược khác nhau cho các ứng dụng hỗ trợ đám mây, trong đó có đồng bộ dữ liệu với đám mây sử dụng ứng dụng web của chính bạn làm back-end[1], và sao lưu dữ liệu với đám mây để người dùng có thể khôi phục dữ liệu của họ khi cài đặt ứng dụng trên một thiết bị mới.

NỘI DUNG BÀI HỌC

Sử dụng Backup API

Học cách tích hợp Backup API (API Sao lưu) vào một ứng dụng Android để dữ liệu người dùng như các tùy chỉnh cá nhân, ghi chú, điểm kỷ lục được cập nhật thông suốt trên tất cả các thiết bị của một người dùng.

Tận dụng Tối đa Google Cloud Messaging

Học cách gửi tin nhắn multicast (cùng lúc tới nhiều người nhận) một cách hiệu quả, phản ứng thông minh với các tin nhắn tới từ Google Cloud Messaging (GCM), và sử dụng các tin nhắn GCM để đồng bộ hiệu quả với server.

Sử dụng Backup API

NỘI DUNG BÀI HỌC

- Đăng ký Dịch vụ Android Backup
- Cấu hình file kê khai
- Viết Backup Agent
- Yêu cầu Sao lưu
- Khôi phục dữ liệu từ một bản sao lưu

[1] *Back-end: Theo cách hiểu thông dụng là bộ phận xử lý dữ liệu ở phía sau, ngược với Front-end - bộ phận nhận dữ liệu đầu vào ở phía trước.*

BÀI ĐỌC THÊM

- Data Backup (tạm dịch: Sao lưu dữ liệu)

Khi một người dùng mua một thiết bị mới hoặc cài đặt lại một thiết bị có sẵn, họ mong đợi rằng khi Google Play khôi phục ứng dụng trở lại thiết bị của họ trong bước cài đặt ban đầu cũng đồng thời làm cho dữ liệu trước đó của ứng dụng khôi phục trở lại. Theo mặc định, điều đó không xảy ra và những thành quả hay thiết lập của người dùng trong ứng dụng sẽ bị mất.

Đối với các tình huống lượng dữ liệu tương đối nhẹ (ít hơn một megabyte), như các tùy chỉnh cá nhân của người dùng, ghi chú, điểm kỷ lục của trò chơi, hoặc những số liệu thống kê khác, Backup API là một giải pháp gọn nhẹ. Bài học này hướng dẫn từng bước tích hợp Backup API vào ứng dụng, và cách khôi phục dữ liệu vào các thiết bị mới với Backup API.

Đăng ký Dịch vụ Android Backup

Bài học này cần dùng tới Dịch vụ Android Backup và bạn cần đăng ký ở đây. Sau khi đăng ký, dịch vụ sẽ sinh sẵn cho bạn một thẻ XML để chèn vào file kê khai Android giống như sau:

```
<meta-data android:name="com.google.android.backup.api_key"
android:value="ABcDe1FGHij2KlmN3oPQRs4TUvW5xYZ" />
```

> Lưu ý rằng mỗi khóa sao lưu làm việc với một tên gói (package) cụ thể. Nếu bạn có nhiều ứng dụng, hãy đăng ký khóa khác nhau cho mỗi ứng dụng.

Cập nhật file kê khai

Để sử dụng Dịch vụ Android Backup, bạn cần bổ sung hai mục nữa vào file kê khai của ứng dụng. Trước tiên, bạn cần khai báo tên của lớp sẽ hoạt động với vai trò Backup Agent, và sau đó thêm đoạn mã sau vào trong thẻ Application. Giả sử Backup Agent của bạn có tên là TheBackupAgent, file kê khai sẽ như sau:

```
<application android:label="MyApp"
          android:backupAgent="TheBackupAgent">
   ...
   <meta-data android:name="com.google.android.backup.api_key"
   android:value="ABcDe1FGHij2KlmN3oPQRs4TUvW5xYZ" />
   ...
</application>
```

Viết Backup Agent

Cách dễ nhất để tạo một Backup Agent là kế thừa lớp BackupAgentHelper. Việc tạo lớp bổ trợ này thực ra rất đơn giản. Bạn chỉ cần tạo một lớp có tên giống với tên đã dùng trong file kê khai ở bước trước (trong ví dụ này là TheBackupAgent), và sau đó kế thừa lớp BackupAgentHelper. Sau đó, bạn hãy ghi đè phương thức onCreate().

Trong phương thức onCreate(), bạn hãy tạo một BackupHelper. Có nhiều lớp kế thừa BackupHelper chuyên biệt cho việc sao lưu các loại dữ liệu khác nhau. Framework Android hiện nay có hai lớp bổ trợ như vậy: FileBackupHelper và SharedPreferencesBackupHelper. Sau khi bạn tạo lớp bổ trợ và trỏ nó tới dữ liệu muốn sao lưu, bạn chỉ cần thêm lớp này vào BackupAgentHelper bằng phương thức addHelper() với một khóa để truy xuất dữ liệu về sau. Trong hầu hết các trường hợp toàn bộ mã của lớp kế thừa có thể chỉ cần 10 dòng mã.

Sau đây là ví dụ sao lưu một file điểm kỷ lục.

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.FileBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
   // Tên của file SharedPreferences
   static final String HIGH_SCORES_FILENAME = "scores";

   // Khóa định danh duy nhất cho tập dữ liệu sao lưu
   static final String FILES_BACKUP_KEY = "myfiles";

   // Cấp phát một đối tượng bổ trợ và thêm nó vào Backup Agent
   @Override
   void onCreate() {
      FileBackupHelper helper =
                new FileBackupHelper(this, HIGH_SCORES_FILENAME);
      addHelper(FILES_BACKUP_KEY, helper);
   }
}
```

Ngoài ra, phương thức khởi tạo của FileBackupHelper có thể nhận nhiều tên file. Bạn có thể dễ dàng sao lưu cả file điểm kỷ lục và file tiến trình trò chơi bằng cách thêm một tham số nữa như sau:

```
@Override
void onCreate() {
    FileBackupHelper helper = new FileBackupHelper(this,
                        HIGH_SCORES_FILENAME, PROGRESS_FILENAME);
    addHelper(FILES_BACKUP_KEY, helper);
}
```

Việc sao lưu các tùy chỉnh cá nhân của người dùng cũng đơn giản tương tự. Bạn tạo một SharedPreferencesBackupHelper giống cách tạo FileBackupHelper. Trong trường hợp này, thay vì cung cấp tên các file cho phương thức khởi tạo, bạn cung cấp tên các nhóm tùy chỉnh chia sẻ đang được sử dụng bởi ứng dụng của bạn. Đây là ví dụ về một lớp BackupAgentHelper trong trường hợp các điểm kỷ lục được lưu dưới dạng các tùy chỉnh cá nhân của người dùng thay vì trong một file.

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.SharedPreferencesBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    // Tên của các nhóm SharedPreferences được dùng trong ứng dụng.
    // Đây chính là các chuỗi được truyền cho phương thức
    // getSharedPreferences(String, int).
    static final String PREFS_DISPLAY = "displayprefs";
    static final String PREFS_SCORES = "highscores";
    // Một chuỗi bất kỳ dùng trong mã BackupAgentHelper để
    // xác định dữ liệu của SharedPreferencesBackupHelper
    static final String MY_PREFS_BACKUP_KEY = "myprefs";

    // Cấp phát một đối tượng bổ trợ và cài đặt đối tượng
    void onCreate() {
        SharedPreferencesBackupHelper helper =
                new SharedPreferencesBackupHelper(this,
                                        PREFS_DISPLAY, PREFS_SCORES);
        addHelper(MY_PREFS_BACKUP_KEY, helper);
    }
}
```

Bạn có thể thêm bao nhiêu đối tượng bổ trợ vào Backup Agent tùy ý, nhưng lưu ý rằng bạn chỉ cần một cho mỗi loại. Một FileBackupHelper xử lý tất cả các file bạn cần sao lưu, và một SharedPreferencesBackupHelper xử lý tất cả các nhóm tùy chỉnh chia sẻ bạn cần sao lưu.

Yêu cầu Sao lưu

Để yêu cầu tạo một bản sao lưu, bạn chỉ cần tạo một thể hiện của lớp BackupManager và gọi phương thức dataChanged() của lớp.

```
import android.app.backup.BackupManager;
...

public void requestBackup() {
  BackupManager bm = new BackupManager(this);
  bm.dataChanged();
}
```

Lời gọi này thông báo cho BackupManager (trình quản lý sao lưu) rằng có dữ liệu sẵn sàng để sao lưu lên đám mây. Vào một thời điểm nào đó trong tương lai, BackupManager sẽ gọi phương thức onBackup() của Backup Agent. Bạn có thể gọi phương thức này bất cứ khi nào dữ liệu thay đổi mà không phải lo lắng về việc gây ra quá nhiều hoạt động mạng. Nếu bạn yêu cầu sao lưu hai lần trước khi sự kiện sao lưu xảy ra, sẽ chỉ có một lần sao lưu.

Khôi phục dữ liệu từ một bản sao lưu

Thông thường, bạn không bao giờ cần tự yêu cầu khôi phục vì sự kiện này xảy ra tự động khi ứng dụng của bạn được cài trên một thiết bị. Tuy nhiên, nếu cần kích hoạt việc khôi phục sao lưu thủ công, bạn chỉ cần gọi phương thức requestRestore().

Tận dụng Tối đa Google Cloud Messaging

NỘI DUNG BÀI HỌC

- Gửi Tin nhắn Multicast Hiệu quả
- Thu gọn các Tin nhắn có thể bị thay thế
- Nhúng Dữ liệu trực tiếp trong Tin nhắn GCM
- Phản ứng thông minh đối với các Tin nhắn GCM

BÀI ĐỌC THÊM

- Google Cloud Messaging cho Android

Google Cloud Messaging (GCM) là một dịch vụ miễn phí phục vụ việc gửi tin nhắn tới các thiết bị Android. GCM có thể cải thiện trải nghiệm người dùng rất nhiều. Ứng dụng của bạn có thể giữ cập nhật mà không phải hao phí năng lượng pin cho việc đánh thức kết nối mạng và liên tục thăm dò server trong khi không có cập nhật mới. Đồng thời, GCM cho phép bạn gửi một tin nhắn cùng lúc tới 1.000 người nhận, nhờ đó bạn có thể dễ dàng truyền tải thông tin tới lượng người dùng lớn một cách nhanh chóng khi thích hợp, đồng thời giảm thiểu tải trên server của bạn.

Bài học này trình bày một số kinh nghiệm thực tiễn tốt nhất khi tích hợp GCM vào một ứng dụng và giả định rằng bạn đã quen thuộc với cách thức hoạt động cơ bản của dịch vụ này. Nếu chưa làm quen với GCM, bạn có thể đọc bài hướng dẫn viết ứng dụng hỗ trợ GCM.

Gửi Tin nhắn Multicast Hiệu quả

Một trong những tính năng hữu ích nhất của GCM là hỗ trợ tới 1.000 người nhận cho mỗi tin nhắn. Khả năng này giúp đơn giản hóa rất nhiều tác vụ gửi những tin nhắn quan trọng tới toàn thể người dùng. Ví dụ, giả sử bạn có một tin nhắn cần gửi tới 1.000.000 người dùng, và server của bạn có thể xử lý và gửi đi khoảng 500 tin nhắn mỗi giây. Nếu bạn gửi mỗi tin nhắn tới chỉ một người nhận, bạn sẽ cần 1.000.000/500 = 2.000 giây, khoảng nửa giờ. Tuy nhiên, thiết lập 1.000 người nhận cho mỗi tin nhắn, tổng thời gian cần để gửi thông điệp tới 1.000.000 người nhận chỉ còn (1.000.000/1.000) / 500 = 2 giây. Khả năng này không chỉ hữu ích, mà còn quan trọng đối với những dữ liệu có giá trị tức thời, ví dụ cảnh báo thảm họa thiên nhiên hoặc điểm số thể thao, trong đó khoảng thời gian 30 phút có thể khiến thông tin trở nên vô nghĩa.

Việc tận dụng tính năng này rất đơn giản. Nếu bạn đang sử dụng thư viện GCM helper cho Java, bạn chỉ cần cung cấp một List các ID đăng ký cho phương thức send hoặc sendNoRetry, thay vì một ID đăng ký đơn lẻ.

```
// Lấy danh sách người nhận
List regIds = whoShouldISendThisTo(message);

// Nếu bạn muốn SDK tự động cố gửi lại một số lần nhất định,
// hãy sử dụng phương thức send tiêu chuẩn.
MulticastResult result = sender.send(message, regIds, 5);

// Ngược lại, hãy sử dụng sendNoRetry.
MulticastResult result = sender.sendNoRetry(message, regIds);
```

Để sử dụng GCM với các ngôn ngữ khác Java, bạn cần tạo một yêu cầu HTTP POST với phần header như sau:

- Authorization: key=YOUR_API_KEY
- Content-type: application/json

Sau đó, mã hóa các tham số bạn muốn vào một đối tượng JSON, liệt kê tất cả các ID đăng ký trong thuộc tính registration_ids. Đoạn mã sau là một ví dụ. Tất cả các tham số đều không bắt buộc ngoại trừ registration_ids, và các phần tử bên trong tham số data đại diện cho thông tin cần truyền tải do người dùng tự định nghĩa, không phải là tham số do GCM định nghĩa. Điểm cuối của tin nhắn HTTP POST sẽ là https://android.googleapis.com/gcm/send.

```
{ "collapse_key": "score_update",
  "time_to_live": 108,
  "delay_while_idle": true,
  "data": {
     "score": "4 x 8",
     "time": "15:16.2342"
  },
  "registration_ids":["4", "8", "15", "16", "23", "42"]
}
```

Để tìm hiểu kỹ hơn về định dạng của tin nhắn multicast trong GCM, bạn hãy đọc phần Sending Messages trong tài liệu hướng dẫn sử dụng GCM.

Thu gọn gọn các Tin nhắn có thể bị thay thế

Mỗi tin nhắn GCM báo cho ứng dụng di động biết cần kết nối với server để làm mới dữ liệu. Trong GCM, bạn có thể (và được khuyên dùng) tạo các tin nhắn có thể thu gọn trong tình huống có tin nhắn mới thay thế các tin nhắn cũ. Hãy cùng xem xét ví dụ về điểm số thể thao. Nếu bạn gửi một tin nhắn với điểm số hiện tại tới tất cả người dùng đang theo dõi một trận đấu nào đó, và 15 phút sau đó lại có một tin nhắn khác cập nhật điểm số mới hơn, tin nhắn trước đó không còn quan trọng nữa. Với bất cứ người dùng nào chưa nhận được tin nhắn thứ nhất, không có lý do gì để gửi cả hai tin nhắn cho họ buộc thiết bị phải phản ứng (có thể là thông báo người dùng) hai lần trong khi chỉ một trong hai tin nhắn thực sự quan trọng.

Nếu bạn khai báo khóa thu gọn (collapseKey), khi có nhiều tin nhắn cho cùng một người dùng đang trong hàng đợi ở các server GCM, chỉ có tin nhắn cuối cùng với khóa thu gọn cho trước được gửi đi. Đối với các tình huống như điểm

số thể thao, tính năng này giúp thiết bị tránh xử lý những công việc không cần thiết và có thể tránh thông báo quá nhiều tới người dùng. Đối với các tình huống có tác vụ đồng bộ với server (như kiểm tra thư điện tử), tính năng này giúp giảm số lần đồng bộ thiết bị phải thực hiện. Ví dụ, nếu có 10 thư đang chờ trên server, và mười tin nhắn GCM "có thư mới" được gửi tới thiết bị. Thực tế, ứng dụng chỉ cần một vì một lần đồng bộ là đủ.

Để sử dụng tính năng này, bạn chỉ cần thêm khóa thu gọn vào tin nhắn gửi đi. Nếu bạn đang dùng thư viện GCM helper, hãy dùng phương thức collapseKey(String key) của lớp Message.

```
Message message = new Message.Builder(regId)
    .collapseKey("game4_scores") // Khóa cho trận đấu 4.
    .ttl(600) // Thời gian tính theo giây để giữ tin nhắn trong
              // hàng đợi nếu thiết bị không trực tuyến
    .delayWhileIdle(true) // Chờ thiết bị hoạt động trở lại trước
                          // khi gửi
    .addPayload("key1", "value1")
    .addPayload("key2", "value2")
    .build();
```

Nếu bạn không dùng thư viện bổ trợ, bạn chỉ cần thêm một biến vào header của yêu cầu POST, với tên biến là collapse_key, và giá trị là chuỗi bạn dùng cho bộ các tin nhắn cập nhật.

Nhúng Dữ liệu trực tiếp trong Tin nhắn GCM

Thông thường, các tin nhắn GCM chỉ đơn giản báo hiệu cho thiết bị biết rằng có dữ liệu mới đang chờ ở một server nào đó. Tuy nhiên, một tin nhắn GCM có thể chứa tới 4kb, vì vậy đôi khi có lý khi gửi cả dữ liệu bên trong tin nhắn GCM, nhờ đó thiết bị không cần kết nối lại với server nữa. Phương án này đáng xem xét nếu tất cả các điều kiện sau là đúng:

- Toàn bộ dữ liệu gói gọn trong 4kb.
- Mọi tin nhắn đều quan trọng và nên được lưu giữ.
- Không hợp lý khi thu gọn nhiều tin nhắn GCM thành một thông báo duy nhất kiểu "có dữ liệu mới trên server".

Ví dụ, các tin nhắn ngắn hoặc các bước di chuyển của người chơi trong một trò chơi theo lượt qua mạng là ví dụ tốt cho dạng dữ liệu phù hợp để nhúng trực tiếp vào tin nhắn GCM. Thư điện tử là ví dụ không tốt, vì các tin nhắn thường sẽ lớn

hơn 4kb, và người dùng không cần nhận một tin nhắn GCM cho mỗi thư điện tử mới có trên server.

Bạn cũng nên xem xét phương án này khi gửi các tin nhắn multicast, vì bạn không muốn tất cả thiết bị của người dùng gửi yêu cầu cập nhật tới server của bạn cùng một lúc.

Phương án này cũng không thích hợp để gửi lượng lớn dữ liệu vì một số lý do:

- Giới hạn lưu lượng được áp dụng để ngăn chặn việc phát tán tin rác tới một thiết bị nào đó từ các ứng dụng đáng ngờ hoặc được viết không tốt.
- Không đảm bảo các tin nhắn sẽ đến nơi theo đúng thứ tự.
- Không đảm bảo các tin nhắn sẽ đến nơi nhanh như khi bạn gửi chúng đi.

Khi sử dụng một cách thích hợp, việc nhúng trực tiếp dữ liệu vào tin nhắn GCM có thể cải thiện cảm nhận về tốc độ của ứng dụng nhờ loại bỏ một chu trình hỏi đáp tới server.

Phản ứng thông minh đối với các Tin nhắn GCM

Ứng dụng của bạn không chỉ cần phản ứng với các tin nhắn GCM mà còn phải phản ứng *thông minh.* Cách phản ứng của ứng dụng phụ thuộc vào từng hoàn cảnh.

Tránh làm phiền người dùng

Khi bạn muốn thông báo cho người dùng có dữ liệu mới, ứng dụng rất dễ vượt qua ranh giới từ "hữu ích" sang "phiền toái". Nếu ứng dụng dùng thông báo ở thanh trạng thái, hãy cập nhật thông báo hiện có thay vì tạo một thông báo mới. Nếu bạn báo bằng chuông hoặc rung, hãy nghĩ tới việc đặt bộ hẹn giờ. Đừng để ứng dụng thông báo nhiều hơn một lần trong một phút, như vậy dễ khiến người dùng nảy ý định gỡ ứng dụng, tắt thiết bị đi, hoặc quẳng thiết bị xuống sông.

Đồng bộ thông minh hơn, không phải chăm chỉ hơn

Khi sử dụng GCM để thông báo cho thiết bị tải dữ liệu từ server, hãy nhớ bạn có thể gửi 4kb siêu dữ liệu để giúp ứng dụng phản ứng thông minh hơn. Ví dụ, nếu bạn có một ứng dụng đọc dữ liệu feed và người dùng theo dõi 100 nguồn feed, hãy giúp thiết bị xử lý thông minh trong việc tải về những thông tin đó từ server. Dưới đây là ví dụ về các siêu dữ liệu có thể được gửi tới ứng dụng trong tin nhắn GCM và các cách xử lý của ứng dụng.

- refresh — Về cơ bản ứng dụng được chỉ thị cập nhật tất cả dữ liệu feed trong danh sách theo dõi. Mỗi một lần cập nhật, ứng dụng sẽ gửi yêu cầu dữ liệu feed tới 100 server khác nhau, hoặc nếu bạn có một nguồn tổng hợp trên server của bạn, ứng dụng sẽ gửi một yêu cầu truy xuất và nhận về dữ liệu tổng hợp của 100 nguồn feed khác nhau.
- refresh, feedID — Tốt hơn: Ứng dụng biết nguồn feed cụ thể nào cần cập nhật.
- refresh, feedID, timestamp — Tốt nhất: Nếu người dùng đã tình cờ cập nhật thủ công trước khi tin nhắn GCM đến, ứng dụng có thể so sánh mốc thời gian (timestamp) của bài viết gần nhất và quyết định *không cần làm gì cả.*

8. Truy cập Dữ liệu Contact

BÀI ĐỌC THÊM

- Kiến thức cơ bản về Content Provider
- Contacts Provider

Contacts Provider là kho lưu trữ trung tâm chứa thông tin các Contact của người dùng, bao gồm dữ liệu từ các ứng dụng quản lý Contact và và các ứng dụng mạng xã hội. Trong ứng dụng của bạn, bạn có thể truy cập trực tiếp thông tin của Contacts Provider bằng cách gọi các phương thức của ContentResolver hoặc gửi các Intent đến một ứng dụng quản lý Contact.

Bài học này tập trung vào các thao tác truy xuất danh sách Contact, hiển thị chi tiết thông tin của một Contact và chỉnh sửa Contact thông qua các Intent. Các kỹ thuật cơ bản trình bày ở đây có thể được mở rộng để thực hiện nhiều tác vụ phức tạp hơn. Ngoài ra, bài học này còn giúp bạn hiểu cấu trúc tổng thể và cách hoạt động của Contacts Provider.

NỘI DUNG BÀI HỌC

Truy xuất Danh sách Contact:

Học cách truy xuất một danh sách chứa các Contact trong đó dữ liệu trùng khớp với tất cả hoặc một phần của chuỗi tìm kiếm với những kỹ thuật sau:

- So khớp theo tên của Contact
- So khớp bất kỳ kiểu dữ liệu Contact nào
- So khớp một kiểu dữ liệu Contact cụ thể, ví dụ như số điện thoại

Truy xuất Thông tin Chi tiết của Contact:

Học cách truy xuất thông tin chi tiết của một Contact. Thông tin chi tiết là các dữ liệu như số điện thoại và địa chỉ email. Bạn có thể truy xuất tất cả thông tin chi tiết hoặc một kiểu thông tin cụ thể, ví dụ như tất cả các địa chỉ email.

Chỉnh sửa Contact bằng Intent:

Học cách chỉnh sửa một Contact bằng cách gửi một Intent tới ứng dụng People.

Hiển thị Quick Contact Badge:

Học cách hiển thị widget QuickContactBadge. Khi người dùng nhấn vào widget này, một hộp thoại mở ra hiển thị thông tin chi tiết của Contact và các nút bấm dành cho những ứng dụng có thể xử lý các thông tin đó. Ví dụ, nếu Contact có một địa chỉ email, hộp thoại trên hiển thị một nút bấm dành cho ứng dụng email mặc định.

Truy xuất một Danh sách Contact

NỘI DUNG BÀI HỌC

- Yêu cầu Quyền đọc Provider
- Tìm Contact theo Tên và Liệt kê các kết quả
- Tìm Contact theo một kiểu dữ liệu cụ thể
- Tìm Contact theo bất kỳ kiểu dữ liệu nào

BÀI ĐỌC LIÊN QUAN

- Kiến thức cơ bản về Content Provider
- Contacts Provider
- Các Loader
- Tạo Giao diện Tìm kiếm

Bạn sẽ tìm hiểu cách truy xuất một danh sách Contact, trong đó dữ liệu khớp với tất cả hoặc một phần của chuỗi tìm kiếm với những kỹ thuật sau:

So khớp với tên Contact

> Truy xuất một danh sách Contact bằng cách so khớp chuỗi tìm kiếm với tất cả hoặc một phần của tên Contact. Contacts Provider cho phép nhiều Contact có cùng tên, vì vậy kỹ thuật này có thể trả về một danh sách chứa các kết quả trùng khớp.

So khớp với một kiểu dữ liệu cụ thể trong Contact

> Truy xuất một danh sách Contact bằng cách so khớp chuỗi tìm kiếm với một kiểu dữ liệu cụ thể về thông tin chi tiết của Contact như địa chỉ email. Ví dụ, kỹ thuật này cho phép bạn liệt kê tất cả các Contact có địa chỉ email trùng khớp với chuỗi tìm kiếm.

So khớp với bất cứ kiểu dữ liệu nào trong Contact

Truy xuất một danh sách Contact bằng cách so khớp chuỗi tìm kiếm với bất kỳ kiểu dữ liệu nào, bao gồm tên, số điện thoại, địa chỉ nhà, địa chỉ email… Ví dụ, kỹ thuật này cho phép bạn dùng bất cứ kiểu dữ liệu nào để làm chuỗi tìm kiếm và sau đó liệt kê các Contact có chứa dữ liệu khớp với chuỗi tìm kiếm.

> **Lưu ý:** Tất cả ví dụ trong bài học này sử dụng CursorLoader để truy xuất dữ liệu từ Contacts Provider. CursorLoader chạy câu truy vấn trong một luồng riêng ngoài luồng UI. Điều này đảm bảo câu truy vấn không làm chậm thời gian đáp ứng của UI gây ảnh hưởng xấu tới trải nghiệm người dùng. Bạn có thể đọc thêm bài Loading Data in Background (tạm dịch: Tải dữ liệu ngầm).

Yêu cầu Quyền đọc Provider

Để thực hiện bất cứ kiểu tìm kiếm nào với Contacts Provider, ứng dụng phải có quyền READ_CONTACTS. Để yêu cầu quyền này, bạn cần thêm phần tử <uses-permission> vào file kê khai bên trong phần tử <manifest>:

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

Tìm Contact theo Tên và Liệt kê các kết quả

Kỹ thuật này so khớp một chuỗi tìm kiếm với tên của một Contact hoặc các Contact trong bảng ContactsContract.Contacts của Contacts Provider. Thông thường, bạn sẽ hiển thị các kết quả trong một ListView để cho phép người dùng thực hiện thao tác chọn trong số các Contact trùng khớp.

Định nghĩa ListView và Item layout

Để hiển thị các kết quả tìm kiếm trong một ListView, bạn cần một file layout chính định nghĩa toàn bộ các thành phần UI bao gồm ListView, và một file Item layout định nghĩa một dòng trong ListView. Ví dụ, bạn có thể tạo file layout chính res/layout/contacts_list_view.xml chứa đoạn XML sau:

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

Đoạn XML này sử dụng id định nghĩa sẵn android:id/list dành cho widget ListView.

File Item layout contacts_list_item.xml chứa đoạn XML sau:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@android:id/text1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:clickable="true" />
```

Đoạn XML này sử dụng id định nghĩa sẵn android:text1 dành cho widget TextView.

> **Lưu ý:** Bài học này không trình bày phần Giao diện Người dùng nhận chuỗi tìm kiếm từ người dùng, vì có thể bạn muốn nhận thông tin đó một cách gián tiếp. Ví dụ, bạn có thể cho phép người dùng tìm kiếm Contact có tên trùng với một chuỗi có trong một tin nhắn SMS.

Hai file layout bạn đã viết định nghĩa giao diện người dùng trên đó chứa một ListView. Bước tiếp theo là viết mã lập trình để hiển thị một danh sách Contact trên Giao diện Người dùng này.

Định nghĩa Fragment để hiển thị danh sách Contact

Để hiển thị danh sách Contact, trước tiên bạn định nghĩa một Fragment sẽ được nạp vào Activity của bạn. Sử dụng Fragment sẽ linh hoạt hơn, vì bạn có thể sử dụng một Fragment để hiển thị danh sách và Fragment thứ hai để hiển thị thông tin chi tiết của Contact mà người dùng đã chọn từ danh sách. Với cách thức này, bạn có thể kết hợp kỹ thuật trong bài học này với kỹ thuật trong phần "Truy xuất Thông tin chi tiết của Contact".

Để tìm hiểu cách sử dụng một hoặc nhiều đối tượng Fragment trong một Activity, bạn hãy đọc bài Xây dựng UI động với Fragment.

Để giúp bạn viết các câu truy vấn tới Contacts Provider, framework Android cung cấp một lớp contracts có tên ContactsContract, trong đó định nghĩa các hằng số hữu ích và các phương thức để truy cập Contacts Provider. Khi sử dụng lớp này, bạn không phải tự định nghĩa các hằng số cho các content URI, tên các bảng hoặc các cột. Để sử dụng lớp này, bạn cần thêm câu lệnh sau:

```
import android.provider.ContactsContract;
```

Vì lớp Fragment sử dụng CursorLoader để truy xuất dữ liệu từ Contacts Provider, bạn phải chỉ định lớp này kế thừa giao diện LoaderManager.LoaderCallbacks. Ngoài ra, để phát hiện Contact nào người dùng đã chọn từ danh sách các kết quả tìm kiếm, bạn cần kế thừa giao diện AdapterView.OnItemClickListener. Ví dụ :

```
...
import android.support.v4.app.Fragment;
import android.support.v4.app.LoaderManager.LoaderCallbacks;
import android.widget.AdapterView;
...
public class ContactsFragment extends Fragment implements
      LoaderManager.LoaderCallbacks<Cursor>,
      AdapterView.OnItemClickListener {
```

Định nghĩa các biến toàn cục

Định nghĩa các biến toàn cục sẽ dùng trong chương trình:

```
    ...
    /*
     * Định nghĩa một mảng chứa tên các cột để chuyển từ
     * Cursor sang ListView.
     */
    @SuppressLint("InlinedApi")
    private final static String[] FROM_COLUMNS = {
          Build.VERSION.SDK_INT
                >= Build.VERSION_CODES.HONEYCOMB ?
                Contacts.DISPLAY_NAME_PRIMARY :
                Contacts.DISPLAY_NAME
    };
    /*
     * Định nghĩa một mảng chứa id của các View trong layout
     * sẽ nhận nội dung của các cột trong Cursor. Id này được định
     * nghĩa sẵn trong framework Android, vì vậy nó bắt đầu bằng
     * android.R.id"
     */
    private final static int[] TO_IDS = {
         android.R.id.text1
    };
    // Định nghĩa các biến toàn cục có thể thay đổi
    // Định nghĩa một đối tượng ListView
    ListView mContactsList;
    // Định nghĩa các biến dành cho Contact người dùng chọn
    // Giá trị _ID của Contact
    long mContactId;
    // LOOKUP_KEY của Contact
    String mContactKey;
```

```
// Content URI của Contact được chọn
Uri mContactUri;
// Adapter dùng để ràng buộc (bind) Cursor chứa kết quả vào ListView
private SimpleCursorAdapter mCursorAdapter;
...
```

Lưu ý: Vì Contacts.DISPLAY_NAME_PRIMARY yêu cầu Android 3.0 (API cấp 11) hoặc cao hơn, thiết lập minSdkVersion của ứng dụng bằng 10 hoặc thấp hơn sẽ sinh một cảnh báo Android Lint[1] trong Eclipse dùng ADK. Để tắt cảnh báo này, bạn cần thêm chú thích @SuppressLint("InlinedApi") trước phần khai báo FROM_COLUMNS.

Khởi tạo Fragment

Khởi tạo Fragment. Tạo một phương thức khởi tạo public rỗng cần cho hệ thống Android và khai triển UI của đối tượng Fragment trong phương thức callback onCreateView(). Ví dụ:

```
// Phương thức khởi tạo public rỗng cần cho hệ thống
public ContactsFragment() {}
// Fragment phải khai triển đối tượng View của nó
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
    // Khai triển layout của Fagment
    return inflater.inflate(R.layout.contacts_list_layout,
                                        container, false);
}
```

Thiết lập CursorAdapter cho ListView

Tạo SimpleCursorAdapter để ràng buộc các kết quả tìm kiếm vào ListView. Để lấy đối tượng ListView sẽ hiển thị các Contact, bạn cần gọi Activity.findViewById() từ Activity cha của đối tượng Fragment. Hãy sử dụng Context của Activity cha khi bạn gọi setAdapter(). Ví dụ:

```
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    ...
    // Lấy ListView từ danh sách View của Activity cha
    mContactsList = (ListView)
        getActivity().findViewById(R.layout.contact_list_view);
```

[1] *Một công cụ trong Bộ công cụ Phát triển Android với chức năng phát hiện các lỗi tiềm ẩn trong mã nguồn của các dự án Android.*

```
        // Lấy một CursorAdapter
        mCursorAdapter = new SimpleCursorAdapter(
                getActivity(),
                R.layout.contact_list_item,
                null,
                FROM_COLUMNS, TO_IDS,
                0);
        // Gán adapter cho ListView
        mContactsList.setAdapter(mCursorAdapter);
    }
```

Thiết lập listener cho sự kiện chọn Contact

Khi hiển thị các kết quả tìm kiếm, bạn thường sẽ muốn cho phép người dùng lựa chọn một Contact để dùng cho thao tác tiếp theo. Ví dụ, khi người dùng nhấn vào một Contact, bạn có thể hiển thị địa chỉ của Contact trên bản đồ. Để cung cấp tính năng này, trước tiên bạn cần định nghĩa Fragment hiện tại làm listener lắng nghe sự kiện click (nhấn chạm của người dùng) bằng cách kế thừa giao diện AdapterView.OnItemClickListener như đã trình bày trong phần "Định nghĩa Fragment để hiển thị danh sách Contact".

Tiếp theo, bạn ràng buộc (bind) Fragment hiện tại vào ListView bằng cách gọi phương thức setOnItemClickListener() trong onActivityCreated(). Ví dụ:

```
    public void onActivityCreated(Bundle savedInstanceState) {
        ...
        // Thiết lập listener để lắng nghe sự kiện click vào một mục
        // cho Fragment hiện hành.
         mContactsList.setOnItemClickListener(this);
        ...
    }
```

Vì bạn đã chỉ định lớp Fragment hiện tại làm OnItemClickListener cho ListView, bạn cần hiện thực hóa phương thức bắt buộc của nó là onItemClick(), trong đó xử lý sự kiện click. Thao tác này được mô tả trong phần sau.

Định nghĩa phép chiếu

Định nghĩa một hằng số chứa các cột bạn muốn trả lại từ câu truy vấn. Mỗi dòng trong ListView hiển thị tên của Contact với dạng hiển thị chính. Trong Android 3.0 (API cấp 11) và cao hơn, tên cột này là Contacts.DISPLAY_NAME_PRIMARY; trong các phiên bản trước, tên cột này là Contacts.DISPLAY_NAME.

Cột Contacts._ID được dùng trong quá trình ràng buộc (bind) của SimpleCursorAdapter. Contacts._ID và LOOKUP_KEY được sử dụng cùng nhau để tạo content URI cho Contact người dùng chọn.

```
...
@SuppressLint("InlinedApi")
private static finalString[] PROJECTION =
      {
         Contacts._ID,
         Contacts.LOOKUP_KEY,
         Build.VERSION.SDK_INT
               >= Build.VERSION_CODES.HONEYCOMB ?
               Contacts.DISPLAY_NAME_PRIMARY :
               Contacts.DISPLAY_NAME

      };
```

Định nghĩa các hằng số làm chỉ mục của các cột trong Cursor

Để lấy dữ liệu từ một cột trong một Cursor, bạn cần chỉ mục (index) của cột này trong Cursor đó. Bạn có thể định nghĩa các hằng số làm chỉ mục của các cột trong Cursor, vì các chỉ mục trùng với thứ tự tên các cột trong phép chiếu. Ví dụ:

```
// Chỉ mục cho cột _ID
private static final int CONTACT_ID_INDEX = 0;
// Chỉ mục cho cột LOOKUP_KEY
private static final int LOOKUP_KEY_INDEX = 1;
```

Định nghĩa tiêu chí lựa chọn

Để chỉ định những dữ liệu bạn muốn, bạn cần tạo một biểu thức kết hợp các chuỗi và các biến để provider biết những cột cần tìm và các giá trị cần chọn.

Để tạo biểu thức này, bạn hãy định nghĩa một hằng số chứa tên các cột cần tìm kiếm. Dù biểu thức này có thể chứa cả các giá trị, bạn nên dùng một dấu "?" để đại diện cho các giá trị. Khi truy xuất, các ký tự giữ chỗ này được thay thế bằng các giá trị từ một mảng. Sử dụng "?" làm ký tự giữ chỗ đảm bảo câu truy vấn được sinh bởi quá trình ràng buộc dữ liệu thay vì dịch SQL. Cách làm này giúp loại trừ khả năng tấn công SQL injection[2]. Ví dụ:

[2] *Là một kỹ thuật cho phép những kẻ tấn công lợi dụng lỗ hổng của việc kiểm tra dữ liệu đầu vào trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu trả về để inject (tiêm vào) và chạy các câu lệnh SQL bất hợp pháp. Tham khảo: http://vi.wikipedia.org/wiki/SQL_injection.*

```
// Định nghĩa biểu thức lựa chọn
@SuppressLint("InlinedApi")
private static final String SELECTION =
      Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
      Contacts.DISPLAY_NAME_PRIMARY + " LIKE ?":
      Contacts.DISPLAY_NAME + " LIKE ?";
// Định nghĩa một biến lưu chuỗi tìm kiếm
private String mSearchString;
// Định nghĩa mảng chứa các giá trị để thay thế cho các dấu ?
private String[] mSelectionArgs = { mSearchString };
```

Định nghĩa phương thức onItemClick()

Trong phần trước, bạn đã thiết lập listener lắng nghe sự kiện click vào một dòng trên đối tượng ListView. Giờ bạn cần viết mã cho listener này bằng cách định nghĩa phương thức AdapterView.OnItemClickListener.onItemClick():

```
@Override
public void onItemClick(
   AdapterView<?> parent, View item, int position, long rowID) {
   // Lấy Cursor
   Cursor cursor = parent.getAdapter().getCursor();
   // Di chuyển tới Contact đã chọn
   cursor.moveToPosition(position);
   // Lấy giá trị _ID
   mContactId = getLong(CONTACT_ID_INDEX);
   // Lấy LOOKUP KEY đã chọn
   mContactKey = getString(CONTACT_KEY_INDEX);
   // Tạo content URI của Contact
   mContactUri = Contacts.getLookupUri(mContactId, mContactKey);
   /*
   * Bạn có thể dùng mContactUri làm content URI cho việc truy xuất
    * thông tin chi tiết của Contact.
    */
}
```

Khởi tạo đối tượng loader

Vì bạn đang sử dụng CursorLoader để truy xuất dữ liệu, bạn phải khởi tạo luồng chạy ngầm và các biến khác để kiểm soát thao tác truy xuất không đồng bộ. Bạn thực hiện việc khởi tạo trong onActivityCreated(), phương thức này được gọi ngay trước khi Giao diện Người dùng của Fragment xuất hiện. Ví dụ như sau:

```
public class ContactsFragment extends Fragment implements
      LoaderManager.LoaderCallbacks<Cursor> {
   ...
   // Được gọi ngay trước khi đối tượng Fragment hiển thị Giao diện
   // Người dùng của nó
   @Override
   public void onActivityCreated(Bundle savedInstanceState) {
      // Luôn gọi phương thức super đầu tiên
      super.onActivityCreated(savedInstanceState);
      ...
      // Khởi tạo đối tượng loader
      getLoaderManager().initLoader(0, null, this);
```

Hiện thực hóa phương thức onCreateLoader()

Phương thức onCreateLoader() được gọi bởi hệ thống loader ngay sau khi bạn gọi initLoader().

Trong onCreateLoader(), bạn thiết lập mẫu chuỗi tìm kiếm. Để biến một chuỗi th ành một mẫu, bạn chèn các ký tự "%" (dấu phần trăm) để đại diện cho một chuỗi từ không đến nhiều ký tự, hoặc các ký tự "_" (dấu gạch dưới) để đại diện cho một ký tự, hoặc bạn có thể kết hợp cả hai. Ví dụ, mẫu "%Jefferson%" sẽ khớp với "Thomas Jefferson" và "Jefferson Davis".

Phương thức sẽ trả về một đối tượng CursorLoader mới. Đối với tham số Content URI, bạn sử dụng Contacts.CONTENT_URI. URI này tham chiếu tới toàn bộ bảng. Ví dụ như sau:

```
   ...
   @Override
   public Loader<Cursor> onCreateLoader(int loaderId, Bundle args)
{
      /*
       * Tạo chuỗi mẫu và
       * lưu chuỗi mẫu vào mảng chứa các tham số lựa chọn
       */
      mSelectionArgs[0]="%"+ mSearchString +"%";
      // Bắt đầu truy vấn
      return new CursorLoader(
            getActivity(),
            Contacts.CONTENT_URI,
            PROJECTION,
            SELECTION,
            mSelectionArgs,
            null
      );
   }
```

Hiện thực hóa phương thức onLoadFinished() và onLoaderReset()

Hệ thống loader gọi onLoadFinished() khi Contacts Provider trả về các kết quả của câu truy vấn. Trong phương thức này, bạn có thể đưa đối tượng Cursor chứa kết quả nhận được vào SimpleCursorAdapter. Thao tác này tự động cập nhật đối tượng ListView để hiển thị các kết quả tìm kiếm:

```
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor)
{
        // Đưa Cursor chứa kết quả nhận được vào đối tượng adapter
        // của ListView
        mCursorAdapter.swapCursor(cursor);
    }
```

Phương thức onLoaderReset() được gọi khi hệ thống loader phát hiện Cursor chứa dữ liệu cũ. Bạn phải xóa tham chiếu của SimpleCursorAdapter tới Cursor hiện có. Nếu thiếu thao tác này, hệ thống loader sẽ không tái sinh đối tượng Cursor và gây ra rò rỉ bộ nhớ. Ví dụ:

```
    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Xóa tham chiếu tới Cursor hiện có
        mCursorAdapter.swapCursor(null);

    }
```

Giờ bạn đã có các mảnh ghép chính cho một ứng dụng tìm Contact có tên khớp với một chuỗi tìm kiếm và trả về kết quả trong một ListView. Người dùng có thể nhấn vào tên một Contact để lựa chọn. Sự kiện này gọi đến một listener, trong đó bạn có thể xử lý tiếp dữ liệu của Contact. Ví dụ, bạn có thể truy xuất thông tin chi tiết của Contact. Để tìm hiểu cách thực hiện điều đó, hãy xem phần tiếp theo "Truy xuất Thông tin Chi tiết của Contact".

Để hiểu rõ về cách tạo giao diện người dùng cho chức năng tìm kiếm, bạn có thể đọc tài liệu hướng dẫn sử dụng API, phần Creating a Search Interface (tạm dịch: Tạo giao diện tìm kiếm).

Các phần tiếp theo trong bài học này trình bày một số cách khác để tìm Contact trong Contacts Provider.

Tìm Contact theo một Kiểu dữ liệu Cụ thể

Kỹ thuật này cho phép bạn chỉ định kiểu dữ liệu mà bạn muốn tìm. Truy xuất theo tên là một trường hợp riêng của kiểu truy vấn này, nhưng bạn cũng có thể

làm điều đó với bất cứ kiểu dữ liệu nào liên quan đến thông tin chi tiết của một Contact. Ví dụ, bạn có thể truy xuất các Contact có một mã bưu chính cụ thể (postal code); trong trường hợp này, chuỗi tìm kiếm phải khớp với dữ liệu lưu trong hàng chứa mã bưu chính.

Với kiểu truy xuất này, trước tiên bạn cần viết các đoạn mã sau giống với ví dụ mẫu trong các phần trước:

- Yêu cầu Quyền đọc Provider
- Định nghĩa ListView và item layout
- Định nghĩa Fragment để hiển thị danh sách các Contact
- Định nghĩa các biến toàn cục
- Khởi tạo Fragment
- Thiết lập CursorAdapter cho ListView
- Thiết lập listener cho sự kiện chọn Contact
- Định nghĩa các hằng số làm chỉ mục của các cột trong Cursor

  Dù bạn đang truy xuất dữ liệu từ một bảng khác, thứ tự các cột trong phép chiếu giống nhau, vì vậy bạn có thể sử dụng cùng các chỉ mục cho Cursor.

- Định nghĩa phương thức onItemClick()
- Khởi tạo đối tượng loader
- Hiện thực hóa phương thức onLoadFinished() và onLoaderReset()

Các bước tiếp theo trình bày những đoạn mã bạn cần bổ sung để so khớp một chuỗi tìm kiếm với một kiểu dữ liệu cụ thể và hiển thị các kết quả.

Chọn kiểu dữ liệu và bảng

Để tìm kiếm một kiểu cụ thể về dữ liệu chi tiết của Contact, bạn phải biết giá trị kiểu MIME dành cho kiểu dữ liệu đó. Mỗi kiểu dữ liệu có một giá trị kiểu MIME duy nhất được định nghĩa bằng một hằng số CONTENT_ITEM_TYPE trong lớp con của ContactsContract.CommonDataKinds gắn với kiểu dữ liệu đó. Tên của các lớp con cho biết kiểu dữ liệu của chúng; ví dụ, lớp con dành cho dữ liệu email là ContactsContract.CommonDataKinds.Email, và kiểu MIME dành cho dữ liệu email được định nghĩa bởi hằng số Email.CONTENT_ITEM_TYPE.

Sử dụng bảng ContactsContract.Data cho câu lệnh tìm kiếm. Tất cả các hằng số bạn cần cho phép chiếu, mệnh đề chọn và thứ tự sắp xếp được định nghĩa trong bảng này hoặc kế thừa bởi bảng này.

Định nghĩa phép chiếu

Để định nghĩa một phép chiếu, bạn cần chọn một hoặc nhiều cột được định nghĩa trong ContactsContract.Data hoặc các lớp nó kế thừa. Contacts Provider thực hiện phép nối (join) ngầm định giữa ContactsContract.Data và các bảng khác trước khi trả về các hàng kết quả. Ví dụ:

```
@SuppressLint("InlinedApi")
private static final String[] PROJECTION =
    {
        /*
         * ID của một hàng chứa dữ liệu chi tiết. Để ListView
         * chạy được, cột này là bắt buộc.
         */
        Data._ID,
        // Tên hiển thị chính
        Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
              Data.DISPLAY_NAME_PRIMARY :
              Data.DISPLAY_NAME,
        // ID của Contact, để tạo một Content URI
        Data.CONTACT_ID
        // LOOKUP_KEY của Contact để tạo Content URI
        Data.LOOKUP_KEY
    };
```

Định nghĩa tiêu chí tìm kiếm

Để tìm kiếm một chuỗi trong một kiểu dữ liệu cụ thể, bạn tạo một mệnh đề chọn từ các thông tin sau:

- Tên của cột chứa chuỗi tìm kiếm của bạn. Tên này khác nhau tùy theo các kiểu dữ liệu, vì vậy bạn cần tìm lớp con của ContactsContract.CommonDataKinds tương ứng với kiểu dữ liệu cần dùng và sau đó chọn tên cột từ lớp con đó. Ví dụ, để tìm kiếm các địa chỉ email, bạn dùng cột Email.ADDRESS.
- Bản thân chuỗi tìm kiếm, được đại diện bằng ký tự "?" trong mệnh đề chọn.
- Tên của cột chứa giá trị kiểu MIME tùy chọn. Tên này luôn là Data.MIMETYPE.
- Giá trị kiểu MIME tùy chọn cho kiểu dữ liệu. Như mô tả ở trên, đây là hằng số CONTENT_ITEM_TYPE trong lớp con của ContactsContract.CommonDataKinds. Ví dụ, giá trị kiểu MIME dành cho dữ liệu email là Email.CONTENT_ITEM_TYPE. Bạn đặt giá trị này vào cặp nháy đơn bằng cách nối ký tự "'" (nháy đơn) vào đầu và cuối của hằng số; trái lại, provider sẽ dịch giá trị này như một tên biến thay vì một giá trị chuỗi. Bạn không cần sử dụng ký tự giữ chỗ cho giá trị này, vì nó là hằng số thay vì một giá trị do người dùng cung cấp.

Ví dụ:

```
/*
 * Tạo tiêu chí tìm kiếm từ chuỗi tìm kiếm và kiểu MIME email
 */
private static final String SELECTION =
      /*
       * Tìm địa chỉ email khớp với chuỗi tìm kiếm
       */
      Email.ADDRESS + " LIKE ? " + "AND " +
      /*
       * Tìm kiểu MIME khớp với giá trị của hằng số
       * Email.CONTENT_ITEM_TYPE. Lưu ý
       * cặp nháy đơn bao ngoài Email.CONTENT_ITEM_TYPE.
       */
      Data.MIMETYPE + " = '" + Email.CONTENT_ITEM_TYPE + "'";
```

Tiếp theo, bạn cần khai báo các biến để chứa biểu thức lựa chọn trên:

```
String mSearchString;
String[] mSelectionArgs = { "" };
```

Hiện thực hóa phương thức onCreateLoader()

Vì bạn đã chỉ định dữ liệu bạn cần và cách tìm kiếm, đã đến lúc khai báo câu truy vấn trong phần thân phương thức onCreateLoader(). Phương thức này trả về một đối tượng CursorLoader mới, trong đó sử dụng phép chiếu, biểu thức lựa chọn và mảng lựa chọn của bạn làm đối số. Đối với tham số Content URI, bạn sử dụng Data.CONTENT_URI. Ví dụ:

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
   // KHÔNG BẮT BUỘC: Tạo mẫu chuỗi tìm kiếm
   mSearchString = "%" + mSearchString + "%";
   // Đưa chuỗi tìm kiếm vào tiêu chí tìm kiếm
   mSelectionArgs[0] = mSearchString;
   // Bắt đầu câu truy vấn
   return new CursorLoader(
         getActivity(),
         Data.CONTENT_URI,
         PROJECTION,
         SELECTION,
         mSelectionArgs,
         null
   );
}
```

Đoạn mã mẫu này là nền tảng cho một ứng dụng tra cứu đơn giản dựa trên một kiểu dữ liệu cụ thể. Đây là kỹ thuật tốt nhất để sử dụng nếu ứng dụng của bạn chú trọng vào một kiểu dữ liệu nào đó, ví dụ như email và bạn muốn cho phép người dùng lấy được các Contact gắn với kiểu dữ liệu đó.

Tìm Contact theo Bất cứ Kiểu dữ liệu nào

Đây chính là việc truy xuất một Contact mà có bất kỳ thông tin nào thuộc bất cứ kiểu dữ liệu nào khớp với chuỗi tìm kiếm, bao gồm tên, địa chỉ email, địa chỉ nhà ở, số điện thoại… Kỹ thuật này này đem lại một danh sách kết quả tìm kiếm dài. Ví dụ, nếu chuỗi tìm kiếm là “Doe”, kết quả tìm kiếm cho bất cứ kiểu dữ liệu nào sẽ trả về Contact “John Doe” và cũng có thể trả về các Contact sống ở “Phố Doe”.

Với kiểu truy xuất này, trước tiên bạn cần viết các đoạn mã sau giống với ví dụ mẫu trong các phần trước:

- Yêu cầu Quyền đọc Provider
- Định nghĩa ListView và item layout
- Định nghĩa Fragment để hiển thị danh sách các Contact
- Định nghĩa các biến toàn cục
- Khởi tạo Fragment
- Thiết lập CursorAdapter cho ListView
- Thiết lập listener cho sự kiện chọn Contact
- Định nghĩa phép chiếu
- Định nghĩa các hằng số cho chỉ mục của các cột trong Cursor

  Đối với kiểu truy xuất này, bạn sẽ sử dụng cùng bảng dữ liệu bạn đã dùng trong phần “Tìm Contact theo Tên và Liệt kê các Kết quả”. Chỉ mục các cột cũng sẽ giống như vậy.

- Định nghĩa phương thức onItemClick()
- Khởi tạo đối tượng loader
- Hiện thực hóa phương thức onLoadFinished() và onLoaderReset()

Các bước tiếp theo trình bày những đoạn mã bạn cần bổ sung để so khớp một chuỗi tìm kiếm với bất cứ kiểu dữ liệu nào và hiển thị các kết quả.

Loại bỏ tiêu chí lựa chọn

Bạn không cần khai báo các hằng số lựa chọn SELECTION hoặc biến mSelectionArgs như phần trên. Các thông tin đó không được dùng cho kiểu truy xuất này.

Hiện thực hóa phương thức onCreateLoader()

Phương thức onCreateLoader() trả về một CursorLoader mới. Bạn không cần chuyển đổi chuỗi tìm kiếm thành một mẫu, vì Contacts Provider tự động làm điều đó. Bạn sử dụng Contacts.CONTENT_FILTER_URI làm URI gốc, và nối thêm chuỗi tìm kiếm bằng cách gọi Uri.withAppendedPath(). Khi sử dụng URI này, việc tìm kiếm mọi kiểu dữ liệu sẽ tự động kích hoạt. Ví dụ như sau:

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
    /*
     * Nối chuỗi tìm kiếm vào URI gốc.
     * Luôn mã hóa các chuỗi tìm kiếm để đảm bảo chúng có định
     * dạng đúng
     */
    Uri contentUri = Uri.withAppendedPath(
            Contacts.CONTENT_FILTER_URI,
            Uri.encode(mSearchString));
    // Bắt đầu câu truy vấn
    return new CursorLoader(
            getActivity(),
            contentUri,
            PROJECTION,
            null,
            null,
            null
    );
}
```

Đoạn mã này là nền tảng cho một ứng dụng tìm kiếm phổ rộng với Contacts Provider. Kỹ thuật này hữu ích cho các ứng dụng muốn thực thi tính năng tương tự như màn hình danh sách Contact của ứng dụng People.

Truy xuất Thông tin Chi tiết của Contact

Bài học này giải thích cách truy xuất dữ liệu chi tiết của một Contact như các địa chỉ email, các số điện thoại… Đó chính là những thông tin chi tiết người dùng muốn tìm khi họ truy xuất một Contact. Bạn có thể cho người

dùng thấy tất cả thông tin của một Contact, hoặc chỉ hiển thị thông tin của một kiểu dữ liệu cụ thể, ví dụ như các địa chỉ email.

Các bước trong bài học này giả định rằng bạn đã có một hàng ContactsContract.Contacts tương ứng với một Contact để người dùng chọn lựa. Phần "Truy xuất một Danh sách Contact" đã giải thích cách truy xuất danh sách các Contact.

Truy xuất Tất cả Thông tin Chi tiết của một Contact

Để lấy tất cả thông tin chi tiết của một Contact, bạn cần tìm kiếm trong bảng ContactsContract.Data bất cứ hàng nào chứa LOOKUP_KEY của Contact. Cột này có trong bảng ContactsContract.Data, vì Contacts Provider thực hiện phép nối ngầm định giữa bảng ContactsContract.Contacts và bảng ContactsContract.Data. Cột LOOKUP_KEY được giải thích chi tiết trong Phần "Truy xuất một Danh sách Contact".

> **Lưu ý:** Truy xuất tất cả thông tin chi tiết của một Contact làm giảm hiệu suất của thiết bị vì để làm được thì bạn cần truy xuất tất cả các cột trong bảng ContactsContract.Data. Bạn cần cân nhắc đến hiệu suất trước khi sử dụng kỹ thuật này.

Yêu cầu Quyền

Để lấy dữ liệu từ Contacts Provider, ứng dụng của bạn phải có quyền READ_CONTACTS. Để yêu cầu quyền này, bạn cần thêm phần tử sau vào bên trong phần tử <manifest> trong file kê khai:

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

Thiết lập một phép chiếu

Tùy thuộc vào kiểu dữ liệu có trong một hàng, phép chiếu có thể chỉ sử dụng một vài hoặc rất nhiều cột. Ngoài ra, dữ liệu trong các cột khác nhau phụ thuộc vào kiểu dữ liệu. Để đảm bảo bạn lấy được tất cả các cột có thể cho tất cả các kiểu dữ liệu có thể, bạn cần đưa tất cả tên các cột vào phép chiếu. Luôn nhớ truy xuất Data._ID nếu bạn ràng buộc Cursor được trả về vào ListView; nếu không làm như vậy, việc ràng buộc sẽ không thành công. Bạn cũng cần truy xuất Data.MIMETYPE để có thể xác định kiểu dữ liệu của mỗi hàng nhận được. Ví dụ:

```
    private static final String PROJECTION =
            {
                Data._ID,
                Data.MIMETYPE,
                Data.DATA1,
                Data.DATA2,
                Data.DATA3,
                Data.DATA4,
                Data.DATA5,
                Data.DATA6,
                Data.DATA7,
                Data.DATA8,
                Data.DATA9,
                Data.DATA10,
                Data.DATA11,
                Data.DATA12,
                Data.DATA13,
                Data.DATA14,
                Data.DATA15
            };
```

Phép chiếu này truy xuất tất cả các cột trong một hàng của bảng ContactsContract.Data, sử dụng tên các cột đã được định nghĩa trong lớp ContactsContract.Data.

Ngoài ra, bạn cũng có thể sử dụng bất cứ hằng số cột nào khác được định nghĩa trong lớp ContactsContract.Data hoặc được kế thừa từ lớp này. Lưu ý rằng các cột từ SYNC1 đến SYNC4 được dùng cho các adapter đồng bộ vì vậy dữ liệu của chúng không hữu ích.

Định nghĩa tiêu chí lựa chọn

Bạn cần định nghĩa một hằng số cho biểu thức lựa chọn, một mảng chứa các đối số lựa chọn và một biến chứa giá trị lựa chọn. Hãy sử dụng cột Contacts.LOOKUP_KEY để tìm Contact. Ví dụ:

```
// Định nghĩa biểu thức lựa chọn
private static final String SELECTION = Data.LOOKUP_KEY + " = ?";
// Định nghĩa mảng chứa các tiêu chí lựa chọn
private String[] mSelectionArgs = { "" };
/*
 * Định nghĩa một biến chứa giá trị lựa chọn. Ngay khi bạn
 * có Cursor từ bảng Contacts và bạn chọn được
 * hàng mong muốn, hãy gán giá trị LOOKUP_KEY của hàng đó vào
 * biến này.
 */
private String mLookupKey;
```

Sử dụng dấu "?" làm ký tự giữ chỗ trong biểu thức lựa chọn để đảm bảo câu truy vấn được sinh bởi quá trình ràng buộc dữ liệu thay vì dịch SQL. Cách làm này giúp loại trừ khả năng xảy ra tấn công SQL injection.

Định nghĩa thứ tự sắp xếp

Bạn có thể định nghĩa thứ tự sắp xếp trong Cursor được trả về. Để nhóm tất cả các hàng của một kiểu dữ liệu nào đó với nhau, bạn sắp xếp theo Data.MIMETYPE. Đối số truy vấn này nhóm tất cả các hàng email với nhau, tất cả hàng số điện thoại với nhau. Ví dụ:

```
/*
 * Định nghĩa một chuỗi chỉ định thứ tự sắp xếp theo kiểu MIME
 */
private static final String SORT_ORDER = Data.MIMETYPE;
```

> **Lưu ý:** Một số kiểu dữ liệu không sử dụng kiểu phụ (subtype), vì vậy bạn không thể sắp xếp theo kiểu phụ. Thay vào đó, bạn phải lặp qua đối tượng Cursor được trả về, quyết định kiểu dữ liệu của hàng hiện hành và lưu dữ liệu của các hàng có sử dụng kiểu phụ. Sau khi đọc xong đối tượng Cursor, bạn có thể sắp xếp mỗi kiểu dữ liệu theo kiểu phụ và hiển thị các kết quả.

Khởi tạo đối tượng loader

Bạn phải luôn truy xuất dữ liệu từ Contacts Provider (và tất cả các content provider khác) trong một luồng chạy ngầm. Bạn sử dụng hệ thống Loader được định nghĩa trong lớp LoaderManager và giao diện LoaderManager.LoaderCallbacks để thực hiện việc truy xuất dữ liệu chạy ngầm.

Khi đã sẵn sàng truy xuất dữ liệu, bạn khởi tạo hệ thống loader bằng cách gọi initLoader(). Bạn truyền vào một định danh kiểu số nguyên vào phương thức này; số này sẽ được truyền vào các phương thức của LoaderManager.LoaderCallbacks. Số định danh này giúp bạn phân biệt các Loader bạn sử dụng trong một ứng dụng.

Đoạn mã sau cho thấy cách khởi tạo hệ thống loader.

```
public class DetailsFragment extends Fragment implements
       LoaderManager.LoaderCallbacks<Cursor> {
    ...
    // Định nghĩa một hằng số xác định đối tượng loader
    DETAILS_QUERY_ID = 0;
    ...
    /*
     * Được gọi khi Activity cha được khởi tạo
     * và UI của Fragment sẵn sàng. Bạn đặt các bước khởi tạo
     * cuối cùng ở đây.
     */
    @Override
    onActivityCreated(Bundle savedInstanceState) {
        ...
        // Khởi tạo hệ thống loader
        getLoaderManager().initLoader(DETAILS_QUERY_ID, null, this);
```

Hiện thực hóa phương thức onCreateLoader()

Phương thức onCreateLoader() được gọi bởi hệ thống loader ngay sau khi bạn gọi initLoader(). Phương thức này trả về một đối tượng CursorLoader. Vì bạn đang tìm kiếm trong bảng ContactsContract.Data, bạn cần sử dụng hằng số Data.CONTENT_URI làm Content URI. Ví dụ:

```
    @Override
    public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
      // Chọn hành động thích hợp
      switch (loaderId) {
      case DETAILS_QUERY_ID:
      // Gán tham số lựa chọn
      mSelectionArgs[0] = mLookupKey;
      // Bắt đầu truy vấn
      CursorLoader mLoader =
         new CursorLoader(
            getActivity(),
            Data.CONTENT_URI,
            PROJECTION,
            SELECTION,
            mSelectionArgs,
            SORT_ORDER
          );
      ...
    }
```

Hiện thực hóa phương thức onLoadFinished() và onLoaderReset()

Hệ thống loader gọi onLoadFinished() khi Contacts Provider trả về các kết quả của câu truy vấn. Ví dụ:

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    switch (loader.getId()) {
        case DETAILS_QUERY_ID:
            /*
             * Xử lý đối tượng Cursor chứa kết quả ở đây.
             */
        }
        break;
        ...
    }
}
```

Phương thức onLoaderReset() được gọi khi hệ thống loader phát hiện dữ liệu bên dưới của đối tượng Cursor đã thay đổi. Tại thời điểm này, bạn xóa các tham chiếu hiện có tới đối tượng Cursor bằng cách gán chúng bằng null. Nếu không làm vậy, hệ thống loader sẽ không hủy đối tượng Cursor cũ, và ứng dụng của bạn sẽ bị rò rỉ bộ nhớ. Ví dụ:

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    switch (loader.getId()) {
        case DETAILS_QUERY_ID:
            /*
            * Nếu bạn đang có các tham chiếu tới đối tượng Cursor,
             * hãy xóa chúng ở đây.
             */
            }
            break;
}
```

Truy xuất Thông tin chi tiết cụ thể của một Contact

Việc truy xuất một kiểu dữ liệu cụ thể của một Contact, ví dụ như tất cả các địa chỉ email, tuân theo cách thức giống như việc truy xuất tất cả thông tin chi tiết. Những thay đổi duy nhất bạn cần thực hiện đối với đoạn mã trong phần "Truy xuất Tất cả Thông tin Chi tiết của Contact" là:

Định nghĩa phép chiếu

Thay đổi phép chiếu để truy xuất các cột ứng với kiểu dữ liệu cụ thể. Đồng thời thay đổi phép chiếu để sử dụng các hằng số tên cột được định nghĩa trong lớp con của ContactsContract.CommonDataKinds tương ứng với kiểu dữ liệu.

Mệnh đề chọn

Thay đổi chuỗi lựa chọn để tìm kiếm giá trị MIMETYPE riêng của kiểu dữ liệu.

Thứ tự sắp xếp

Vì bạn chỉ chọn một kiểu dữ liệu duy nhất, đừng nhóm Cursor được trả về theo Data.MIMETYPE.

Các thay đổi này được mô tả trong những phần dưới đây.

Định nghĩa phép chiếu

Bạn khai báo những cột muốn truy xuất bằng cách dùng các hằng số tên cột trong lớp con của ContactsContract.CommonDataKinds tương ứng với kiểu dữ liệu. Nếu bạn dự định ràng buộc Cursor vào ListView, hãy nhớ truy xuất cột _ID. Ví dụ, để truy xuất dữ liệu email, bạn định nghĩa phép chiếu sau:

```
    private static final String[] PROJECTION =
            {
                Email._ID,
                Email.ADDRESS,
                Email.TYPE,
                Email.LABEL
            };
```

> Lưu ý rằng phép chiếu này sử dụng các tên cột được định nghĩa trong lớp ContactsContract.CommonDataKinds.Email thay vì các tên cột định nghĩa trong lớp ContactsContract.Data. Việc sử dụng các tên cột dành riêng cho email khiến mã dễ đọc hơn.

Trong phép chiếu, bạn cũng có thể sử dụng bất cứ cột nào khác được định nghĩa trong lớp con của ContactsContract.CommonDataKinds.

Định nghĩa tiêu chí lựa chọn

Bạn chỉ định biểu thức lựa chọn để truy xuất các dòng có LOOKUP_KEY cụ thể và có kiểu Data.MIMETYPE của thông tin Contact chi tiết theo ý bạn. Bạn đưa giá trị MIMETYPE vào cặp nháy đơn bằng cách nối ký tự "'" (nháy đơn) vào đầu và cuối của hằng số; nếu không bao hàm cặp nháy đơn này, provider

sẽ dịch giá trị này như một tên biến thay vì một giá trị chuỗi. Bạn không cần sử dụng ký tự giữ chỗ cho giá trị này vì bạn đang dùng một hằng số thay vì một giá trị do người dùng cung cấp. Ví dụ:

```
/*
 * Định nghĩa biểu thức lựa chọn. Tìm một lookup key
 * và MIME kiểu Email
 */
private static final String SELECTION =
      Data.LOOKUP_KEY + " = ?" +
      " AND " +
      Data.MIMETYPE + " = " +
      "'" + Email.CONTENT_ITEM_TYPE + "'";
// Định nghĩa mảng chứa các tiêu chí tìm kiếm
private String[] mSelectionArgs = { "" };
```

Định nghĩa thứ tự sắp xếp

Định nghĩa thứ tự sắp xếp cho Cursor chứa kết quả trả về. Vì bạn đang truy xuất một kiểu dữ liệu cụ thể, hãy bỏ qua thao tác sắp xếp theo MIMETYPE. Thay vào đó, nếu kiểu dữ liệu bạn đang tìm kiếm có kiểu phụ, hãy sử dụng kiểu phụ để sắp xếp. Ví dụ, với dữ liệu email bạn có thể sắp xếp theo Email.TYPE:

```
private static final String SORT_ORDER = Email.TYPE + " ASC ";
```

Chỉnh sửa Contact bằng cách dùng Intent

Bài học này trình bày cách sử dụng Intent để tạo một Contact mới hoặc chỉnh sửa dữ liệu của một Contact. Thay vì truy cập trực tiếp Contacts Provider, bạn dùng một Intent để mở ứng dụng Contacts, trong đó chạy Activity thích hợp. Với các thao tác chỉnh sửa mô tả trong bài học này, nếu bạn gửi đi dữ liệu mở rộng trong Intent, chúng sẽ được truyền vào UI của Activity đã khởi động.

Sử dụng Intent để tạo mới hoặc cập nhật một Contact là cách ưu tiên khi làm việc với Contacts Provider vì những lý do sau:

- Bạn tiết kiệm được thời gian và công sức dành cho việc tự phát triển UI và viết mã.
- Bạn tránh được lỗi gây ra bởi các thao tác chỉnh sửa không tuân theo quy tắc của Contacts Provider.

- Bạn giảm được số quyền cần yêu cầu. Ứng dụng không cần quyền ghi vào Contacts Provider vì đã ủy thác việc chỉnh sửa cho ứng dụng Contacts vốn có quyền đó.

Tạo một Contact mới bằng cách dùng Intent

Thông thường, bạn muốn cho phép người dùng tạo một Contact mới khi ứng dụng của bạn nhận dữ liệu mới. Ví dụ, một ứng dụng đánh giá các nhà hàng có thể cho phép người dùng thêm một nhà hàng vào danh sách Contact khi họ đang đánh giá nó. Để làm điều này với Intent, bạn tạo một Intent với tất cả dữ liệu bạn có, sau đó gửi Intent đó tới ứng dụng Contacts.

Việc tạo mới một Contact bằng ứng dụng Contacts sẽ tạo mới một Contact *thô* (raw contact) trong bảng ContactsContract.RawContacts của Contacts Provider. Nếu cần thiết, ứng dụng Contacts sẽ hỏi người dùng kiểu tài khoản và tên tài khoản để sử dụng khi tạo Contact thô. Ứng dụng Contacts cũng thông báo cho người dùng nếu Contact thô đã tồn tại. Sau đó, người dùng có thể lựa chọn hủy thao tác tạo mới và sẽ không Contact mới nào được tạo ra. Để tìm hiểu hơn về Contact thô, bạn có thể đọc tài liệu hướng dẫn sử dụng API Contacts Provider.

Tạo một Intent

Trước tiên, bạn tạo một đối tượng Intent mới với hành động Intents.Insert.ACTION. Bạn đặt kiểu MIME bằng RawContacts.CONTENT_TYPE. Ví dụ:

```
...
// Tạo một Intent để tạo Contact
Intent intent = new Intent(Intents.Insert.ACTION);
// Đặt kiểu MIME khớp với Contacts Provider
intent.setType(ContactsContract.RawContacts.CONTENT_TYPE);
```

Nếu bạn đã có thông tin chi tiết của Contact, ví dụ như số điện thoại hoặc địa chỉ email, bạn có thể đưa chúng vào Intent dưới dạng dữ liệu mở rộng. Đối với giá trị làm khóa, bạn sử dụng hằng số thích hợp từ Intents.Insert. Ứng dụng Contacts hiển thị dữ liệu đó trong màn hình tạo mới của nó, trong đó cho phép người dùng chỉnh sửa và bổ sung hơn nữa.

```
/* Giả sử các trường EditText trong UI của bạn chứa một địa chỉ email
 * và một số điện thoại.
 *
 */
```

```
private EditText mEmailAddress = (EditText) findViewById(R.id.email);
private EditText mPhoneNumber = (EditText) findViewById(R.id.phone);
...
/*
 * Chèn dữ liệu mới vào Intent. Dữ liệu này được truyền vào
 * màn hình Tạo mới Contact của ứng dụng Contacts
 */
// Chèn một địa chỉ email
intent.putExtra(Intents.Insert.EMAIL, mEmailAddress.getText())
/*
 * Trong ví dụ này, bạn đặt kiểu email là email công việc.
 * Bạn có thể dùng các kiểu email khác nếu cần.
 */
     .putExtra(Intents.Insert.EMAIL_TYPE, CommonDataKinds.Email.TYPE_WORK)
// Chèn một số điện thoại
     .putExtra(Intents.Insert.PHONE, mPhoneNumber.getText())
/*
 * Trong ví dụ này, bạn đặt kiểu số điện thoại là số điện thoại công việc.
 * Bạn có thể dùng các kiểu số điện thoại khác nếu cần.
 */
     .putExtra(Intents.Insert.PHONE_TYPE, Phone.TYPE_WORK);
```

Khi đã tạo xong Intent, bạn gửi đi bằng cách gọi startActivity().

```
    /* Gửi Intent
     */
    startActivity(intent);
```

Lời gọi này mở ra một màn hình trong ứng dụng Contacts cho phép người dùng nhập một Contact mới. Kiểu tài khoản và tên tài khoản cho Contact này được liệt kê ở đầu màn hình. Sau khi người dùng nhập dữ liệu và nhấn *Done*, danh sách Contact của ứng dụng Contacts sẽ xuất hiện. Người dùng trở lại ứng dụng của bạn bằng cách nhấn *Back*.

Chỉnh sửa Contact Hiện có bằng Intent

Chỉnh sửa một Contact hiện có bằng Intent là thao tác hữu dụng khi người dùng đã chọn sẵn một Contact họ quan tâm. Ví dụ, một ứng dụng tìm kiếm các Contact có địa chỉ bưu điện nhưng không có mã bưu điện có thể cho phép người dùng tra cứu mã bưu điện và sau đó thêm thông tin đó vào Contact.

Để chỉnh sửa một Contact hiện có bằng Intent, bạn sử dụng quy trình tương tự thao tác tạo mới Contact. Bạn tạo một Intent như mô tả trong

phần "Tạo một Contact mới bằng cách dùng Intent", nhưng truyền thêm Contacts.CONTENT_LOOKUP_URI của Contact và kiểu MIME Contacts.CONTENT_ITEM_TYPE vào Intent đó. Nếu bạn muốn chỉnh sửa Contact với các thông tin bạn đã có, bạn có thể đưa chúng vào phần dữ liệu mở rộng của Intent. Lưu ý rằng một số cột không thể chỉnh sửa được khi sử dụng Intent; các cột này được liệt kê trong tài liệu tham khảo API của lớp ContactsContract.Contacts, phần tóm tắt (summary), mục "Update".

Cuối cùng, bạn gửi Intent đó đi. Đáp lại, ứng dụng Contacts hiển thị màn hình chỉnh sửa Contact. Khi người dùng hoàn thành việc chỉnh sửa và lưu thông tin, ứng dụng Contacts sẽ hiển thị một danh sách Contact. Khi người dùng nhấn *Back*, ứng dụng của bạn sẽ được hiển thị.

Lookup Key của Contact

Giá trị LOOKUP_KEY của một Contact là mã định danh dùng để truy xuất một Contact. Giá trị này không đổi kể cả khi Contacts Provider thay đổi ID hàng dữ liệu của Contact để xử lý các thao tác nội bộ.

Tạo Intent

Để chỉnh sửa một Contact, bạn gọi Intent(action) để tạo một Intent với action ACTION_EDIT. Sau đó, bạn gọi setDataAndType() để gán giá trị dữ liệu cho Intent bằng Contacts.CONTENT_LOOKUP_URI của Contact và kiểu MIME bằng Contacts.CONTENT_ITEM_TYPE; vì lời gọi setType() ghi đè giá trị dữ liệu hiện tại của Intent, bạn phải gán dữ liệu và kiểu MIME cùng lúc.

Để lấy Contacts.CONTENT_LOOKUP_URI của Contact, bạn gọi Contacts.getLookupUri(id, lookupkey) với Contacts._ID của Contact và các giá trị Contacts.LOOKUP_KEY làm đối số.

Đoạn mã sau cho thấy cách tạo một Intent như vậy:

```
// Cursor chứa hàng Contact
public Cursor mCursor;
// Chỉ mục của cột lookup key trong Cursor
public int mLookupKeyIndex;
// Chỉ mục của giá trị _ID của Contact
public int mIdIndex;
// lookup key từ Cursor
public String mCurrentLookupKey;
// Giá trị _ID từ Cursor
public long mCurrentId;
```

```
// Content URI trỏ tới Contact
Uri mSelectedContactUri;
...
/*
 * Khi người dùng đã chọn một Contact để chỉnh sửa,
 * đoạn mã sau lấy lookup key của Contact và các giá trị _ID từ
 * Cursor và tạo URI cần thiết.
 */
// Lấy chỉ mục của cột lookup key
mLookupKeyIndex = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Lấy giá trị của lookup key
mCurrentLookupKey = mCursor.getString(mLookupKeyIndex);
// Lấy chỉ mục cột_ID
mIdIndex = mCursor.getColumnIndex(Contacts._ID);
mCurrentId = mCursor.getLong(mIdIndex);
mSelectedContactUri =
      Contacts.getLookupUri(mCurrentId, mCurrentLookupKey);
...
// Tạo một Intent mới để chỉnh sửa Contact
Intent editIntent = new Intent(Intent.ACTION_EDIT);
/*
 * Gán URI của Contact cần chỉnh sửa và kiểu dữ liệu phù hợp
 * cho Intent
 */
editIntent.setDataAndType(mSelectedContactUri,
                          Contacts.CONTENT_ITEM_TYPE);
```

Thêm cờ điều hướng

Trong Android 4.0 (API cấp 14) và cao hơn, ứng dụng Contacts có lỗi trong việc điều hướng. Khi ứng dụng của bạn gửi một Intent chỉnh sửa tới ứng dụng Contacts, người dùng chỉnh sửa và lưu Contact, khi họ nhấn *Back* họ sẽ thấy màn hình danh sách Contact. Để quay trở lại ứng dụng của bạn, họ phải nhấn *Recents* và chọn ứng dụng của bạn.

Để xử lý vấn đề này, trong Android 4.0.3 (API cấp 15) và cao hơn, bạn có thể thêm dữ liệu mở rộng với khóa finishActivityOnSaveCompleted vào Intent và đặt giá trị bằng true. Các bản Android từ trước Android 4.0 chấp nhận khóa này, nhưng không có tác động gì. Để thiết lập dữ liệu mở rộng, bạn dùng câu lệnh sau:

```
// Thiết lập dữ liệu mở rộng đặc biệt cho việc điều hướng
editIntent.putExtra("finishActivityOnSaveCompleted", true);
```

Thêm các dữ liệu mở rộng khác

Để thêm dữ liệu mở rộng vào Intent, bạn gọi putExtra() khi cần. Bạn có thể thêm dữ liệu mở rộng cho các trường phổ biến của Contact bằng cách sử dụng các giá trị khóa định nghĩa trong Intents.Insert. Hãy luôn nhớ một số cột trong bảng ContactsContract.Contacts không thể chỉnh sửa được. Các cột này được liệt kê trong tài liệu tham khảo API của lớp ContactsContract.Contacts, phần tóm tắt (summary), mục "Update".

Gửi Intent

Cuối cùng, bạn gửi đi Intent vừa tạo. Ví dụ:

```
// Gửi Intent
startActivity(editIntent);
```

Để Người dùng Lựa chọn Tạo mới hay Chỉnh sửa bằng Intent

Bạn có thể cho phép người dùng lựa chọn tạo mới một Contact hoặc chỉnh sửa một Contact hiện có bằng cách gửi đi một Intent với action ACTION_INSERT_OR_EDIT. Ví dụ, một ứng dụng khách email có thể cho phép người dùng sử dụng một địa chỉ email trong thư nhận được để tạo một Contact mới, hoặc dùng nó để thêm địa chỉ email cho một Contact hiện có. Bạn gán kiểu MIME cho Intent này bằng Contacts.CONTENT_ITEM_TYPE, nhưng không gán URI dữ liệu.

Khi bạn gửi đi Intent này, ứng dụng Contacts hiển thị danh sách Contact. Người dùng có thể tạo mới Contact hoặc chọn một Contact hiện có để chỉnh sửa. Bất cứ trường dữ liệu mở rộng nào bạn đã đưa vào Intent sẽ được điền vào màn hình xuất hiện sau đó. Bạn có thể sử dụng bất cứ giá trị khóa nào được định nghĩa trong Intents.Insert. Đoạn mã sau minh họa cách tạo và gửi một Intent như vậy:

```
// Tạo một Intent để tạo hoặc chỉnh sửa một Contact
Intent intentInsertEdit = new Intent(Intent.ACTION_INSERT_OR_EDIT);
// Thiết lập kiểu MIME
intentInsertEdit.setType(Contacts.CONTENT_ITEM_TYPE);
// Thêm mã ở đây để chèn các dữ liệu mở rộng nếu cần
...
// Gửi Intent với ID của yêu cầu
startActivity(intentInsertEdit);
```

Hiển thị Quick Contact Badge

Bài học này trình bày cách đưa QuickContactBadge (Biểu tượng lối tắt cho Contact) vào Giao diện Người dùng và cách ràng buộc dữ liệu cho nó. QuickContactBadge là một widget xuất hiện ban đầu dưới dạng một hình ảnh thu nhỏ (thumbnail). Dù bạn có thể sử dụng bất cứ ảnh Bitmap nào làm hình thu nhỏ, thường bạn sẽ dùng một ảnh Bitmap được giải mã từ hình thu nhỏ của Contact.

Hình thu nhỏ này đóng vai trò như là một đối tượng điều khiển, khi người dùng nhấn vào hình, QuickContactBadge mở rộng thành một hộp thoại chứa những thành phần sau:

Một hình ảnh lớn

Hình ảnh lớn gắn với Contact hoặc một hình đại diện giữ chỗ nếu Contact không có ảnh.

Các biểu tượng ứng dụng

Mỗi biểu tượng ứng dụng dành cho một dữ liệu mà có thể được xử lý bởi một ứng dụng đã cài đặt sẵn. Ví dụ, nếu thông tin chi tiết của Contact có một hoặc nhiều địa chỉ email, một biểu tượng email sẽ xuất hiện. Khi người dùng nhấn vào biểu tượng đó, tất cả các địa chỉ email của Contact hiện ra. Khi người dùng nhấn vào một trong các địa chỉ đó, ứng dụng email sẽ hiển thị để người dùng soạn thư tới địa chỉ email đã chọn.

QuickContactBadge cung cấp cách truy cập trực tiếp tới thông tin chi tiết của một Contact, đồng thời cung cấp một cách nhanh chóng để liên lạc với Contact đó. Người dùng không phải tra cứu Contact, tìm kiếm và sao chép thông tin, sau đó dán thông tin vào ứng dụng phù hợp. Thay vào đó, họ có thể nhấn vào QuickContactBadge, chọn cách thức liên lạc họ muốn dùng, và gửi trực tiếp thông tin dùng cho cách thức đó tới ứng dụng phù hợp.

Tạo View kiểu QuickContactBadge

Để tạo một QuickContactBadge, bạn chèn phần tử <QuickContactBadge> vào layout của bạn. Ví dụ:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
...
   <QuickContactBadge
            android:id=@+id/quickbadge
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:scaleType="centerCrop"/>
   ...
</RelativeLayout>
```

Truy xuất dữ liệu từ provider

Để hiển thị một Contact trong QuickContactBadge, bạn cần content URI của Contact và một ảnh Bitmap làm hình thu nhỏ. Bạn có thể tạo cả content URI và ảnh Bitmap từ các cột lấy từ Contacts Provider. Bạn chỉ định các cột này trong phép chiếu khi nạp dữ liệu vào Cursor.

Đối với Android 3.0 (API cấp 11) và cao hơn, bạn cần đưa các cột sau vào phép chiếu:

- Contacts._ID
- Contacts.LOOKUP_KEY
- Contacts.PHOTO_THUMBNAIL_URI

Đối với Android 2.3.3 (API cấp 10) và thấp hơn, bạn sử dụng các cột sau:

- Contacts._ID
- Contacts.LOOKUP_KEY

Phần còn lại của bài học này giả định rằng bạn đã nạp một Cursor chứa các cột này cũng như các cột khác mà bạn có thể chọn thêm. Để tìm hiểu cách truy xuất các cột này và đưa kết quả vào một Cursor, bạn có thể đọc phần “Truy xuất một Danh sách Contact”.

Thiết lập Contact URI và Hình thu nhỏ

Khi đã có các cột cần thiết, bạn có thể ràng buộc dữ liệu vào QuickContactBadge.

Thiết lập Contact URI

Để thiết lập content URI cho Contact, bạn gọi getLookupUri(id,lookupKey) để lấy một CONTENT_LOOKUP_URI, sau đó gọi assignContactUri() để thiết lập Contact. Ví dụ:

```
// Cursor chứa các hàng Contact
Cursor mCursor;
// Chỉ mục của cột _ID trong Cursor
int mIdColumn;
// Chỉ mục của cột LOOKUP_KEY trong Cursor
int mLookupKeyColumn;
// Content URI của Contact mong muốn
Uri mContactUri;
// Tham chiếu tới View QuickContactBadge
QuickContactBadge mBadge;
...
mBadge = (QuickContactBadge) findViewById(R.id.quickbadge);
/*
 * Chèn mã ở đây để di chuyển đến hàng mong muốn trong Cursor
 */
// Lấy chỉ mục của cột _ID
mIdColumn = mCursor.getColumnIndex(Contacts._ID);
// Lấy chỉ mục của cột LOOKUP_KEY
mLookupKeyColumn = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Lấy Content URI của Contact
mContactUri =
      Contacts.getLookupUri(
         mCursor.getLong(mIdColumn),
         mCursor.getString(mLookupKeyColumn)
      );
mBadge.assignContactUri(mContactUri);
```

Khi người dùng nhấn vào biểu tượng QuickContactBadge, thông tin chi tiết của Contact tự động hiện lên trong hộp thoại.

Thiết lập hình thu nhỏ

Việc thiết lập Contact URI cho QuickContactBadge không tự động nạp hình thu nhỏ của Contact. Để nạp hình ảnh đó, bạn lấy URI của bức ảnh từ một hàng trong Cursor của Contact, sau đó sử dụng URI này để mở file chứa bức ảnh đã nén, và đọc file đó vào một Bitmap.

> **Lưu ý:** Cột PHOTO_THUMBNAIL_URI không có ở các phiên bản trước 3.0. Đối với các phiên bản đó, bạn phải lấy URI của bức ảnh từ bảng Contacts.Photo.

Trước tiên, bạn thiết lập các biến để truy cập Cursor chứa cột Contacts._ID và cột Contacts.LOOKUP_KEY, như đã mô tả bên trên:

```
// Cột chứa ID hình thu nhỏ cần tìm
int mThumbnailColumn;
/*
 * URI của hình thu nhỏ, dưới dạng một chuỗi.
 * Contacts Provider lưu các URI dưới dạng các giá trị chuỗi.
 */
String mThumbnailUri;
...
/*
 * Lấy chỉ mục của cột hình thu nhỏ nếu
 * phiên bản Android >= Honeycomb
 */
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
   mThumbnailColumn =
         mCursor.getColumnIndex(Contacts.PHOTO_THUMBNAIL_URI);
// Ngược lại, thiết lập cột hình thu nhỏ bằng cột _ID
} else {
   mThumbnailColumn = mIdColumn;
}
/*
 * Giả sử vị trí Cursor hiện tại là Contact bạn cần,
 * lấy ID hình thu nhỏ
 */
mThumbnailUri = mCursor.getString(mThumbnailColumn);
...
```

Sau đó, bạn định nghĩa một phương thức nhận vào dữ liệu liên quan đến ảnh của Contact và các kích thước của đối tượng View đích, và trả về một hình thu nhỏ có kích thước phù hợp dạng Bitmap. Đầu tiên, bạn tạo một URI trỏ tới hình thu nhỏ:

```
/**
 * Nạp một hình của Contact và trả về bức ảnh dưới dạng Bitmap,
 * @param photoData là ID của hình ảnh. Trước Honeycomb, đây là
 * giá trị _ID của Contact.
 * Kể từ Honeycomb, đây là giá trị PHOTO_THUMBNAIL_URI.
 * @return Trả về ảnh Bitmap thu nhỏ
 * Trả về null nếu không tìm thấy hình thu nhỏ.
 */
private Bitmap loadContactPhotoThumbnail (String photoData) {
   // Tạo một AssetFileDescriptor (Bộ đặc tả tài nguyên file)
   // cho file hình thu nhỏ
   AssetFileDescriptor afd = null;
   // khối try-catch cho trường hợp không tìm thấy file
   try {
      // Tạo biến chứa URI.
      Uri thumbUri;
      // Nếu Android 3.0 hoặc sau đó
      if (Build.VERSION.SDK_INT >=
      Build.VERSION_CODES.HONEYCOMB) {
         // Thiết lập URI từ PHOTO_THUMBNAIL_URI nhận vào
         thumbUri = Uri.parse(photoData);
      } else {
      // Trước Android 3.0, tạo URI ảnh bằng _ID
         /*
          * Tạo một Contact URI từ content URI của Contacts
          * và dữ liệu ảnh nhận được (_ID)
          */
         final Uri contactUri = Uri.withAppendedPath(
               Contacts.CONTENT_URI, photoData);
         /*
          * Tạo URI ảnh bằng cách ghép thêm Content URI của
          * Contacts.Photo.
          */
         thumbUri =
               Uri.withAppendedPath(
                     contactUri, Photo.CONTENT_DIRECTORY);
      }

   /*
    * Truy xuất một đối tượng AssetFileDescriptor dành cho URI
    * của hình thu nhỏ bằng cách gọi ContentResolver.
    * openAssetFileDescriptor
    */
   afd = getActivity().getContentResolver().
         openAssetFileDescriptor(thumbUri,"r");
```

```
        /*
         * Lấy FileDescriptor từ AssetFileDescriptor.
         * Đối tượng này có thể được dùng giữa các tiến trình.
         */
        FileDescriptor fileDescriptor = afd.getFileDescriptor();
        // Giải mã file ảnh và trả về kết quả dạng Bitmap
        // Nếu fileDescriptor hợp lệ
        if (fileDescriptor != null) {
            // Giải mã ảnh bitmap
            return BitmapFactory.decodeFileDescriptor(
                    fileDescriptor, null, null);
            }
        // Nếu không tìm thấy file
        } catch (FileNotFoundException e) {
            /*
             * Xử lý các lỗi không tìm thấy file
             */
        }
        // Trong mọi trường hợp, bạn cần đóng đối tượng
        // AssetFileDescriptor
        } finally {
            if (afd != null) {
                try {
                    afd.close();
                } catch (IOException e) {}
            }
        }
        return null;
    }
```

Bạn gọi loadContactPhotoThumbnail() trong đoạn mã để lấy ảnh thu nhỏ dạng Bitmap và sử dụng kết quả để thiết lập hình thu nhỏ trong QuickContactBadge:

```
    ...
    /*
     * Giải mã file ảnh thu nhỏ thành một Bitmap.
     */
    Bitmap mThumbnail =
            loadContactPhotoThumbnail(mThumbnailUri);
    /*
     * Thiết lập ảnh đó vào QuickContactBadge
     * QuickContactBadge kế thừa từ ImageView, vì vậy bạn có thể gọi
     */
    mBadge.setImageBitmap(mThumbnail);
```

Thêm QuickContactBadge vào ListView

QuickContactBadge là phần bổ sung hữu ích cho các ListView dùng để hiển thị một danh sách Contact. Bạn có thể sử dụng QuickContactBadge để hiển thị hình thu nhỏ cho mỗi Contact, khi người dùng nhấn vào hình thu nhỏ, hộp thoại QuickContactBadge sẽ xuất hiện.

Thêm phần tử QuickContactBadge

Trước tiên, bạn thêm một phần tử View kiểu QuickContactBadge vào item layout. Ví dụ, nếu bạn muốn hiển thị một QuickContactBadge và tên cho mỗi Contact trong danh sách kết quả truy vấn, bạn đặt đoạn XML sau vào file layout:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
   <QuickContactBadge
      android:id="@+id/quickcontact"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:scaleType="centerCrop" />
   <TextView android:id="@+id/displayname"
         android:layout_width=”match_parent”
          android:layout_height=”wrap_content”
          android:layout_toRightOf=”@+id/quickcontact”
          android:gravity=”center_vertical”
          android:layout_alignParentRight=”true”
          android:layout_alignParentTop=”true” />
</RelativeLayout>
```

Trong các phần sau, file này được tham chiếu bằng tên contact_item_layout.xml.

Thiết lập CursorAdapter tùy chỉnh

Để ràng buộc một CursorAdapter vào một ListView chứa QuickContactBadge, bạn cần định nghĩa một adapter tùy chỉnh kế thừa CursorAdapter. Cách làm này cho phép bạn xử lý dữ liệu trong Cursor trước khi ràng buộc dữ liệu vào QuickContactBadge. Cách làm này cũng cho phép bạn ràng buộc nhiều cột của Cursor vào QuickContactBadge. Cả hai thao tác này đều không thể thực hiện được với CursorAdapter thông thường.

Lớp con của CursorAdapter mà bạn định nghĩa phải ghi đè các phương thức sau:

CursorAdapter.newView()

Khai triển một đối tượng View mới chứa item layout. Trong phương thức ghi đè, bạn lưu tham chiếu tới các đối tượng View con của layout, bao gồm đối tượng QuickContactBadge. Với cách làm này, bạn tránh được việc phải lấy về tham chiếu tới các đối tượng View con mỗi lần bạn khai triển một layout mới.

Bạn phải ghi đè phương thức này để có thể lấy tham chiếu tới từng đối tượng View con. Kỹ thuật này cho phép bạn kiểm soát việc ràng buộc dữ liệu của chúng trong CursorAdapter.bindView().

CursorAdapter.bindView()

Chuyển dữ liệu từ hàng hiện thời trong Cursor tới các đối tượng View con trong item layout. Bạn phải ghi đè phương thức này để có thể ràng buộc cả Uri của Contact và hình thu nhỏ vào QuickContactBadge. Phương thức mặc định chỉ cho phép ánh xạ 1 : 1 giữa một cột và một View.

Đoạn mã sau trình bày ví dụ về một lớp con tùy chỉnh của CursorAdapter:

Định nghĩa adapter tùy chỉnh cho ListView

Đoạn mã định nghĩa một lớp con của CursorAdapter gồm phương thức khởi tạo của lớp con này, và ghi đè phương thức newView() và bindView():

```
/**
 *
 *
 */
private class ContactsAdapter extends CursorAdapter {
    private LayoutInflater mInflater;
    ...
    public ContactsAdapter(Context context) {
        super(context, null, 0);
        /*
         * Lấy bộ khai triển mà có thể khởi tạo
         * layout của ListView từ file
         */
        mInflater = LayoutInflater.from(context);
        ...
    }
    ...
```

```
/**
 * Định nghĩa một lớp chứa các resource ID của mỗi hàng
 * trong layout để tránh phải tra cứu chúng mỗi lần dữ liệu
 * được ràng buộc vào một hàng.
 */
private class ViewHolder {
   TextView displayname;
   QuickContactBadge quickcontact;
}
..
@Override
public View newView(
      Context context,
      Cursor cursor,
      ViewGroup viewGroup) {
   /* Khai triển item layout. Lưu các resource ID trong một
    * đối tượng ViewHolder để tránh phải tra cứu chúng
    * mỗi lần bindView() được gọi.
    */
   final View itemView =
         mInflater.inflate(
               R.layout.contact_list_layout,
               viewGroup,
               false
         );

   final ViewHolder holder = new ViewHolder();
   holder.displayname =
         (TextView) view.findViewById(R.id.displayname);
   holder.quickcontact = (QuickContactBadge)
         view.findViewById(R.id.quickcontact);
   view.setTag(holder);
   return view;
}
...
@Override
public void bindView(View view,Context context,
Cursor cursor) {
   final ViewHolder holder = (ViewHolder) view.getTag();
   final String photoData =
         cursor.getString(mPhotoDataIndex);

   final String displayName =
         cursor.getString(mDisplayNameIndex);
   ...
```

```
        // Thiết lập tên hiển thị trong layout
        holder.displayname = cursor.getString(mDisplayNameIndex);
        ...
        /*
         * Tạo Contact URI dùng cho QuickContactBadge.
         */
        final Uri contactUri = Contacts.getLookupUri(
                cursor.getLong(mIdIndex),
                cursor.getString(mLookupKeyIndex));
        holder.quickcontact.assignContactUri(contactUri);
        String photoData = cursor.getString(mPhotoDataIndex);
        /*
         * Giải mã file ảnh thu nhỏ thành Bitmap.
         * Phương thức loadContactPhotoThumbnail() được định
        * nghĩa trong phần "Thiết lập Contact URI và Hình thu nhỏ"
         */
        Bitmap thumbnailBitmap =
                loadContactPhotoThumbnail(photoData);
        /*
         * Thiết lập bức ảnh vào QuickContactBadge
         * QuickContactBadge kế thừa từ ImageView
         */
        holder.quickcontact.setImageBitmap(thumbnailBitmap);
    }
```

Thiết lập các biến

Trong đoạn mã, bạn cần thiết lập các biến bao gồm một phép chiếu cho Cursor chứa các cột cần thiết.

> **Lưu ý:** Đoạn mã sau sử dụng phương thức loadContactPhotoThumbnail(), được định nghĩa trong phần "Thiết lập Contact URI và Hình thu nhỏ"

Ví dụ:

```
public class ContactsFragment extends Fragment implements
LoaderManager.LoaderCallbacks<Cursor> {
...
    // Định nghĩa ListView
    private ListView mListView;
    // Định nghĩa ContactsAdapter
    private ContactsAdapter mAdapter;
    ...
    // Định nghĩa Cursor chứa dữ liệu truy xuất được
    private Cursor mCursor;
    /*
     * Định nghĩa một phép chiếu dựa trên phiên bản của nền tảng.
```

```
     * Điều này đảm bảo bạn truy xuất đúng các cột.
     */
    private static final String[] PROJECTION =
            {
                Contacts._ID,
                Contacts.LOOKUP_KEY,
                (Build.VERSION.SDK_INT >=
                 Build.VERSION_CODES.HONEYCOMB) ?
                        Contacts.DISPLAY_NAME_PRIMARY :
                        Contacts.DISPLAY_NAME
               (Build.VERSION.SDK_INT >=
                Build.VERSION_CODES.HONEYCOMB) ?
                        Contacts.PHOTO_THUMBNAIL_ID :
                        /*
                         * Dù không cần thiết đưa cột Contacts_ID vào
                         * hai lần, thao tác này giữ số lượng cột
                         * giống nhau bất kể phiên bản
                         */
                        Contacts_ID
                ...
            };
    /*
     * Để tiện, bạn định nghĩa các hằng số lưu
     * chỉ mục các cột trong Cursor. Chỉ mục được đánh
     * từ 0 và luôn khớp với thứ tự cột
     * trong phép chiếu.
     */
    // Chỉ mục của cột _ID
    private int mIdIndex = 0;
    // Chỉ mục của cột LOOKUP_KEY
    private int mLookupKeyIndex = 1;
    // Chỉ mục của cột tên hiển thị
    private int mDisplayNameIndex = 3;
    /*
     * Chỉ mục của cột dữ liệu ảnh.
     * Bằng PHOTO_THUMBNAIL_URI đối với Honeycomb và các phiên bản
     * sau đó, hoặc _ID cho các phiên bản trước đó.
     */
    private int mPhotoDataIndex =
            Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
            3:
            0;
    ...
```

Thiết lập ListView

Trong Fragment.onCreate(), bạn khởi tạo CursorAdapter tùy chỉnh và lấy tham chiếu tới đối tượng ListView:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        /*
         * Khởi tạo lớp con của
         * CursorAdapter
         */
        ContactsAdapter mContactsAdapter =
                new ContactsAdapter(getActivity());
        /*
         * Lấy một tham chiếu tới đối tượng ListView trong file
         * contact_list_layout.xml
         */
        mListView = (ListView) findViewById(R.layout.contact_list_layout);
        ...
    }
    ...
```

Trong onActivityCreated(), bạn ràng buộc ContactsAdapter vào ListView:

```
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        ...
        // Thiết lập adapter cho ListView
        mListView.setAdapter(mAdapter);
        ...
    }
    ...
```

Khi bạn nhận lại được đối tượng Cursor chứa dữ liệu các Contact, thông thường trong onLoadFinished(), bạn gọi swapCursor() để chuyển dữ liệu từ Cursor tới ListView. Thao tác này hiển thị QuickContactBadge cho mỗi dòng trong danh sách Contact:

```
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        // Khi việc nạp dữ liệu hoàn thành, chuyển dữ liệu trong
        // cursor vào trong adapter.
        mContactsAdapter.swapCursor(cursor);
    }
```

Khi bạn ràng buộc Cursor vào ListView bằng cách dùng một CursorAdapter (hoặc lớp con của CursorAdapter), và bạn sử dụng CursorLoader để nạp Cursor, hãy luôn nhớ xóa các tham chiếu tới đối tượng Cursor trong phần thân phương thức onLoaderReset() của bạn. Ví dụ:

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // Hủy tham chiếu tới Cursor trước
    mContactsAdapter.swapCursor(null);
}
```

9. Tạo các Giao diện người dùng tương thích ngược

Bài học này trình bày cách sử dụng các thành phần UI và API có sẵn trong những phiên bản mới của Android theo hướng tương thích ngược, đảm bảo ứng dụng vẫn chạy được trên các phiên bản Android cũ.

Trong suốt bài học này, tính năng Action Bar Tab[1] xuất hiện kể từ Android 3.0 (API cấp 11) sẽ được dùng làm ví dụ nhưng bạn có thể áp dụng các kỹ thuật này cho những thành phần UI và tính năng khác trong API.

NỘI DUNG BÀI HỌC

Trừu tượng hóa các API mới

Quyết định các tính năng và API nào ứng dụng cần. Học cách định nghĩa các giao diện Java trung gian đặc thù trên ứng dụng để trừu tượng hóa các lớp kế thừa thành phần UI trong ứng dụng của bạn.

Tạo Proxy[2] cho các API mới

Học cách tạo lớp kế thừa giao diện trong đó sử dụng các API mới hơn.

Tạo lớp kế thừa sử dụng các API cũ

Học cách tạo lớp tùy chỉnh kế thừa giao diện sử dụng các API cũ hơn.

Sử dụng Thành phần Tự Nhận biết Phiên bản

Học cách lựa chọn lớp kế thừa dùng khi ứng dụng chạy, và bắt đầu sử dụng giao diện đã tạo trong ứng dụng của bạn.

Trừu tượng hóa các API mới

Giả sử bạn muốn sử dụng các tab trong action bar làm thành phần điều hướng trong màn hình ban đầu của ứng dụng. Thật không may, các API của ActionBar chỉ có trong Android 3.0 hoặc cao hơn (API cấp 11+). Vì vậy, nếu muốn phân phối ứng dụng tới những thiết bị chạy những phiên bản Android cũ hơn, bạn cần cung cấp phần mã nguồn hỗ trợ API mới đồng thời cung cấp cơ chế dự phòng sử dụng các API cũ hơn.

[1] *Action Bar Tab: Là các tab nằm trong Action bar ở trên cùng của một ứng dụng Android, còn gọi là các tab điều hướng.*
[2] *Proxy: Trong trường hợp này, proxy là các lớp trung gian có giao diện giống như API cũ nhưng bên trong sử dụng tính năng của API mới.*

Trong bài học này, bạn sẽ tạo một thành phần Giao diện Người dùng (UI) dạng tab trong đó sử dụng các lớp trừu tượng cùng với các lớp kế thừa chuyên biệt cho từng phiên bản để cung cấp khả năng tương thích ngược. Bài học này mô tả cách tạo một tầng trừu tượng cho API Tab mới, đây là bước đầu tiên tiến đến xây dựng thành phần Tab.

Chuẩn bị cho việc Trừu tượng hóa

Trừu tượng hóa trong ngôn ngữ lập trình Java liên quan đến việc tạo một hoặc nhiều giao diện hoặc lớp trừu tượng (abstract) để ẩn đi phần viết mã chi tiết. Trong trường hợp có các Android API mới, bạn có thể sử dụng trừu tượng hóa để xây dựng những thành phần tự nhận biết phiên bản với khả năng dùng các API hiện có trên các thiết bị mới hơn, hoặc quay trở lại dùng các API cũ và tương thích cao hơn trên những thiết bị cũ hơn.

Khi sử dụng phương pháp này, trước tiên bạn quyết định các lớp mới hơn nào muốn sử dụng theo hướng tương thích ngược, sau đó tạo các lớp trừu tượng dựa trên các giao diện public của các lớp mới hơn này. Khi định nghĩa các giao diện trừu tượng, bạn nên làm giống API mới càng nhiều càng tốt. Điều này giúp tăng tối đa khả năng tương thích xuôi và giúp bạn dễ dàng hơn trong việc loại bỏ tầng trừu tượng hóa này trong tương lai khi không còn cần thiết.

Sau khi tạo các lớp trừu tượng cho những API mới này, bạn có thể tạo bao nhiêu lớp kế thừa tùy ý và chúng có thể được lựa chọn khi chạy. Nhằm đạt được khả năng tương thích ngược, các lớp kế thừa sẽ khác nhau bởi cấp API chúng cần dùng. Như vậy, một lớp kế thừa có thể sử dụng các API mới phát hành, trong khi các lớp khác sử dụng những API cũ hơn.

Tạo Giao diện Tab trừu tượng

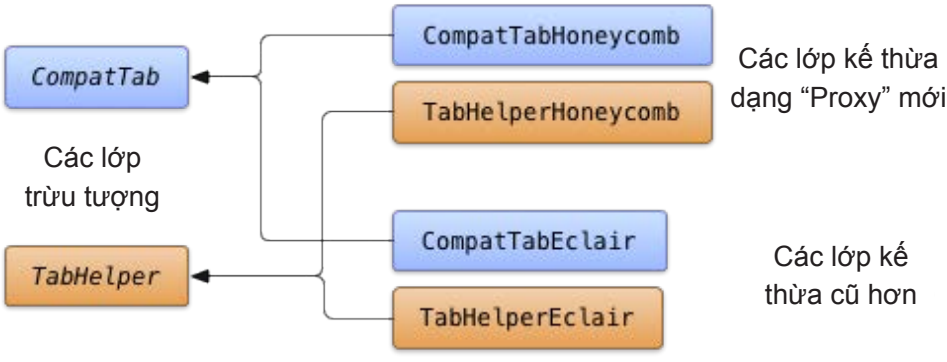
Để tạo một phiên bản tương thích ngược của Tab, trước tiên bạn cần quyết định những tính năng và các API cụ thể mà ứng dụng cần đến. Đối với thành phần Tab ở màn hình đầu tiên, giả sử bạn có những yêu cầu tính năng sau:

1. Thẻ tab cần hiển thị chữ và một biểu tượng.
2. Các Tab có thể liên kết với một thể hiện Fragment.
3. Activity cần có khả năng lắng nghe các sự kiện thay đổi tab.

Việc chuẩn bị trước các yêu cầu này cho phép bạn kiểm soát phạm vi của tầng trừu tượng hóa. Điều này nghĩa là bạn có thể dành ít thời gian hơn cho việc tạo nhiều lớp kế thừa của tầng trừu tượng hóa và có thể bắt đầu sử dụng lớp kế thừa tương thích ngược mới sớm hơn.

Các API chính dành cho Tab nằm trong ActionBar và ActionBar.Tab. Đây là các API cần trừu tượng hóa để tạo ra khả năng tự nhận biết phiên bản cho Tab. Các

yêu cầu trong dự án ví dụ này đòi hỏi tương thích ngược tới phiên bản Eclair (API cấp 5) đồng thời tận dụng các tính năng Tab mới trong Honeycomb (API cấp 11). Hình dưới đây minh họa một biểu đồ cấu trúc lớp hỗ trợ hai lớp kế thừa và các lớp (hoặc giao diện) trừu tượng cơ sở.



Hình 9.1. Biểu đồ lớp của các lớp trừu tượng cơ sở và các lớp kế thừa chuyên biệt cho từng phiên bản.

Trừu tượng hóa ActionBar.Tab

Công việc đầu tiên để xây dựng tầng trừu tượng hóa cho thành phần Tab là tạo một lớp trừu tượng biểu diễn Tab, lớp này ánh xạ giao diện ActionBar.Tab:

```
public abstract class CompatTab {
    ...
    public abstract CompatTab setText(int resId);
    public abstract CompatTab setIcon(int resId);
    public abstract CompatTab setTabListener(
                        CompatTabListener callback);
    public abstract CompatTab setFragment(Fragment fragment);

    public abstract CharSequence getText();
    public abstract Drawable getIcon();
    public abstract CompatTabListener getCallback();
    public abstract Fragment getFragment();
    ...
}
```

Bạn có thể sử dụng một lớp trừu tượng thay cho một giao diện để đơn giản hóa việc hiện thực hóa các tính năng chung như liên kết các đối tượng Tab với các đối tượng Activity (không trình bày trong đoạn mã này).

Trừu tượng hóa các Phương thức làm việc với Tab trong ActionBar

Tiếp theo, bạn định nghĩa một lớp trừu tượng cho phép tạo và gán các Tab vào một Activity, giống như phương thức ActionBar.newTab() và ActionBar.addTab():

```
public abstract class TabHelper {
    ...

    public CompatTab newTab(String tag) {
        // Phương thức này sẽ được hiện thực hóa ở bên dưới
    }
    public abstract void addTab(CompatTab tab);
    ...
}
```

Trong những phần tiếp theo, bạn sẽ viết các lớp kế thừa TabHelper và CompatTab với khả năng hoạt động trên cả phiên bản cũ và mới.

Tạo Proxy cho các API mới

Bài học này trình bày cách kế thừa các lớp trừu tượng CompatTab, TabHelper và sử dụng các API mới. Ứng dụng có thể sử dụng lớp kế thừa này trên các thiết bị chạy một phiên bản có hỗ trợ chúng.

Kế thừa Tab dùng các API mới

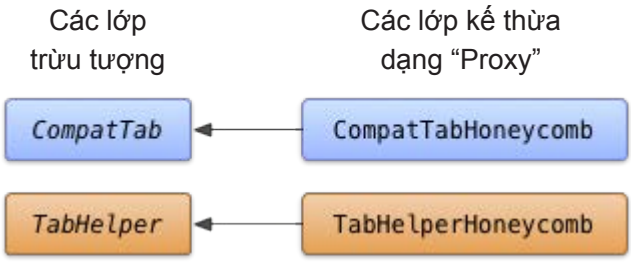
Các lớp thực của CompatTab và TabHelper trong đó sử dụng các API mới được gọi là lớp kế thừa dạng *proxy* (ủy nhiệm). Vì các lớp trừu tượng đã định nghĩa trong phần trước phản ánh các API mới (cấu trúc lớp, chữ ký phương thức…), các lớp thực sử dụng những API mới này đơn giản ủy nhiệm các lời gọi phương thức và trả về các kết quả từ đó.

Bạn có thể sử dụng trực tiếp các API mới trong những lớp thực này – mà không gây lỗi tắt đột ngột ứng dụng khi chạy trên các thiết bị cũ – nhờ cơ chế lazy loading[3] khi nạp các lớp. Các lớp được nạp và khởi tạo khi lớp được truy cập lần đầu tiên – gồm thao tác khởi tạo lớp hoặc truy cập một trong các trường hoặc phương thức static của nó lần đầu tiên. Như vậy, miễn là bạn không khởi tạo các lớp kế thừa chuyên biệt dành cho Honeycomb trên những thiết bị dùng phiên bản trước Honeycomb, Dalvik VM sẽ không ném ra ngoại lệ VerifyError.

Một quy ước đặt tên tốt cho lớp kế thừa này là nối thêm cấp API hoặc mã phiên bản tương ứng với các API cần dùng cho tên lớp thực. Ví dụ, lớp kế thừa Tab

[3] *Lazy loading: Là kỹ thuật khởi tạo một tài nguyên chỉ khi có lệnh truy cập tài nguyên đó.*

thuần có thể là CompatTabHoneycomb and TabHelperHoneycomb, vì chúng dựa trên các API có trong Android 3.0 (API cấp 11) hoặc cao hơn.



Hình 9.2. Biểu đồ lớp cho lớp kế thừa Tab trên Honeycomb.

Viết mã CompatTabHoneycomb

CompatTabHoneycomb là lớp kế thừa của lớp trừu tượng CompatTab mà TabHelperHoneycomb dùng để tham chiếu tới mỗi Tab. CompatTabHoneycomb đơn giản ủy nhiệm tất cả các lời gọi phương thức tới đối tượng ActionBar.Tab chứa bên trong nó.

Sau đây là mã nguồn CompatTabHoneycomb trong đó sử dụng các API mới của ActionBar.Tab:

```
public class CompatTabHoneycomb extends CompatTab {
    // Đối tượng Tab thuần được đối tượng CompatTab này ủy nhiệm
    ActionBar.Tab mTab;
    ...

    protected CompatTabHoneycomb(FragmentActivity activity,
    String tag) {
        ...
        // Ủy nhiệm tới API mới của ActionBar.newTab
        mTab = activity.getActionBar().newTab();
    }

    public CompatTab setText(int resId) {
        // Ủy nhiệm tới API mới của ActionBar.Tab.setText
        mTab.setText(resId);
        return this;
    }

    ...
    // Thực hiện tương tự cho các thuộc tính khác (biểu tượng,
    // callback...)
}
```

Thực thi TabHelperHoneycomb

TabHelperHoneycomb là lớp kế thừa của lớp trừu tượng TabHelper trong đó ủy nhiệm các lời gọi phương thức tới một đối tượng ActionBar thực được lấy từ Activity chứa nó.

Sau đây là mã nguồn của lớp TabHelperHoneycomb, trong đó các lời gọi phương thức được ủy nhiệm tới API của ActionBar:

```
public class TabHelperHoneycomb extends TabHelper {
    ActionBar mActionBar;
    ...

    protected void setUp() {
        if (mActionBar == null) {
            mActionBar = mActivity.getActionBar();
            mActionBar.setNavigationMode(
                    ActionBar.NAVIGATION_MODE_TABS);
        }
    }

    public void addTab(CompatTab tab) {
        ...
        // tab là một thể hiện của CompatTabHoneycomb, vì vậy
        // đối tượng Tab thuần của nó là một ActionBar.Tab.
        mActionBar.addTab((ActionBar.Tab) tab.getTab());
    }

    // Phương thức quan trọng còn lại là newTab() nằm trong
    // TabHelper - lớp cha.
}
```

Tạo lớp kế thừa dùng API cũ

Bài học này trình bày cách tạo lớp kế thừa giống với giao diện API mới hơn nhưng đồng thời hỗ trợ các thiết bị cũ hơn.

Quyết định một Giải pháp Thay thế

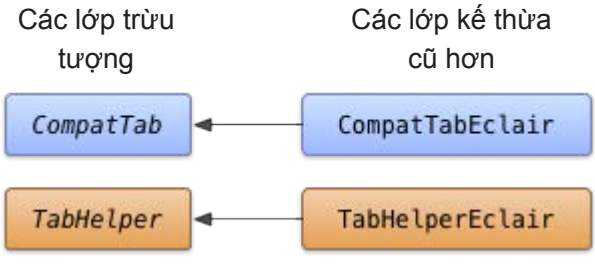
Nhiệm vụ khó khăn nhất trong việc sử dụng các tính năng UI mới hơn theo hướng tương thích ngược là quyết định và thực thi giải pháp dự phòng cho các phiên bản cũ hơn. Trong nhiều trường hợp, ta có thể thực hiện các chức năng của thành phần UI mới bằng cách sử dụng tính năng của các phiên bản UI cũ hơn. Ví dụ:

- Action bar (thanh Action) có thể được hiện thực hóa bằng cách dùng một LinearLayout nằm ngang chứa các nút bấm dạng ảnh, dưới dạng title bar (thanh tiêu đề) tùy chỉnh hoặc dưới dạng View trong layout của Activity. Phần Action ẩn (Overflow action) có thể được hiển thị khi người dùng nhấn nút *Menu* trên thiết bị.
- Các Tab trên Action bar có thể được hiện thực hóa bằng cách dùng một LinearLayout nằm ngang chứa các nút bấm, hoặc sử dụng thành phần UI TabWidget.
- NumberPicker và Switch có thể được hiện thực hóa bằng cách dùng Spinner và ToggleButton tương ứng.
- ListPopupWindow và PopupMenu có thể được hiện thực hóa bằng cách dùng PopupWindow.

Nhìn chung, không có giải pháp kiểu "một cỡ vừa cho tất cả" cho việc chuyển ngược các thành phần UI mới sang các thiết bị cũ. Bạn hãy lưu ý tới trải nghiệm người dùng: trên các thiết bị cũ hơn, người dùng có thể không quen với mẫu thiết kế mới và những thành phần UI mới. Bạn cần cân nhắc làm cách nào để cung cấp tính năng tương tự bằng các thành phần quen thuộc. Trong nhiều trường hợp, đây không phải vấn đề đáng lo – nếu là các thành phần UI phổ biến trong hệ sinh thái ứng dụng (ví dụ như Action bar) hoặc khi mô hình tương tác cực kỳ đơn giản và trực quan (như thao tác vuốt khi dùng ViewPager).

Lớp kế thừa Tab dùng các API cũ

Để tạo lớp kế thừa cũ hơn cho các Tab trên Action bar, bạn có thể sử dụng TabWidget và TabHost (cũng có cách khác là sử dụng các Button đặt ngang). Bạn kế thừa các lớp có tên TabHelperEclair và CompatTabEclair, vì lớp kế thừa này sử dụng các API xuất hiện trong các phiên bản Android 2.0 (Eclair) trở về trước.



Hình 9.3. Biểu đồ lớp cho các lớp kế thừa Tab trên Eclair.

CompatTabEclair lưu các thuộc tính của Tab như tên và biểu tượng trong các biến thể hiện, vì không có đối tượng ActionBar.Tab để chứa các thông tin đó:

```
public class CompatTabEclair extends CompatTab {
   // Lưu các thuộc tính này trong biến thể hiện,
   // vì không có đối tượng ActionBar.Tab.
   private CharSequence mText;
   ...

   public CompatTab setText(int resId) {
      // Phần mã hỗ trợ API cũ đơn giản lưu thông tin này
      // trong thể hiện của đối tượng.
      mText = mActivity.getResources().getText(resId);
      return this;
   }

   ...
   // Thực hiện tương tự cho các thuộc tính khác (biểu tượng,
   // callback, ...)
}
```

TabHelperEclair tận dụng các phương thức của TabHost để tạo đối tượng TabHost.TabSpec và phần chỉ thị Tab (tab indicator):

```
public class TabHelperEclair extends TabHelper {
   private TabHost mTabHost;
   ...

   protected void setUp() {
      if (mTabHost == null) {
         // Layout của Activity dành cho các thiết bị
         // trước phiên bản Honeycomb phải chứa một TabHost.
         mTabHost = (TabHost) mActivity.findViewById(
                                        android.R.id.tabhost);
         mTabHost.setup();
      }
   }
  public void addTab(CompatTab tab) {
      ...
      TabSpec spec = mTabHost
            .newTabSpec(tag)
            .setIndicator(tab.getText()); // Và một biểu tượng
                                          // tùy chọn
      ...
      mTabHost.addTab(spec);
   }

   // Phương thức quan trọng còn lại là newTab() nằm trong
   // TabHelper - lớp cha.
}
```

Giờ bạn đã có hai lớp kế thừa của lớp trừu tượng CompatTab và TabHelper: một hoạt động trên các thiết bị chạy Android 3.0 hoặc cao hơn trong đó sử dụng các API mới, và một hoạt động trên các thiết bị chạy Android 2.0 hoặc cao hơn trong đó sử dụng các API cũ hơn. Phần tiếp theo trình bày cách sử dụng các lớp này trong ứng dụng của bạn.

Sử dụng Thành phần Tự nhận biết Phiên bản

Giờ bạn đã có hai lớp kế thừa của lớp trừu tượng TabHelper và CompatTab—một cho Android 3.0 hoặc cao hơn và một cho các phiên bản trước đó — đã đến lúc làm gì đó với hai lớp này. Bài học này trình bày cách viết mã chuyển đổi giữa hai lớp này, cách tạo các layout tự nhận biết phiên bản và cuối cùng là cách sử dụng thành phần UI tương thích ngược.

Viết Lôgic Chuyển đổi

Lớp trừu tượng TabHelper đóng vai trò là một factory tạo các thể hiện kiểu TabHelper and CompatTab thích hợp cho từng phiên bản dựa trên phiên bản Android của thiết bị hiện tại:

```
public abstract classTabHelper {
    ...
    // Cách dùng: TabHelper.createInstance(activity)
    public static TabHelper createInstance(FragmentActivity activity) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            return new TabHelperHoneycomb(activity);
        } else {
            return new TabHelperEclair(activity);
        }
    }

    // Cách dùng: mTabHelper.newTab("tag")
    public CompatTab newTab(String tag) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
          return new CompatTabHoneycomb(mActivity, tag);
        } else {
            return new CompatTabEclair(mActivity, tag);
        }
    }
    ...
}
```

Tạo Layout Tự nhận biết Phiên bản cho Activity

Bước tiếp theo là cung cấp các layout cho Activity của bạn để có thể hỗ trợ hai lớp hiện thực hóa Tab. Đối với lớp kế thừa cũ hơn (TabHelperEclair), bạn cần đảm bảo layout chứa một TabWidget và một TabHost, cùng với một đối tượng chứa nội dung của Tab:

res/layout/main.xml:

```
<!-- Layout này chỉ dành cho API cấp 5-10. -->
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@android:id/tabhost"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <LinearLayout
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:padding="5dp">

      <TabWidget
         android:id="@android:id/tabs"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />

      <FrameLayout
         android:id="@android:id/tabcontent"
         android:layout_width="match_parent"
         android:layout_height="0dp"
         android:layout_weight="1" />

   </LinearLayout>
</TabHost>
```

Đối với lớp kế thừa TabHelperHoneycomb, tất cả những thứ bạn cần là một FrameLayout để chứa nội dung Tab, vì các phần chỉ thị tab được cung cấp bởi đối tượng ActionBar:

```
res/layout-v11/main.xml:
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@android:id/tabcontent"
   android:layout_width="match_parent"
   android:layout_height="match_parent" />
```

Lúc chạy, Android sẽ quyết định khai triển phiên bản main.xml nào dựa trên phiên bản của Android. Quyết định này cũng giống như lôgic trong phần trước khi bạn lựa chọn bản thực thi TabHelper nào để sử dụng.

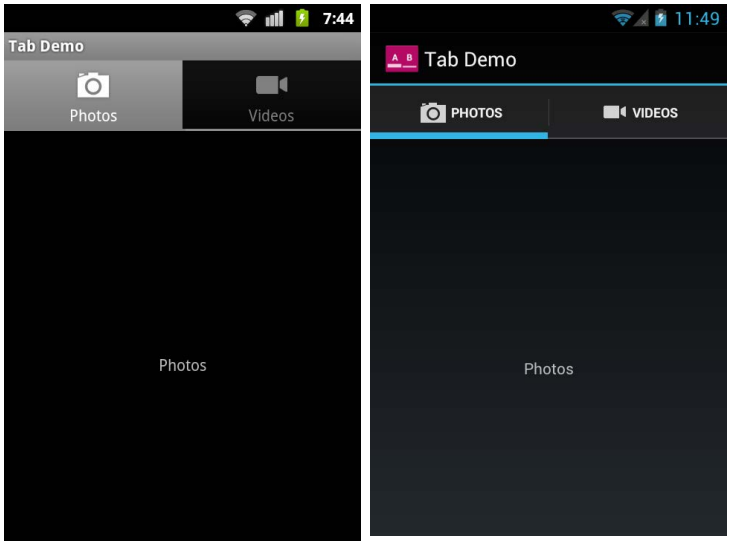
Sử dụng TabHelper trong Activity của bạn

Trong phương thức onCreate() của Activity, bạn có thể lấy một đối tượng TabHelper và thêm mới các Tab với đoạn mã sau:

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

    TabHelper tabHelper = TabHelper.createInstance(this);
    tabHelper.setUp();

    CompatTab photosTab = tabHelper
            .newTab("photos")
            .setText(R.string.tab_photos);
    tabHelper.addTab(photosTab);

    CompatTab videosTab = tabHelper
            .newTab("videos")
            .setText(R.string.tab_videos);
    tabHelper.addTab(videosTab);
}
```

Khi chạy ứng dụng, đoạn mã này khai triển đúng layout của Activity và khởi tạo hoặc đối tượng TabHelperHoneycomb hoặc đối tượng TabHelperEclair. Lớp thực tế được sử dụng hoàn toàn không được Activity biết tới vì chúng làm việc với giao diện chung TabHelper.

Dưới đây là ảnh chụp màn hình của mã nguồn chạy trên thiết bị Android 2.3 và Android 4.0.



Hình 9.4. Ảnh chụp màn hình minh họa Tab tương thích ngược chạy trên một thiết bị Android 2.3 (sử dụng TabHelperEclair) và một thiết bị Android 4.0 (sử dụng TabHelperHoneycomb).