



# Google Android官方培训教程

## 中文版

# Table of Contents

---

1. 序言
2. Android入門基礎：從這裏開始
  - i. 建立第一個App
    - i. 創建Android項目
    - ii. 執行Android程序
    - iii. 建立簡單的用戶界面
    - iv. 啓動其他的Activity
  - ii. 添加ActionBar
    - i. 建立ActionBar
    - ii. 添加ActionBar按鈕
    - iii. 自定義ActionBar的風格
    - iv. ActionBar的覆蓋層疊
  - iii. 兼容不同的設備
    - i. 適配不同的語言
    - ii. 適配不同的屏幕
    - iii. 適配不同的系統版本
  - iv. 管理Activity的生命週期
    - i. 啓動與銷燬Activity
    - ii. 暫停與恢復Activity
    - iii. 停止與重啓Activity
    - iv. 重新創建Activity
  - v. 使用Fragment建立動態的UI
    - i. 創建一個Fragment
    - ii. 建立靈活動態的UI
    - iii. Fragments之間的交互
  - vi. 數據保存
    - i. 保存到Preference
    - ii. 保存到文件
    - iii. 保存到數據庫
  - vii. 與其他應用的交互
    - i. Intent的發送
    - ii. 接收Activity返回的結果
    - iii. Intent過濾
3. Android分享操作
  - i. 分享簡單的數據
    - i. 紿其他App發送簡單的數據
    - ii. 接收從其他App返回的數據
    - iii. 紿ActionBar增加分享功能
  - ii. 分享文件
    - i. 建立文件分享
    - ii. 分享文件
    - iii. 請求分享一個文件
    - iv. 獲取文件信息
  - iii. 使用NFC分享文件
    - i. 發送文件給其他設備
    - ii. 接收其他設備的文件
4. Android多媒體

- i. 管理音頻播放
  - i. 控制音量與音頻播放
  - ii. 管理音頻焦點
  - iii. 兼容音頻輸出設備

- ii. 拍照
  - i. 簡單的拍照
  - ii. 簡單的錄像
  - iii. 控制相機硬件
- iii. 打印
  - i. 打印照片
  - ii. 打印HTML文檔
  - iii. 打印自定義文檔

## 5. Android圖像與動畫

- i. 高效顯示Bitmap
  - i. 高效加載大圖
  - ii. 非UI線程處理Bitmap
  - iii. 緩存Bitmap
  - iv. 管理Bitmap的內存
  - v. 在UI上顯示Bitmap
- ii. 使用OpenGL ES顯示圖像
  - i. 建立OpenGL ES的環境
  - ii. 定義Shapes
  - iii. 繪製Shapes
  - iv. 運用投影與相機視圖
  - v. 添加移動
  - vi. 響應觸摸事件
- iii. 添加動畫
  - i. View間漸變
  - ii. 使用ViewPager實現屏幕側滑
  - iii. 展示卡片翻轉動畫
  - iv. 縮放View
  - v. 佈局變更動畫

## 6. Android網絡連接與雲服務

- i. 無線連接設備
  - i. 使得網絡服務可發現
  - ii. 使用WiFi建立P2P連接
  - iii. 使用WiFi P2P服務
- ii. 執行網絡操作
  - i. 連接到網絡
  - ii. 管理網絡
  - iii. 解析XML數據
- iii. 高效下載
  - i. 為網絡訪問更加高效而優化下載
  - ii. 最小化更新操作的影響
  - iii. 避免下載多餘的數據
  - iv. 根據網絡類型改變下載模式
- iv. 雲同步
  - i. 使用備份API
  - ii. 使用Google Cloud Messaging
  - v. 解決雲同步的保存衝突
  - vi. 使用Sync Adapter傳輸數據

- i. 創建Stub授權器
  - ii. 創建Stub Content Provider
  - iii. 創建Sync Adapter
  - iv. 執行Sync Adapter
- vii. 使用Volley執行網絡數據傳輸
- i. 發送簡單的網絡請求
  - ii. 建立請求隊列
  - iii. 創建標準的網絡請求
  - iv. 實現自定義的網絡請求
7. Android聯繫人與位置信息
- i. Android聯繫人信息
    - i. 獲取聯繫人列表
    - ii. 獲取聯繫人詳情
    - iii. 使用Intents修改聯繫人信息
    - iv. 顯示聯繫人頭像
  - ii. Android位置信息
    - i. 獲取最後可知位置
    - ii. 獲取位置更新
    - iii. 顯示位置地址
    - iv. 創建和監視地理圍欄
8. Android可穿戴應用
- i. 賦予Notification可穿戴特性
    - i. 創建Notification
    - ii. 在Notification中接收語音輸入
    - iii. 為Notification添加顯示頁面
    - iv. 以Stack的方式顯示Notifications
  - ii. 創建可穿戴的應用
    - i. 創建並執行可穿戴應用
    - ii. 創建自定義的佈局
    - iii. 添加語音能力
    - iv. 打包可穿戴應用
    - v. 通過藍牙進行調試
  - iii. 創建自定義的UI
    - i. 定義Layouts
    - ii. 創建Cards
    - iii. 創建Lists
    - iv. 創建2D-Picker
    - v. 創建確認界面
    - vi. 退出全屏的Activity
  - iv. 發送並同步數據
    - i. 訪問可穿戴數據層
    - ii. 同步數據單元
    - iii. 傳輸資源
    - iv. 發送與接收消息
    - v. 處理數據層的事件
9. Android TV應用
- i. 創建TV應用
    - i. 創建TV應用的第一步
    - ii. 處理TV硬件部分
    - iii. 創建TV的佈局文件
    - iv. 創建TV的導航欄

- ii. 創建TV播放應用
  - i. 創建目錄瀏覽器
  - ii. 提供一個Card視圖
  - iii. 創建詳情頁
  - iv. 顯示正在播放卡片
- iii. 幫助用戶在TV上探索內容
  - i. TV上的推薦內容
  - ii. 使得TV App能夠被搜索
  - iii. 使用TV應用進行搜索
- iv. 創建TV遊戲應用
- v. 創建TV直播應用
- vi. TV Apps Checklist

## 10. Android企業級應用

- i. Ensuring Compatibility with Managed Profiles
- ii. Implementing App Restrictions
- iii. Building a Work Policy Controller

## 11. Android交互設計

- i. 設計高效的導航
  - i. 規劃屏幕界面與他們之間的關係
  - ii. 為多種大小的屏幕進行規劃
  - iii. 提供向下和橫嚮導航
  - iv. 提供向上和歷史導航
  - v. 綜合：設計樣例 App
- ii. 實現高效的導航
  - i. 使用Tabs創建Swipe視圖
  - ii. 創建抽屜導航
  - iii. 提供向上的導航
  - iv. 提供向後的導航
  - v. 實現向下的導航
- iii. 通知提示用戶
  - i. 建立Notification
  - ii. 當啓動Activity時保留導航
  - iii. 更新Notification
  - iv. 使用BigView風格
  - v. 顯示Notification進度
- iv. 增加搜索功能
  - i. 建立搜索界面
  - ii. 保存並搜索數據
  - iii. 保持向下兼容
- v. 使得你的App內容可被Google搜索
  - i. 為App內容開啓深度鏈接
  - ii. 為索引指定App內容

## 12. Android界面設計

- i. 為多屏幕設計
  - i. 兼容不同的屏幕大小
  - ii. 兼容不同的屏幕密度
  - iii. 實現可適應的UI
- ii. 創建自定義View
  - i. 創建自定義的View類
  - ii. 實現自定義View的繪製
  - iii. 使得View可交互

- iv. 優化自定義View
- iii. 創建向後兼容的UI
  - i. 抽象新的APIs
  - ii. 代理至新的APIs
  - iii. 使用舊的APIs實現新API的效果
  - iv. 使用版本敏感的組件
- iv. 實現輔助功能
  - i. 開發輔助程序
  - ii. 開發輔助服務
- v. 管理系統UI
  - i. 淡化系統Bar
  - ii. 隱藏系統Bar
  - iii. 隱藏導航Bar
  - iv. 全屏沉浸式應用
  - v. 韻應UI可見性的變化
- vi. 創建使用Material Design的應用
  - i. 開始使用Material Design
  - ii. 使用Material的主題
  - iii. 創建Lists與Cards
  - iv. 定義Shadows與Clipping視圖
  - v. 使用Drawables
  - vi. 自定義動畫
  - vii. 維護兼容性

### 13. Android用戶輸入

- i. 使用觸摸手勢
  - i. 檢測常用的手勢
  - ii. 跟蹤手勢移動
  - iii. Scroll手勢動畫
  - iv. 處理多觸摸手勢
  - v. 拖拽與縮放
  - vi. 管理ViewGroup中的觸摸事件
- ii. 處理鍵盤輸入
  - i. 指定輸入法類型
  - ii. 處理輸入法可見性
  - iii. 兼容鍵盤導航
  - iv. 處理按鍵動作
- iii. 兼容遊戲控制器
  - i. 處理控制器輸入動作
  - ii. 支持不同的Android系統版本
  - iii. 支持多個控制器

### 14. Android後臺任務

- i. 在IntentService中執行後臺任務
  - i. 創建IntentService
  - ii. 發送工作任務到IntentService
  - iii. 報告後臺任務執行狀態
- ii. 使用CursorLoader在後臺加載數據
  - i. 使用CursorLoader執行查詢任務
  - ii. 處理查詢的結果
- iii. 管理設備的喚醒狀態
  - i. 保持設備的喚醒
  - ii. 制定重複定時的任務

## 15. Android性能優化

- i. 管理應用的內存
- ii. 代碼性能優化建議
- iii. 提升Layout的性能
  - i. 優化layout的層級
  - ii. 使用include標籤重用layouts
  - iii. 按需加載視圖
  - iv. 使得ListView滑動順暢
- iv. 優化電池壽命
  - i. 監測電量與充電狀態
  - ii. 判斷與監測Docking狀態
  - iii. 判斷與監測網絡連接狀態
  - iv. 根據需要操作Broadcast接受者
- v. 多線程操作
  - i. 在一個線程中執行一段特定的代碼
  - ii. 為多線程創建線程池
  - iii. 啓動與停止線程池中的線程
  - iv. 與UI線程通信
  - vi. 避免出現程序無響應ANR
  - vii. JNI使用指南
  - viii. 優化多核處理器(SMP)下的Android程序

## 16. Android安全與隱私

- i. Security Tips
- ii. 使用HTTPS與SSL
- iii. 為防止SSL漏洞而更新Security
- iv. 使用設備管理條例增強安全性

## 17. Android測試程序

- i. 測試你的Activity
  - i. 建立測試環境
  - ii. 創建與執行測試用例
  - iii. 測試UI組件
  - iv. 創建單元測試
  - v. 創建功能測試

## 18. Glossary

# Android官方培訓課程中文版(v0.9.2)

---



Google Android團隊在2012年的時候開設了Android Training板塊 -

<http://developer.android.com/training/index.html>，這些課程是學習Android應用開發的絕佳資料。我們通過Github發起開源協作翻譯的項目，完成中文版的輸出，歡迎大家傳閱學習！項目難免會有很多寫的不對不好的地方，歡迎讀者加入此協作項目，進行糾錯，為完善這份教程貢獻你的一點力量！

## Github託管主頁

---

<https://github.com/kesenhoo/android-training-course-in-chinese>

請讀者點擊Star進行關注並支持！

## 在線閱讀

---

<http://hukai.me/android-training-course-in-chinese/index.html>

## 離線文檔

---

### 下載鏈接

- PDF: <http://hukai.me/eBooks/AndroidTrainingCHS.pdf>
- ePUB: <http://hukai.me/eBooks/AndroidTrainingCHS.epub>
- MOBI: <http://hukai.me/eBooks/AndroidTrainingCHS.mobi>

### 更新記錄

- v0.9.2 - 2015/03/30
- v0.9.1 - 2015/03/14
- v0.9.0 - 2015/03/09
- v0.8.0 - 2015/02/12
- v0.7.0 - 2014/11/30

- v0.6.0 - 2014/11/02
- v0.5.0 - 2014/10/18
- v0.4.0 - 2014/09/11
- v0.3.0 - 2014/08/31
- v0.2.0 - 2014/08/14
- v0.1.0 - 2014/08/05

## 參與方式

---

你可以選擇以下的方式幫忙修改糾正這份教程（推薦使用方法1）：

1. 通過[在線閱讀](#)課程的頁面，找到[Github倉庫](#)對應的章節文件，直接在線編輯修改提交即可。
2. 在線閱讀的文章底部留言，提出問題與修改意見，我們會及時處理。
3. 寫郵件給發起人：[胡凱](#)，郵箱是kesenhoo at gmail.com，郵件內容註明需要糾正的章節段落位置，並給出糾正的建議。

你也可以選擇加入QQ羣和學習Training課程的小夥伴一起討論交流：

- Android Training基礎羣，適合剛接觸Android Training課程的同學。
  - 基礎羣(1)：363415744，已滿
  - 基礎羣(2)：399077455
- Android Training進階羣：414115939，適合實際Android開發經驗1-3年的同學。
- Android Training高級羣：399096506，理論上Android開發經驗至少3年以上，部分能力突出的也可以申請，請在申請入羣的時候填寫能力舉證(例如，XXX公司Android高級開發/個人博客/Github帳號等等)，謝謝配合！

## 課程結構

---

- [序言](#)
- [Android入門基礎：從這裏開始](#)
  - [建立你的第一個App](#) - @yuanfentiank789
  - [添加ActionBar](#) - @vincent4j
  - [兼容不同的設備](#) - @Lin-H
  - [管理Activity的生命週期](#) - @kesenhoo
  - [使用Fragment建立動態的UI](#) - @fastcome1985
  - [數據保存](#) - @kesenhoo
  - [與其他應用的交互](#) - @kesenhoo
- [Android分享操作](#)
  - [分享簡單的數據](#) - @kesenhoo
  - [分享文件](#) - @jdneo
  - [使用NFC分享文件](#) - @jdneo
- [Android多媒體](#)
  - [管理音頻播放](#) - @kesenhoo
  - [拍照](#) - @kesenhoo
  - [打印](#) - @jdneo
- [Android圖像與動畫](#)
  - [高效顯示Bitmap](#) - @kesenhoo
  - [使用OpenGL ES顯示圖像](#) - @jdneo
  - [添加動畫](#) - @XizhiXu
- [Android網絡連接與雲服務](#)
  - [無線連接設備](#) - @naizhengtan

- 網絡連接操作 - @kesenhoo
- 高效下載 - @kesenhoo
- 使用Sync Adapter傳輸數據 - @jdneo
- 使用Volley執行網絡數據傳輸 - @kesenhoo
- 雲同步 - @kesenhoo , @jdneo
- 解決雲同步的保存衝突 - @jdneo
- Android聯繫人與位置信息
  - Android聯繫人信息 - @spencer198711
  - Android位置信息 - @penkzhou
- Android可穿戴應用
  - 賦予Notification可穿戴的特性 - @wangyachen
  - 創建可穿戴的應用 - @kesenhoo
  - 創建自定義的UI - @Roya
  - 發送並同步數據 - @wly2014
- Android企業級應用
  - Ensuring Compatibility with Managed Profiles - @2015/03/14 - 待認領
  - Implementing App Restrictions - @2015/03/14 - 待認領
  - Building a Work Policy Controller - @2015/03/14 - 待認領
- Android TV應用
  - 創建TV應用 - @awong1900
  - 創建TV播放應用 - @huanglizhuo
  - 幫助用戶在TV上探索內容 - @awong1900
  - 創建TV遊戲應用 - @dupengwei
  - 創建TV直播應用 - @dupengwei
  - TV應用清單 - @awong1900
- Android交互設計
  - 設計高效的導航 - @XizhiXu
  - 實現高效的導航 - @Lin-H
  - 通知提示用戶 - @fastcome1985
  - 增加搜索功能 - @Lin-H
  - 使得你的App內容可被Google搜索 - @Lin-H
- Android界面設計
  - 爲多屏幕設計 - @riverfeng
  - 創建自定義View - @kesenhoo
  - 創建向後兼容的UI - @spencer198711
  - 實現輔助功能 - @KOST
  - 管理系統UI - @KOST
  - 創建Material Design的應用 - @allenlsy
- Android用戶輸入
  - 使用觸摸手勢 - @Andrwyw
  - 處理鍵盤輸入 - @zhaochunqi
  - 兼容遊戲控制器 - @heray1990 - 30%
- Android後臺任務
  - 在IntentService中執行後臺任務 - @kesenhoo
  - 在後臺加載數據 - @kesenhoo
  - 管理設備的喚醒狀態 - @jdneo,@lltowq
- Android性能優化
  - 管理應用的內存 - @kesenhoo
  - 性能優化Tips - @kesenhoo
  - 提升Layout的性能 - @allenlsy
  - 優化電池壽命 - @kesenhoo

- 多線程操作 - @AllenZheng1991
- 避免程序無響應ANR - @kesenhoo
- JNI Tips - @pedant
- 優化多核處理器(SMP)下的Android程序 - @kesenhoo - 20%
- Android安全與隱私
  - Security Tips - @craftsmanBai
  - 使用HTTPS與SSL - @craftsmanBai
- Android測試程序
  - 測試你的Activity - @huanglizhuo

## 致謝

---

發起這個項目之後，得到很多人的支持，有經驗豐富的Android開發者，也有剛接觸Android的愛好者。他們有些已經上班，有些還是學生，有些在國內，還有的在國外！感謝所有參與或者關注這個項目的小夥伴！

下面是參與翻譯的小夥伴(Github ID按照課程結構排序)：

0	1	2
@yuanfentiank789	@vincent4j	@Lin-H
@kesenhoo	@fastcome1985	@jdneo
@XizhiXu	@naizhengtan	@spencer198711
@penzhou	@wangyachen	@wly2014
@fastcome1985	@riverfeng	@xrayzh
@K0ST	@Andrwyw	@zhaochunqi
@lltowq	@allenlsy	@AllenZheng1991
@pedant	@craftsmanBai	@huanglizhuo
@Roya	@awong1900	@dupengwei
0:10	1:10	2:10

@發起人:胡凱，博客：<http://hukai.me>，Github：<https://github.com/kesenhoo>，微博：<http://weibo.com/kesenhoo>

還有衆多參與糾錯校正的同學名字就不一一列舉了，謝謝所有關注這個項目的小夥伴！特別感謝[安卓巴士社區](#)，[愛開發社區](#)，[碼農週刊](#)對項目的宣傳！

## License

---

本站作品由<https://github.com/kesenhoo/android-training-course-in-chinese>創作，採用[知識共享 署名-非商業性使用-相同方式共享 4.0 國際 許可](#)協議進行許可。

# Android入門基礎：從這裏開始

編寫:kesenhoo - 原文:<http://developer.android.com/training/index.html>

歡迎來到為Android開發者準備的培訓項目。在這裏你會找到一系列的課程，這些課程會演示你如何使用可重用的代碼來完成特定的任務。所有的課程分為若干不同的小組。你可以通過左邊的導航來查看。

第1組：“從這裏開始”，教你Android應用開發的最基本的知識。如果你是一個Android應用開發的新手，你應該按照順序學習完下面的課程：

## 建立你的第一個App(Building Your First App)

在你安裝Android SDK之後，從這節課開始學習Android應用開發的基礎知識。

## 添加ActionBar(Adding the Action Bar)

ActionBar是你的Activity中最重要的設計元素之一。儘管ActionBar是從API 11開始被引入的，你仍然可以從Android 2.1開始使用Support Library去實現ActionBar。

## 兼容不同的設備(Supporting Different Devices)

學習給應用提供可選擇的資源文件來實現如何使用一個APK來使得你的應用能夠在不同的設備上獲取到最佳的用戶體驗。

## 管理Activity的生命週期(Managing the Activity Lifecycle)

學習Android的Activity的創建與銷燬，學習如何通過實現生命週期的回調方法來創建一個無縫的用戶體驗。

## 使用Fragment建立動態的UI(Building a Dynamic UI with Fragments)

學習如何為你的應用建立一套足夠靈活的UI，這套UI能夠在大屏幕的設備上顯示多個UI組件，在小屏幕的設備上呈現緊湊的UI組件。這使得你能夠為手機與平板只建立同一個APK。

## 數據保存(Saving Data)

學習如何在設備上保存數據。無論這些數據是臨時的文件，應用下載的資源，用戶的多媒體數據，結構化的數據還是其他。

## 與其他應用的交互(Interacting with Other Apps)

學習如何利用其他已經存在應用的既有功能來執行更進一步的用戶任務。例如拍照或者在地圖上查看某個地址。

# 建立第一個App

編寫:yuanfentiank789 - 原文:<http://developer.android.com/training/basics/firstapp/index.html>

歡迎開始Android應用開發之旅！

本章節我們將學習如何建立我們的第一個Android應用程序。我們將學到如何創建一個Android項目和運行可調試版本的應用程序。此外，我們還將學習到一些Android應用程序設計的基礎知識，包括如何創建一個簡單的用戶界面，以及處理用戶輸入。

開始本章節學習之前，我們要確保已經安裝了開發環境。我們需要：

- 1 下載[Android Studio](#)。
- 2 使用[SDK Manager](#)下載最新的SDK tools和platforms。

Note：雖然大多數章節期望你使用Android Studio完成開發，但對SDK tools來說，有的提供了命令行的方式運行。

本章節通過嚮導的方式逐步建立一個小型的Android應用，來教給我們一些Android開發的基本概念，因此對入門來說每一步都很重要。

[開始學習](#)

# 創建Android項目

編寫:yuanfentiank789 - 原文:<http://developer.android.com/training/basics/firstapp/creating-project.html>

一個Android項目包含了所有構成Android應用的源代碼文件。

本小節介紹如何使用Android Studio或SDK Tools命令行來創建一個新的項目。

Note：在此之前，我們應該已經安裝了Android SDK，如果使用Android Studio開發，應該確保已經安裝了Android Studio。否則，請先閱讀 [Installing the Android SDK](#)按照嚮導完成步驟安裝。

## 使用Android Studio創建項目

### 1. 使用Android Studio創建Android項目，啓動Android Studio

- 如果我們還沒有用Android Studio打開過項目，會看到歡迎頁，點擊New Project
- 如果已經用Android Studio打開過項目，點擊File=>New Project新建項目

### 2. 參照圖1在彈出的窗口（Configure your new project）中填入內容，點擊Next.

按照要求使用如圖所示的值會使學習變得更容易。

- Application Name 此處填寫想呈現給用戶的應用名稱，此處我們使用“My First App”。
- Company domain 包名限定符，Android Studio會將這個限定符應用於每個新建的Android項目
- Package Name是應用的包命名空間（同Java的包的概念），該包名在同一Android系統上所有已安裝的應用中具有唯一性，我們可以獨立地編輯該包名。
- Project location操作系統存放項目的目錄

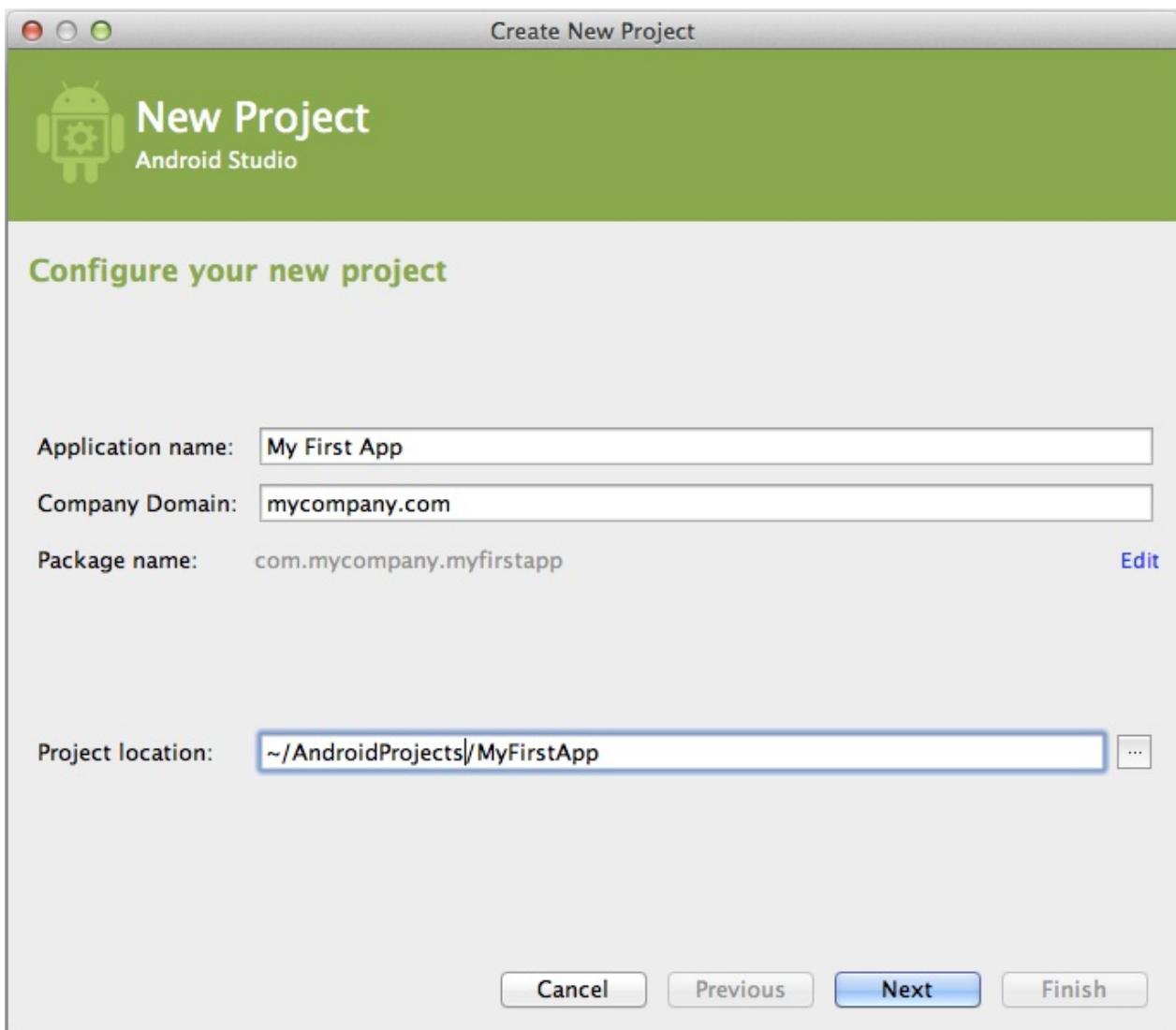


圖1 Configure your new project

3. 在Select the form factors your app will run on窗口勾選Phone and Tablet。

4. Minimum SDK, 選擇 API 8: Android 2.2 (Froyo).

Minimum Required SDK表示我們的應用支持的最低Android版本，為了支持儘可能多的設備，我們應該設置為能支持你應用核心功能的最低API版本。如果某些非核心功能僅在較高版本的API支持，你可以只在支持這些功能的版本上開啓它們(參考兼容不同的系統版本)，此處採用默認值即可。

5. 不要勾選其他選項 (TV, Wear, and Glass)，點擊 Next.

6. 在Add an activity to 窗口選擇Blank Activity，點擊 Next.

7. 在Choose options for your new file 窗口修改Activity Name 為MyActivity，修改 Layout Name 為activity\_my，Title 修改為MyActivity，Menu Resource Name 修改為menu\_my。

8. 點擊Finish完成創建。

剛創建的Android項目是一個基礎的Hello World項目，包含一些默認文件，我們花一點時間看看最重要的部分：

app/src/main/res/layout/activity\_my.xml

這是剛才用Android Studio創建項目時新建的Activity對應的xml佈局文件，按照創建新項目的流程，Android Studio會同時展示這個文件的文本視圖和圖形化預覽視圖，該文件包含一些默認設置和一個顯示內容為“Hello world!”的TextView元素。

app/src/main/java/com.mycompany.myfirstapp/MyActivity.java

用Android Studio創建新項目完成後，可在Android Studio看到該文件對應的選項卡，選中該選項卡，可以看到剛創建的Activity類的定義。編譯並運行該項目後，Activity啓動並加載佈局文件activity\_my.xml，顯示一條文本：“Hello world!”

app/src/main/AndroidManifest.xml

manifest文件描述了項目的基本特徵並列出了組成應用的各個組件，接下來的學習會更深入瞭解這個文件並添加更多組件到該文件中。

app/build.gradle

Android Studio使用Gradle編譯運行Android工程。工程的每個模塊以及整個工程都有一個build.gradle文件。通常你只需要關注模塊的build.gradle文件，該文件存放編譯依賴設置，包括defaultConfig設置：

- compiledSdkVersion 是我們的應用將要編譯的目標Android版本，此處默認為你的SDK已安裝的最新Android版本（目前應該是4.1或更高版本，如果你沒有安裝一個可用Android版本，就要先用SDK Manager來完成安裝），我們仍然可以使用較老的版本編譯項目，但把該值設為最新版本，可以使用Android的最新特性，同時可以在最新的設備上優化應用來提高用戶體驗。
- applicationId 創建新項目時指定的包名。
- minSdkVersion 創建項目時指定的最低SDK版本，是新建應用支持的最低SDK版本。
- targetSdkVersion 表示你測試過你的應用支持的最高Android版本（同樣用API level表示）。當Android發佈最新版本後，我們應該在最新版本的Android測試自己的應用同時更新target sdk到Android最新版本，以便充分利用Android新版本的特性。更多知識，請閱讀[Supporting Different Platform Versions](#)。

更多關於Gradle的知識請閱讀[Building Your Project with Gradle](#)

注意/res目錄下也包含了resources資源：

drawable/

存放各種densities圖像的文件夾，mdpi，hdpi等，這裏能夠找到應用運行時的圖標文件ic\_launcher.png

layout/

存放用戶界面文件，如前邊提到的activity\_my.xml，描述了MyActivity對應的用戶界面。

menu/

存放應用裏定義菜單項的文件。

values/

存放其他xml資源文件，如string，color定義。string.xml定義了運行應用時顯示的文本“Hello world!”

要運行這個APP，繼續[下個小節](#)的學習。

# 使用命令行創建項目

如果沒有使用Android Studio開發Android項目，我們可以在命令行使用SDK提供的tools來創建一個Android項目。

1. 打開命令行切換到SDK根目錄下；

2. 執行：

```
tools/android list targets
```

會在屏幕上打印出我們所有的Android SDK中下載好的可用Android platforms，找想要創建項目的目標platform，記錄該platform對應的Id，推薦使用最新的platform。我們仍可以使自己的應用支持較老版本的platform，但設置為最新版本允許我們為最新的Android設備優化我們的應用。如果沒有看到任何可用的platform，我們需要使用Android SDK Manager完成下載安裝，參見 [Adding Platforms and Packages](#)。

3. 執行：

```
android create project --target <target-id> --name MyFirstApp \
--path <path-to-workspace>/MyFirstApp --activity MyActivity \
--package com.example.myfirstapp
```

替換 `<target-id>` 為上一步記錄好的Id，替換 `<path-to-workspace>` 為我們想要保存項目的路徑。

Tip:把 `platform-tools/` 和 `tools/` 添加到環境變量 `PATH`，開發更方便。

到此為止，我們的Android項目已經是一個基本的“Hello World”程序，包含了一些默認的文件。要運行它，繼續[下個小節](#)的學習。

# 執行Android程序

編寫:yuanfentian789 - 原文:<http://developer.android.com/training/basics/firstapp/running-app.html>

通過上一節課創建了一個Android的Hello World項目，項目默認包含一系列源文件，它讓我們可以立即運行應用程序。

如何運行Android應用取決於兩件事情：是否有一個Android設備和是否正在使用Android Studio開發程序。本節課將會教使用Android Studio和命令行兩種方式在真實的android設備或者android模擬器上安裝並且運行應用。

## 在真實設備上運行

如果有一個真實的Android設備，以下的步驟可以使我們在自己的設備上安裝和運行應用程序：

### 手機設置

1. 把設備用USB線連接到計算機上。如果是在windows系統上進行開發的，你可能還需要安裝你設備對應的USB驅動，詳見[OEM USB Drivers](#) 文檔。
2. 開啓設備上的USB調試選項。
  - 在大部分運行Andriod3.2或更老版本系統的設備上，這個選項位於“設置>應用程序>開發選項”裏。
  - 在Andriod 4.0或更新版本中，這個選項在“設置>開發人員選項”裏。

Note: 從Android4.2開始，開發人員選項在默認情況下是隱藏的，想讓它可見，可以去設置>關於手機（或者關於設備）點擊版本號七次。再返回就能找到開發人員選項了。

### 從Android Studio運行程序

1. 選擇項目的一個文件，點擊工具欄裏的Run  按鈕。
2. Choose Device窗口出現時，選擇Choose a running device單選框，點擊OK。

Android Studio 會把應用程序安裝到我們的設備中並啓動應用程序。

### 從命令行安裝運行應用程序

打開命令行並切換當前目錄到Andriod項目的根目錄，在debug模式下使用Gradle編譯項目，使用gradle腳本執行assembleDebug編譯項目，執行後會在build/目錄下生成MyFirstApp-debug.apk。

Windows操作系統下，執行：

```
gradlew.bat assembleDebug
```

Mac OS或Linux系統下：

```
$ chmod +x gradlew  
$ ./gradlew assembleDebug
```

編譯完成後在app/build/outputs/apk/目錄生成apk。

Note: chmod命令是給gradlew增加執行權限，只需要執行一次。

確保Android SDK裏的platform-tools/路徑已經添加到環境變量PATH中，執行：

```
adb install bin/MyFirstApp-debug.apk
```

在我們的Android設備中找到MyFirstActivity，點擊打開。

## 在模擬器上運行

無論是使用Android Studio還是命令行，在模擬器中運行程序首先要創建一個Android Virtual Device (AVD)。AVD是對Android模擬器的配置，可以讓我們模擬不同的設備。

### 創建一個AVD:

1. 啓動Android Virtual Device Manager (AVD Manager)的兩種方式：

- 用Android Studio, Tools > Android > AVD Manager,或者點擊工具欄裏面Android Virtual Device Manager



；

- 在命令行窗口中，把當前目錄切換到<sdk>/tools/後執行：

```
android avd
```



2 在AVD Manager面板中，點擊Create Virtual Device。

3 在Select Hardware窗口，選擇一個設備，比如Nexus 6，點擊Next。

4 選擇列出的合適系統鏡像。

5 校驗模擬器配置，點擊Finish。

更多AVD的知識請閱讀[Managing AVDs with AVD Manager](#).

### 從Android Studio運行程序：

1 在Android Studio選擇要運行的項目，從工具欄選擇Run ；

2 Choose Device窗口出現時，選擇Launch emulator單選框；

3 從 Android virtual device下拉菜單選擇創建好的模擬器，點擊OK；

模擬器啓動需要幾分鐘的時間，啓動完成後，解鎖即可看到程序已經運行到模擬器屏幕上。

## 從命令行安裝運行應用程序

1 用命令行編譯應用，生成位於app/build/outputs/apk/的apk；

2 確認platform-tools/ 已添加到PATH環境變量；

3 執行如下命令：

```
adb install app/build/outputs/MyFirstApp-debug.apk
```

4 在模擬器上找到MyFirstApp，並運行。

以上就是創建並在設備上運行一個應用的全部過程！想要開始開發，點擊[next lesson](#)。

# 建立簡單的用戶界面

編寫:yuanfentiank789 - 原文:<http://developer.android.com/training/basics/firstapp/building-ui.html>

在本小節裏，我們將學習如何用XML創建一個帶有文本輸入框和按鈕的界面，下一節課將學會使app對按鈕做出響應：按鈕被按下時，文本框裏的內容被發送

Android的圖形用戶界面是由多個View和ViewGroup構建出來的。View是通用的UI窗體小組件，比如按鈕(Button)或者文本框(text field)，而ViewGroup是不可見的，是用於定義子View佈局方式的容器，比如網格部件(grid)和垂直列表部件(list)。

Android提供了一個對應於View和ViewGroup子類的一系列XML標籤，我們可以在XML裏使用層級視圖元素創建自己的UI。

Layouts是ViewGroup的子類，接下來的練習將使用LinearLayout。

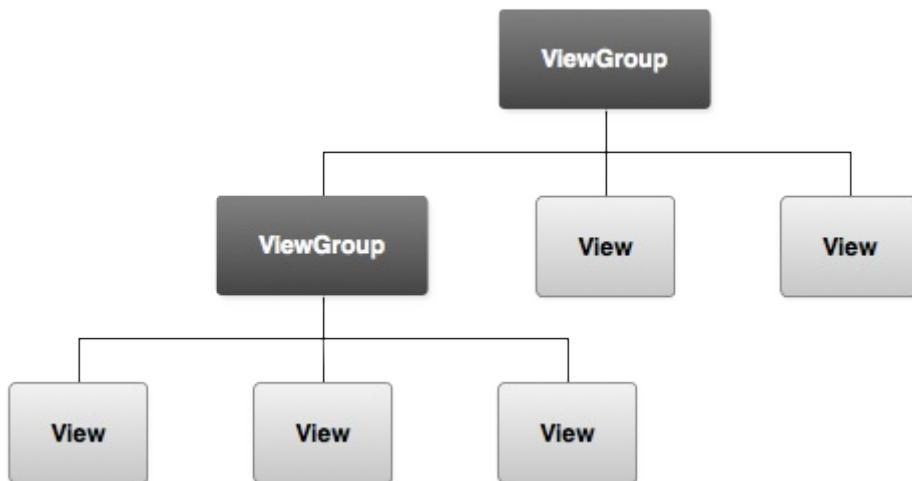


Figure 1. 關於viewgroup對象如何組織佈局分支和包含其他view對象。

可選的佈局文件：在XML中定義界面佈局而不是在運行時去動態生成佈局是有多個原因的，其中最重要的一點是這樣可以使得你為不同大小的屏幕創建不同的佈局文件。例如，你可以創建創建2個版本的佈局文件，告訴系統在小的屏幕上使用其中一個佈局文件，在大的屏幕上使用另外一個佈局文件。更多信息，請參考[兼容不同的設備](#)

## 創建一個LinearLayout

1 在Android Studio中，從res/layout目錄打開activity\_my.xml文件。上一節創建新項目時生成的BlankActivity，包含一個activity\_my.xml文件，該文件根元素是一個包含TextView的RelativeLayout。

2 在Preview面板點擊 關閉右側Preview面板，在Android Studio中，當打開佈局文件的時，可以看到一個Preview面板，點擊這個面板中的標籤，可利用WYSIWYG（所見即所得）工具在Design面板看到對應的圖形化效果，但在本節直接操作XML文件即可。

3 刪除 TextView 標籤。

4 把 RelativeLayout 標籤改為 LinearLayout.

5 為添加 android:orientation 屬性並設置值為 "horizontal".

6 去掉 android:padding 屬性和 tools:context 屬性.

修改後結果如下：

res/layout/activity\_my.xml

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

LinearLayout是ViewGroup的一個子類，用於放置水平或者垂直方向的子視圖部件，放置方向由屬性 android:orientation 設定。LinearLayout裏的子佈局按照XML裏定義的順序顯示在屏幕上。

所有的Views都需要用到 android:layout\_width 和 android:layout\_height 這兩個屬性來設置自身的大小。

由於LinearLayout是整個視圖的根佈局，所以其寬和高都應充滿整個屏幕的，通過指定 width 和 height 屬性為 "match\_parent"。該值表示子View擴張自己 width 和 height 來匹配父控件的 width 和 height。

更多關於 Layout 屬性的信息，請參照 XML 佈局嚮導。

## 添加一個文本輸入框

與其它View一樣，我們需要設置XML裏的某些屬性來指定EditText的屬性值，以下是應該在線性布局裏指定的一些屬性元素：

1 在 activity\_my.xml 文件的 標籤內定義一個 標籤，並設置 id 屬性為 @+id/edit\_message.

2 設置 layout\_width 和 layout\_height 屬性為 wrap\_content.

3 設置 hint 屬性為一個 string 值的引用 edit\_message.

代碼如下：

res/layout/activity\_my.xml

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

各屬性說明：

### android:id

這是定義 View 的唯一標識符。可以在程序代碼中通過該標識符對對象進行引用，例如對這個對象進行讀和修改的操作（在一課裏將會用到）。

當想從XML裏引用資源對象的時候必須使用@符號。緊隨@之後的是資源的類型(這裏是 `id`)，然後是資源的名字(這裏使用的是 `edit_message`)。

+號只是當你第一次定義一個資源ID的時候需要。這裏是告訴SDK此資源ID需要被創建出來。在應用程序被編譯之後，SDK就可以直接使用ID值，`edit_message`是在項目 `gen/R.java` 文件中創建一個新的標識符，這個標識符就和`EditText`關聯起來了。一旦資源ID被創建了，其他資源如果引用這個ID就不再需要+號了。這裏是唯一一個需要+號的屬性。

## android:layout\_width 和 android:layout\_height

對於寬和高不建議指定具體的大小，使用 `wrap_content` 指定之後，這個視圖將只佔據內容大小的空間。如果你使用了 `match_parent`，這時`EditText`將會佈滿整個屏幕，因為它將適應父佈局的大小。更多信息，請參考 [佈局嚮導](#)。

## android:hint

當文本框為空的時候，會默認顯示這個字符串。對於字符串 `@string/edit_message` 的值所引用的資源應該是定義在單獨的文件裏，而不是直接使用字符串。因為使用的值是存在的資源，所以不需要使用+號。然而，由於你還沒有定義字符串的值，所以在添加 `@string/edit_message` 時候會出現編譯錯誤。下邊你可以定義字符串資源值來去除這個錯誤。

Note: 該字符串資源與id使用了相同的名稱（`edit_message`）。然而，對於資源的引用是區分類型的（比如id和字符串），因此，使用相同的名稱不會引起衝突。

## 增加字符串資源

默認情況下，你的Android項目包含一個字符串資源文件，`res/values/string.xml`。打開這個文件，為 "edit\_message" 增加一個供使用的字符串定義，設置值為"Enter a message."

1 在Android Studio裏，編輯 `res/values` 下的 `strings.xml` 文件.

2 添加一個string名為"edit\_message" ,值為 "Enter a message".

3 再添加一個string名為 "button\_send"，值為"Send".下面的內容將使用這個string來創建一個按鈕.

4 刪除 "hello world" string這一行.

下邊就是修改好的 `res/values/strings.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
    <string name="action_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

當你在用戶界面定義一個文本的時候，你應該把每一個文本字符串列入資源文件。這樣做的好處是：對於所有字符串值，字符串資源能夠單獨的修改，在資源文件裏你可以很容易的找到並且做出相應的修改。通過選擇定義每個字符串，還允許您對不同語言本地化應用程序。

更多的於不同語言本字符串資源本地化的問題，請參考[兼容不同的設備\(Supporting Different Devices\)](#)。

# 添加一個按鈕

- 1 在 Android Studio 裏, 編輯 res/layout 下的 activity\_my.xml 文件.
- 2 在 LinearLayout 內部, 在標籤之後定義一個標籤.
- 3 設置 Button 的 width 和 height 屬性值為 "wrap\_content" 以便讓 Button 大小能完整顯示其上的文本.
- 4 定義 button 的文本使用 android:text 屬性, 設置其值為之前定義好的 button\_send 字符串.

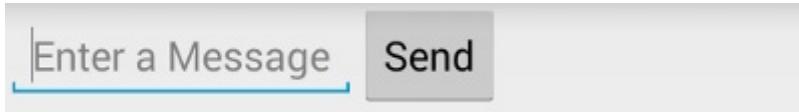
此時的 LinearLayout 看起來應該是這樣

res/layout/activity\_my.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

Note 寬和高被設置為 "wrap\_content"，這時按鈕佔據的大小就是按鈕裏文本的大小。這個按鈕不需要指定 android:id 的屬性，因為 Activity 代碼中不會引用該 Button。

當前 EditText 和 Button 部件只是適應了他們各自內容的大小，如下圖所示：



這樣設置對按鈕來說很合適，但是對於文本框來說就不太好，因為用戶可能輸入更長的文本內容。因此如果能夠佔滿整個屏幕寬度會更好。LinearLayout 使用權重屬性來達到這個目的，你可以使用 android:layout\_weight 屬性來設置。

權重的值指的是每個部件所佔剩餘空間的大小，該值與同級部件所佔空間大小有關。就類似於飲料的成分配方：“兩份伏特加酒，一份咖啡利口酒”，即該酒中伏特加酒佔三分之二。例如，我們設置一個 View 的權重是 2，另一個 View 的權重是 1，那麼總數就是 3，這時第一個 View 佔據 2/3 的空間，第二個佔據 1/3 的空間。如果你再加入第三個 View，權重設為 1，那麼第一個 View(權重為 2 的)會佔據 1/2 的空間，剩餘的另外兩個 View 各佔 1/4。(請注意，使用權重的前提一般是給 View 的寬或者高的大小設置為 0dp，然後系統根據上面的權重規則來計算 View 應該佔據的空間。但是很多情況下，如果給 View 設置了 match\_parent 的屬性，那麼上面計算權重時則不是通常的正比，而是反比，也就是權重值大的反而佔據空間小)。

對於所有的 View 默認的權重是 0，如果只設置了一個 View 的權重大於 0，則該 View 將佔據除去別的 View 本身佔據的空間的所有剩餘空間。因此這裏設置 EditText 的權重為 1，使其能夠佔據除了按鈕之外的所有空間。

## 讓輸入框充滿整個屏幕的寬度

為讓 EditText充滿剩餘空間，做如下操作：

1 在 activity\_my.xml 文件裏，設置EditText的layout\_weight屬性值為1。

2 設置EditText的layout\_width值為0dp。

res/layout/activity\_my.xml

```
<EditText  
    android:layout_weight="1"  
    android:layout_width="0dp"  
    ... />
```

為了提升佈局的效率，在設置權重的時候，應該把EditText的寬度設為0dp。如果設置"wrap\_content"作為寬度，系統需要自己去計算這個部件所佔有的寬度，而此時的因為設置了權重，所以系統自動會佔據剩餘空間，EditText的寬度最終成了不起作用的屬性。

設置權重後的效果圖



現在看一下完整的佈局文件內容：

res/layout/activity\_my.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="horizontal">  
    <EditText android:id="@+id/edit_message"  
        android:layout_weight="1"  
        android:layout_width="0dp"  
        android:layout_height="wrap_content"  
        android:hint="@string/edit_message" />  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/button_send" />  
</LinearLayout>
```

## 運行應用

整個佈局默認被應用於創建項目的時候SDK工具自動生成的Activity，運行看下效果：

- 在Android Studio裏，點擊工具欄裏的Run按鈕 
- 或者使用命令行，進入你項目的根目錄直接執行

```
ant debug  
adb install bin/MyFirstApp-debug.apk
```

下一小節將學習有關如何對按鈕做出相應，同時讀取文本中的內容，啓動另外一個Activity等。

下一節：啓動另一個Activity

# 啓動另一個Activity

編寫:yuanfentiank789 - 原文:<http://developer.android.com/training/basics/firstapp/starting-activity.html>

在完成上一課(建立簡單的用戶界面)後，我們已經擁有了顯示一個activity (一個界面)的app (應用)，該activity包含了一個文本字段和一個按鈕。在這節課中，我們將添加一些新的代碼到 MyActivity 中，當用戶點擊發送(Send)按鈕時啓動一個新的activity 。

## 響應Send(發送)按鈕

1 在Android Studio中打開res/layout目錄下的activity\_my.xml 文件.

2 為 Button 標籤添加android:onClick屬性.

res/layout/activity\_my.xml

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

android:onClick 屬性的值 "sendMessage" 即為用戶點擊屏幕按鈕時觸發方法的名字。

3 打開java/com.mycompany.myfirstapp目錄下MyActivity.java 文件.

4 在MyActivity.java 中添加sendMessage() 函數：

java/com.mycompany.myfirstapp/MyActivity.java

```
/** Called when the user clicks the Send button */  
public void sendMessage(View view) {  
    // Do something in response to button  
}
```

爲使系統能夠將該方法 (你剛在MyActivity.java中添加的sendMessage方法) 與在 android:onClick 屬性中提供的方法名字匹配，它們的名字必須一致，特別需要注意的是，這個方法必須滿足以下條件：

- 是public函數
- 無返回值
- 參數唯一(爲View類型,代表被點擊的視圖)

接下來，你可以在這個方法中編寫讀取文本內容，並將該內容傳到另一個Activity的代碼。

## 構建一個Intent

Intent是在不同組件中(比如兩個Activity)提供運行時綁定的對象。 Intent 代表一個應用"想去做什麼事"，你可以用它做各種各樣的任務，不過大部分的時候他們被用來啓動另一個Activity。更詳細的內容可以參考Intents and Intent Filters。

1 在`MyActivity.java`的 `sendMessage()` 方法中創建一個 `Intent` 並啓動名為 `DisplayMessageActivity` 的`Activity`：

`java/com.mycompany.myfirstapp/MyActivity.java`

```
Intent intent = new Intent(this, DisplayMessageActivity.class);
```

Note：如果使用的是類似Android Studio的IDE，這裏對 `DisplayMessageActivity` 的引用會報錯，因為這個類還不存在；暫時先忽略這個錯誤，我們很快就要去創建這個類了。

在這個`Intent`構造函數中有兩個參數：

- 第一個參數是`Context`(之所以用 `this` 是因為當前`Activity`是 `Context` 的子類)
- 接受系統發送`Intent`的應用組件的`Class` (在這個案例中，指將要被啓動的`activity`) 。

Android Studio會提示導入`Intent`類。

2 在文件開始處導入`Intent`類:

`java/com.mycompany.myfirstapp/MyActivity.java`

```
import android.content.Intent;
```

Tip:在Android Studio中，按Alt + Enter 可以導入缺失的類(在Mac中使用option + return)

3 在 `sendMessage()` 方法裏用`findViewById()`方法得到`EditText`元素.

`java/com.mycompany.myfirstapp/MyActivity.java`

```
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
}
```

4 在文件開始處導入`EditText`類.

在Android Studio中，按Alt + Enter 可以導入缺失的類(在Mac中使用option + return)

5 把`EditText`的文本內容關聯到一個本地 `message` 變量，並使用`putExtra()`方法把值傳給`intent`.

`java/com.mycompany.myfirstapp/MyActivity.java`

```
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
    String message = editText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
}
```

`Intent`可以攜帶稱作 `extras` 的鍵-值對數據類型。`putExtra()`方法把鍵名作為第一個參數，把值作為第二個參數。

6 在MyActivity class, 定義EXTRA\_MESSAGE :

java/com.mycompany.myfirstapp/MyActivity.java

```
public class MyActivity extends ActionBarActivity {  
    public final static String EXTRA_MESSAGE = "com.mycompany.myfirstapp.MESSAGE";  
    ...  
}
```

爲讓新啓動的activity能查詢extra數據。定義key爲一個public型的常量，通常使用應用程序包名作爲前綴來定義鍵是很好的做法，這樣在應用程序與其他應用程序進行交互時仍可以確保鍵是唯一的。

7 在sendMessage()函數裏，調用startActivity()完成新activity的啓動，現在完整的代碼應該是下面這個樣子：

java/com.mycompany.myfirstapp/MyActivity.java

```
/** Called when the user clicks the Send button */  
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
    String message = editText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
    startActivity(intent);  
}
```

運行這個方法，系統收到我們的請求後會實例化在 Intent 中指定的 Activity，現在需要創建一個 DisplayMessageActivity 類使程序能夠執行起來。

## 創建第二個Activity

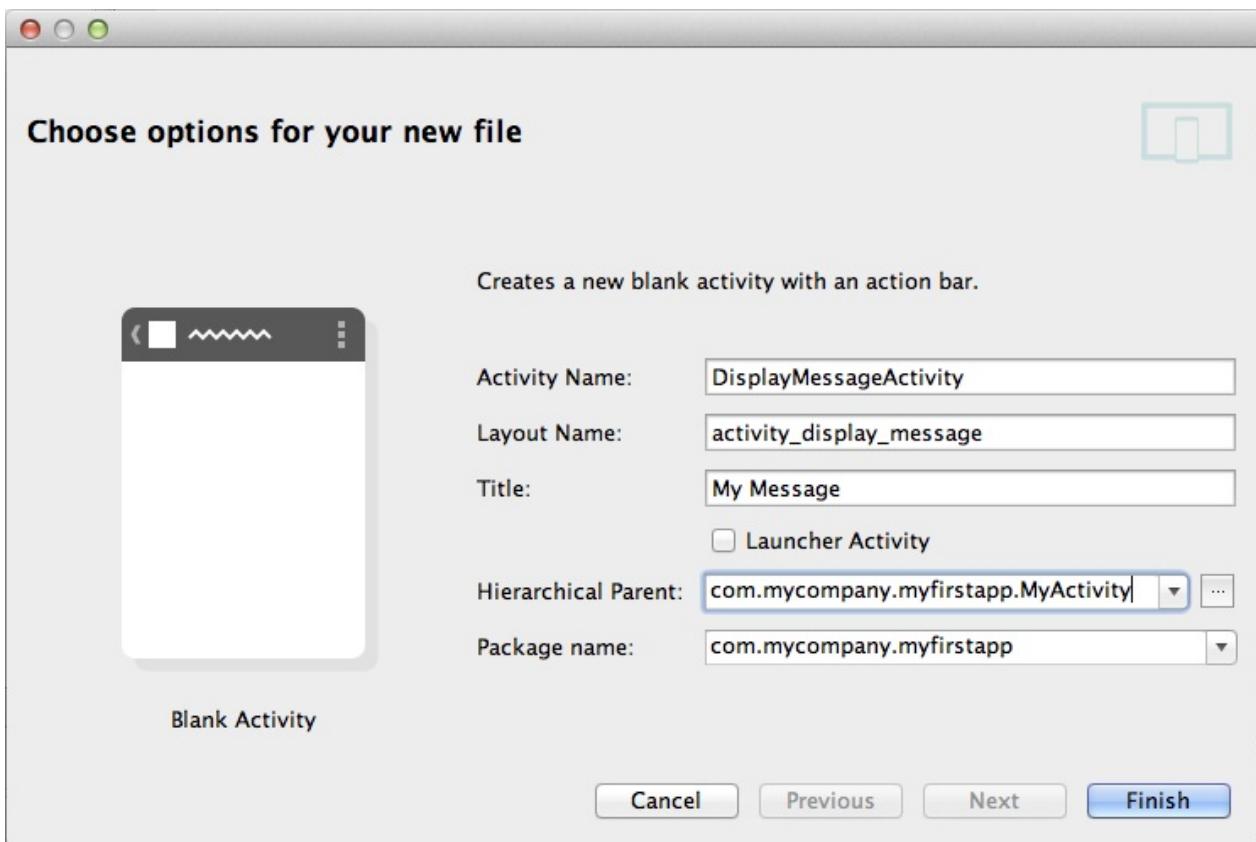
Activity所有子類都必須實現onCreate()方法。創建activity的實例時系統會調用該方式，此時必須用setContentView()來定義Activity佈局，以對Activity進行初始化。

### 使用Android Studio創建新的Activity

使用Android Studio創建的activity會實現一個默認的onCreate()方法。

1. 在Android Studio的java 目錄，選擇包名 com.mycompany.myfirstapp,右鍵選擇 New > Activity > Blank Activity.
2. 在Choose options窗口，配置activity：
  - Activity Name: DisplayMessageActivity
  - Layout Name: activity\_display\_message
  - Title: My Message
  - Hierarchical Parent: com.mycompany.myfirstapp.MyActivity

Package name: com.mycompany.myfirstapp 點擊 Finish.



3 打開DisplayMessageActivity.java文件，此類已經實現了onCreate()方法，稍後需要更新此方法。另外還有一個onOptionsItemSelected()方法，用來處理action bar的點擊行爲。暫時保留這兩個方法不變。

4 由於這個應用程序並不需要onCreateOptionsMenu()，直接刪除這個方法。

如果使用Android Studio開發，現在已經可以點擊Send按鈕啓動這個activity了，但顯示的仍然是模板提供的默認內容"Hello world"，稍後修改顯示自定義的文本內容。

## 使用命令行創建activity

如果使用命令行工具創建activity，按如下步驟操作：

1 在工程的src/目錄下，緊挨着MyActivity.java創建一個新文件DisplayMessageActivity.java.

2 寫入如下代碼：

```
public class DisplayMessageActivity extends ActionBarActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_display_message);  
  
        if (savedInstanceState == null) {  
            getSupportFragmentManager().beginTransaction()  
                .add(R.id.container, new PlaceholderFragment()).commit();  
        }  
    }  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        // Handle action bar item clicks here. The action bar will  
        // automatically handle clicks on the Home/Up button, so long  
        // as you specify a parent activity in AndroidManifest.xml.  
    }  
}
```

```

        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    /**
     * A placeholder fragment containing a simple view.
     */
    public static class PlaceholderFragment extends Fragment {

        public PlaceholderFragment() { }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                               Bundle savedInstanceState) {
            View rootView = inflater.inflate(R.layout.fragment_display_message,
                                             container, false);
            return rootView;
        }
    }
}

```

Note:如果使用的IDE不是 Android Studio，工程中可能不會包含由 `setContentView()` 請求的 `activity_display_message` layout，但這沒關係，因為等下會修改這個方法。

3 把新Activity的標題添加到strings.xml文件:

```

<resources>
    ...
    <string name="title_activity_display_message">My Message</string>
</resources>

```

4 在 AndroidManifest.xml的Application 標籤內為 DisplayMessageActivity添加 標籤，如下:

```

<application ... >
    ...
    <activity
        android:name="com.mycompany.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.mycompany.myfirstapp.MyActivity" >
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.mycompany.myfirstapp.MyActivity" />
    </activity>
</application>

```

`android:parentActivityName` 屬性聲明瞭在應用程序中該Activity邏輯層面的父類Activity的名稱。系統使用此值來實現默認導航操作，比如在Android 4.1 ( API level 16 ) 或者更高版本中的Up navigation。使用Support Library，如上所示的 `<meta-data>` 元素可以為安卓舊版本提供相同功能。

Note:我們的Android SDK應該已經包含了最新的Android Support Library，它包含在ADT插件中。但如果用的是別的IDE，則需要在 [Adding Platforms and Packages](#) 中安裝。當Android Studio中使用模板時，Support Library會自動加入我們的工程中(在Android Dependencies中你以看到相應的JAR文件)。如果不使用Android Studio，就需要手動將Support Library添加到我們的工程中，參考[setting up the Support Library](#)。

## 接收Intent

不管用戶導航到哪，每個Activity都是通過Intent被調用的。我們可以通過調用getIntent()來獲取啓動activity的Intent及其包含的數據。

1 編輯java/com.mycompany.myfirstapp目錄下的DisplayMessageActivity.java文件.

2 刪除onCreate()方法中下面一行:

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

3 得到intent 並賦值給本地變量.

```
Intent intent = getIntent();
```

4 為Intent導入包.

在Android Studio中，按Alt + Enter 可以導入缺失的類(在Mac中使用option + return).

5 調用 getStringExtra()提取從 MyActivity 傳遞過來的消息.

```
String message = intent.getStringExtra(MyActivity.EXTRA_MESSAGE);
```

## 顯示文本

1 在onCreate() 方法中，創建一個 TextView 對象.

```
TextView textView = new TextView(this);
```

2 設置文本字體大小和內容.

```
textView.setTextSize(40);
textView.setText(message);
```

3 通過調用activity的setContentView()把TextView作為activity佈局的根視圖.

```
setContentView(textView);
```

4 為TextView 導入包.

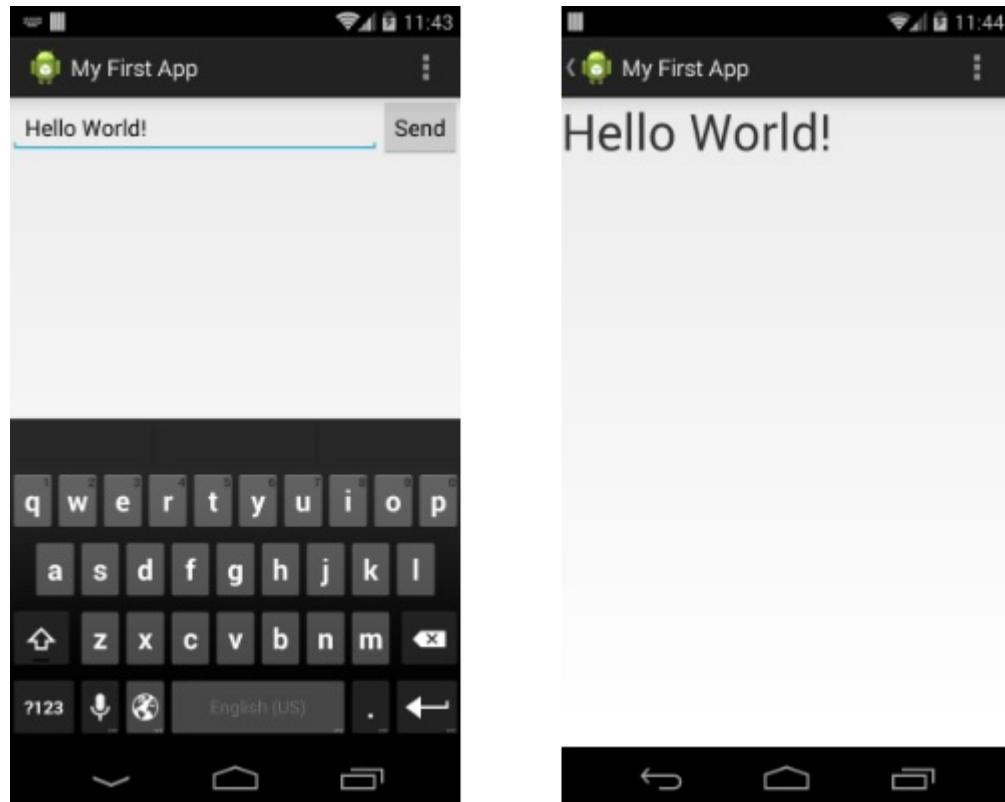
在Android Studio中，按Alt + Enter 可以導入缺失的類(在Mac中使用option + return).

DisplayMessageActivity的完整onCreate()方法應該如下：

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

```
// Get the message from the intent  
Intent intent = getIntent();  
String message = intent.getStringExtra(MyActivity.EXTRA_MESSAGE);  
  
// Create the text view  
TextView textView = new TextView(this);  
textView.setTextSize(40);  
textView.setText(message);  
  
// Set the text view as the activity layout  
setContentView(textView);  
}
```

現在你可以運行app，在文本中輸入信息，點擊Send(發送)按鈕，ok，現在就可以在第二Activity上看到發送過來信息了。如圖：



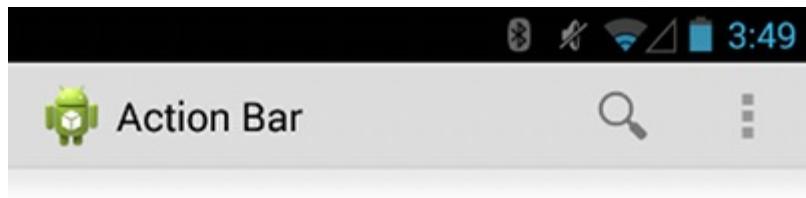
到此為止，已經創建好我們的第一個Android應用！

# 添加Action Bar

編寫:Vincent 4J - 原文:<http://developer.android.com/training/basics/actionbar/index.html>

Action Bar是我們可以為activity實現的最重要的設計元素之一。其提供了多種UI特性，可以讓我們的app與其他Android app保持較高的一致性，從而為用戶所熟悉。核心的功能包括：

- 一個專門的空間用來顯示你的app的標識，以及指出目前所處在app的哪個頁面。
- 以一種可預見的方式訪問重要的操作（比如搜索）。
- 支持導航和視圖切換（通過Tabs和下拉列表）



本章為action bar的基本知識提供了一個快速指南。關於action bar的更多特性，請查看 [Action Bar指南](#)。

## Lessons

- [建立ActionBar](#)

學習如何為activity添加一個基本的action bar，是僅僅支持Android 3.0及以上的版本，還是同時也支持至Android 2.1的版本（通過使用Andriod Support Library）。

- [添加Action按钮](#)

學習如何在action bar中添加和響應用戶操作。

- [ActionBar的風格化](#)

學習如何自定義action bar的外觀。

- [ActionBar覆蓋疊加](#)

學習如何在佈局上面疊加action bar，允許action bar隱藏時無縫過渡。

# 建立ActionBar

編寫:Vincent 4J - 原文:<http://developer.android.com/training/basics/actionbar/setting-up.html>

Action bar 最基本的形式，就是為 `Activity` 顯示標題，並且在標題左邊顯示一個 app icon。即使在這樣簡單的形式下，action bar對於所有的 `activity` 來說是十分有用的。它告知用戶他們當前所處的位置，併為你的 app 維護了持續的同一標識。

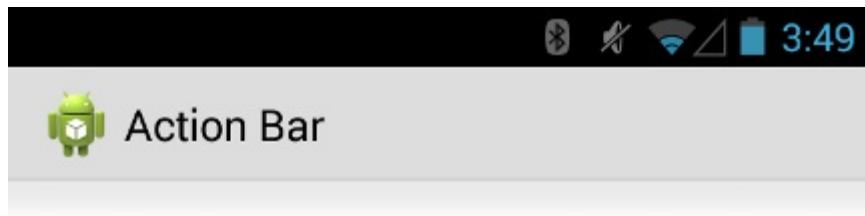


圖 1. 一個有 app icon 和 `Activity` 標題的 action bar

設置一個基本的 action bar，需要 app 使用一個 `activity` 主題，該主題必須是 action bar 可用的。如何聲明這樣的主題取決於我們 app 支持的 Android 最低版本。本課程根據我們 app 支持的 Android 最低版本分為兩部分。

## 僅支持 Android 3.0 及以上版本

從 Android 3.0(API lever 11) 開始，所有使用 `Theme.Holo` 主題（或者它的子類）的 `Activity` 都包含了 action bar，當 `targetSdkVersion` 或 `minSdkVersion` 屬性被設置成 “11” 或更大時，它是默認主題。

所以，要為 `activity` 添加 action bar，只需簡單地設置屬性為 `11` 或者更大。例如：

```
<manifest ... >
    <uses-sdk android:minSdkVersion="11" ... />
    ...
</manifest>
```

注意: 如果創建了一個自定義主題，需確保這個主題使用一個 `Theme.Holo` 的主題作為父類。詳情見 [Action bar 的風格化](#)

到此，我們的 app 使用了 `Theme.Holo` 主題，並且所有的 `activity` 都顯示 action bar。

## 支持 Android 2.1 及以上版本

當 app 運行在 Andriod 3.0 以下版本（不低於 Android 2.1）時，如果要添加 action bar，需要加載 Android Support 庫。

開始之前，通過閱讀[Support Library Setup](#)文檔來建立v7 appcompat library（下載完library包之後，按照[Adding libraries with resources](#)的指引進行操作）。

在 Support Library集成到你的 app 工程中之後：

1、更新 `activity`，以便於它繼承於 `ActionBarActivity`。例如：

```
public class MainActivity extends ActionBarActivity { ... }
```

2、在 manifest 文件中，更新 `<application>` 標籤或者單一的 `<activity>` 標籤來使用一個 `Theme.AppCompat` 主題。例如：

```
<activity android:theme="@style/Theme.AppCompat.Light" ... >
```

注意：如果創建一個自定義主題，需確保其使用一個 `Theme.AppCompat` 主題作為父類。詳情見 [Action bar 風格化](#)

現在，當 app 運行在 Android 2.1(API level 7) 或者以上時，`activity` 將包含 action bar。

切記，在 manifest 中正確地設置 app 支持的 API level：

```
<manifest ... >
  <uses-sdk android:minSdkVersion="7"  android:targetSdkVersion="18" />
  ...
</manifest>
```

# 添加Action按鈕

編寫:Vincent 4J - 原文:<http://developer.android.com/training/basics/actionbar/adding-buttons.html>

Action bar 允許我們為當前環境下最重要的操作添加按鈕。那些直接出現在 action bar 中的 icon 和/或文本被稱作action buttons(操作按鈕)。安排不下的或不足夠重要的操作被隱藏在 action overflow (超出空間的action，譯者注)中。



圖 1. 一個有search操作按鈕和 action overflow 的 action bar，在 action overflow 裏能展現額外的操作。

## 在 XML 中指定操作

所有的操作按鈕和 action overflow 中其他可用的條目都被定義在 [menu資源](#) 的 XML 文件中。通過在項目的 `res/menu` 目錄中新增一個 XML 文件來為 action bar 添加操作。

為想要添加到 action bar 中的每個條目添加一個 `<item>` 元素。例如：

`res/menu/main_activity_actions.xml`

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- 搜索，應該作為動作按鈕顯示 -->
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"
          android:showAsAction="ifRoom" />
    <!-- 設置，在溢出菜單中顯示 -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:showAsAction="never" />
</menu>
```

上述代碼聲明，當 action bar 有可用空間時，搜索操作將作為一個操作按鈕來顯示，但設置操作將一直只在 action overflow 中顯示。(默認情況下，所有的操作都顯示在 action overflow 中，但為每一個操作指明設計意圖是很好的做法。)

icon 屬性要求每張圖片提供一個 `resource ID`。在 `@drawable/` 之後的名字必須是存儲在項目目錄 `res/drawable/` 下位圖圖片的文件名。例如：`ic_action_search.png` 對應 `"@drawable/ic_action_search"`。同樣地，title 屬性使用通過 XML 文件定義在項目目錄 `res/values/` 中的一個 `string` 資源，詳情請參見 [創建一個簡單的 UI](#)。

注意：當創建 icon 和其他 bitmap 圖片時，要為不同屏幕密度下的顯示效果提供多個優化的版本，這一點很重要。在 [支持不同屏幕](#) 課程中將會更詳細地討論。

如果為了兼容 Android 2.1 的版本使用了 Support 庫，在 `android` 命名空間下 `showAsAction` 屬性是不可用的。Support 庫會提供替代它的屬性，我們必須聲明自己的 XML 命名空間，並且使用該命名空間作為屬性前綴。(一個自定義 XML 命名空間需要以我們的 app 名稱為基礎，但是可以取任何想要的名稱，它的作用域僅僅在我們聲明的文

件之內。) 例如：

res/menu/main\_activity\_actions.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:yourapp="http://schemas.android.com/apk/res-auto" >
    <!-- 搜索，應該展示為動作按鈕 -->
    <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search"
        yourapp:showAsAction="ifRoom" />
    ...
</menu>
```

## 為 Action Bar 添加操作

要為 action bar 佈局菜單條目，就要在 `activity` 中實現 `onCreateOptionsMenu()` 回調方法來 `inflate` 菜單資源從而獲取 `Menu` 對象。例如：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // 為ActionBar擴展菜單項
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_activity_actions, menu);
    return super.onCreateOptionsMenu(menu);
}
```

## 為操作按鈕添加響應事件

當用戶按下某一個操作按鈕或者 action overflow 中的其他條目，系統將調用 `activity` 中 `onOptionsItemSelected()` 的回調方法。在該方法的實現裏面調用 `MenuItem` 的 `getItemId()` 來判斷哪個條目被按下 - 返回的 ID 會匹配我們聲明對應的 `<item>` 元素中 `android:id` 屬性的值。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // 處理動作按鈕的點擊事件
    switch (item.getItemId()) {
        case R.id.action_search:
            openSearch();
            return true;
        case R.id.action_settings:
            openSettings();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

## 為下級 Activity 添加向上按鈕

在不是程序入口的其他所有屏中（`activity` 不位於主屏時），需要在 action bar 中為用戶提供一個導航到邏輯父屏的 up button(向上按鈕)。



圖 2. Gmail 中的 up button。

當運行在 Android 4.1(API level 16) 或更高版本，或者使用 Support 庫中的 `ActionBarActivity` 時，實現向上導航需要在 manifest 文件中聲明父 `activity`，同時在 action bar 中設置向上按鈕可用。

如何在 manifest 中聲明一個 `activity` 的父類，例如：

```
<application ... >
    ...
    <!-- 主 main/home 活動（沒有上級活動）-->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- 主活動的一個子活動-->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- meta-data 用於支持 support 4.0 以及以下來指明上級活動 -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

然後，通過調用 `setDisplayHomeAsUpEnabled()` 來把 app icon 設置成可用的向上按鈕：

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

    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    // 如果你的minSdkVersion屬性是11或更高，應該這麼用：
    // getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}
```

由於系統已經知道 `MainActivity` 是 `DisplayMessageActivity` 的父 `activity`，當用戶按下向上按鈕時，系統會導航到恰當的父 `activity` - 你不需要去處理向上按鈕的事件。

更多關於向上導航的信息，請見 [提供向上導航](#)。

# 自定義ActionBar的風格

編寫:Vincent 4J - 原文:<http://developer.android.com/training/basics/actionbar/styling.html>

Action bar 為用戶提供一種熟悉可預測的方式來展示操作和導航，但是這並不意味着我們的 app 要看起來和其他 app 一樣。如果想將 action bar 的風格設計的合乎我們產品的定位，只需簡單地使用 Android 的 [樣式和主題](#) 資源。

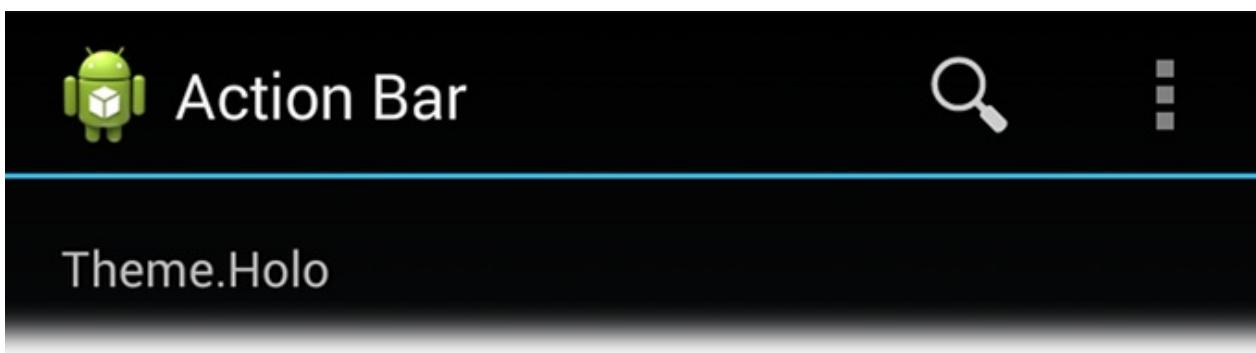
Android 包括一些部分內置的 [activity](#) 主題，這些主題中包含 “dark” 或 “light” 的 action bar 樣式。我們也可以擴展這些主題，以便於更好的為 action bar 自定義外觀。

注意：如果我們為 action bar 使用了 Support 庫的 API，那我們必須使用（或重寫）[Theme.AppCompat](#) 家族樣式（甚至 [Theme.Holo](#) 家族，在 API level 11 或更高版本中可用）。如此一來，聲明的每一個樣式屬性都必須被聲明兩次：一次使用系統平臺的樣式屬性（[android:](#) 屬性），另一次使用 Support 庫中的樣式屬性（[appcompat.R.attr](#) 屬性 - 這些屬性的上下文其實就是我們的 app）。更多細節請查看下面的示例。

## 使用一個 Android 主題

Android 包含兩個基本的 [activity](#) 主題，這兩個主題決定了 action bar 的顏色：

- [Theme.Holo](#)，一個 “dark” 的主題
- [Theme.Holo.Light](#)，一個 “light” 的主題

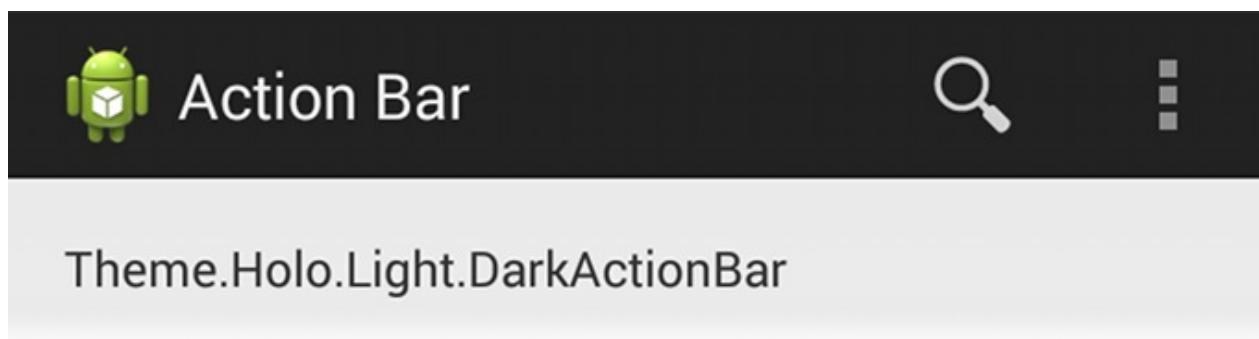


這些主題即可以被應用到 app 全局，也可以通過在 manifest 文件中設置 [application](#) 元素或 [activity](#) 元素的 [android:theme](#) 屬性，對單一的 [activity](#) 進行設置。

例如：

```
<application android:theme="@android:style/Theme.Holo.Light" ... />
```

可以通過聲明 `activity` 的主題為 `Theme.Holo.Light.DarkActionBar` 來達到如下效果：action bar 為 dark，其他部分為 light。



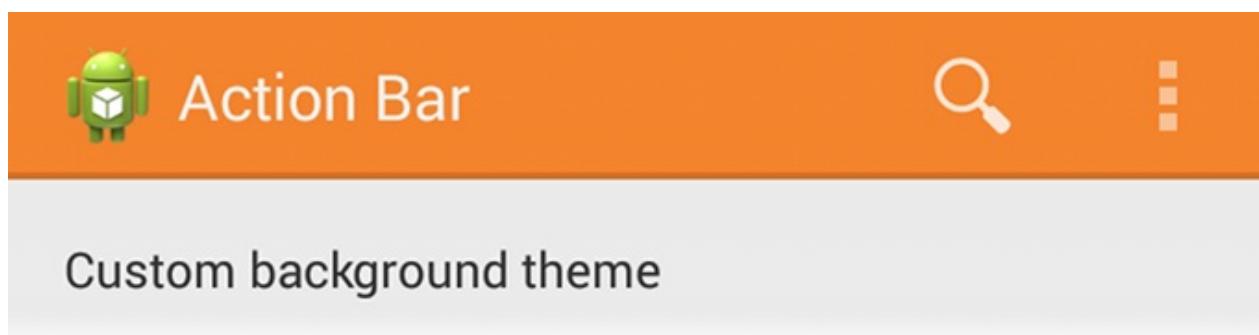
當使用 Support 庫時，必須使用 `Theme.AppCompat` 主題替代：

- `Theme.AppCompat`，一個 “dark” 的主題
- `Theme.AppCompat.Light`，一個 “light” 的主題
- `Theme.AppCompat.Light.DarkActionBar`，一個帶有 “dark” action bar 的 “light” 主題

一定要確保我們使用的 action bar icon 的顏色與 action bar 本身的顏色有差異。`Action Bar Icon Pack` 為 Holo “dark” 和 “light” 的 action bar 提供了標準的 action icon。

## 自定義背景

為改變 action bar 的背景，可以通過為 `activity` 創建一個自定義主題，並重寫 `ActionBarStyle` 屬性來實現。`ActionBarStyle` 屬性指向另一個樣式；在該樣式裏，通過指定一個 `drawable` 資源來重寫 `background` 屬性。



如果我們的 app 使用了 `navigation tabs` 或 `split action bar`，也可以通過分別設置 `backgroundStacked` 和 `backgroundSplit` 屬性來為這些條指定背景。

Note：為自定義主題和樣式聲明一個合適的父主題，這點很重要。如果沒有父樣式，action bar 將會失去很多默認的樣式屬性，除非我們自己顯式的對他們進行聲明。

## 僅支持 Android 3.0 和更高

當僅支持 Android 3.0 和更高版本時，可以通過如下方式定義 action bar 的背景：

`res/values/themes.xml`

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

```
<!-- 應用於程序或者活動的主題 -->
<style name="CustomActionBarTheme"
    parent="@android:style/Theme.Holo.Light.DarkActionBar">
    <item name="android: actionBarStyle">@style/MyActionBar</item>
</style>

<!-- ActionBar 樣式 -->
<style name="MyActionBar"
    parent="@android:style/Widget.Holo.Light.ActionBar.Solid.Inverse">
    <item name="android: background">@drawable/actionbar_background</item>
</style>
</resources>
```

然後，將主題應用到 app 全局或單個的 `activity` 之中：

```
<application android: theme="@style/CustomActionBarTheme" ... />
```

## 支持 Android 2.1 和更高

當使用 Support 庫時，上面同樣的主題必須被替代成如下：

`res/values/themes.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 應用於程序或者活動的主題 -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.AppCompat.Light.DarkActionBar">
        <item name="android: actionBarStyle">@style/MyActionBar</item>

        <!-- 支持庫兼容 -->
        <item name="actionBarStyle">@style/MyActionBar</item>
    </style>

    <!-- ActionBar 樣式 -->
    <style name="MyActionBar"
        parent="@style/Widget.AppCompat.Light.ActionBar.Solid.Inverse">
        <item name="android: background">@drawable/actionbar_background</item>

        <!-- 支持庫兼容 -->
        <item name="background">@drawable/actionbar_background</item>
    </style>
</resources>
```

然後，將主題應用到 app 全局或單個的 `activity` 之中：

```
<application android: theme="@style/CustomActionBarTheme" ... />
```

## 自定義文本顏色

修改 action bar 中的文本顏色，需要分別設置每個元素的屬性：

- Action bar 的標題：創建一種自定義樣式，並指定 `textColor` 屬性；同時，在自定義的 `ActionBarStyle` 中為 `titleTextStyle` 屬性指定為剛纔的自定義樣式。

注意：被應用到 `titleTextStyle` 的自定義樣式應該使用 `TextAppearance.Holo.Widget.ActionBar.Title` 作為父樣

式。

- Action bar tabs：在 `activity` 主題中重寫  `actionBarTabTextStyle`
- Action 按鈕：在 `activity` 主題中重寫  `actionMenuTextColor`

## 僅支持 Android 3.0 和更高

當僅支持 Android 3.0 和更高時，樣式 XML 文件應該是這樣的：

`res/values/themes.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 應用於程序或者活動的主題 -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.Holo">
        <item name="android:actionBarStyle">@style/MyActionBar</item>
        <item name="android:actionBarTabTextStyle">@style/MyActionBarTabText</item>
        <item name="android:actionMenuTextColor">@color/actionbar_text</item>
    </style>

    <!-- ActionBar 樣式 -->
    <style name="MyActionBar"
        parent="@style/Widget.Holo.ActionBar">
        <item name="android:titleTextStyle">@style/MyActionBarTitleText</item>
    </style>

    <!-- ActionBar 標題文本 -->
    <style name="MyActionBarTitleText"
        parent="@style/TextAppearance.Holo.Widget.ActionBar.Title">
        <item name="android:textColor">@color/actionbar_text</item>
    </style>

    <!-- ActionBar Tab標籤 文本樣式 -->
    <style name="MyActionBarTabText"
        parent="@style/Widget.Holo.ActionBar.TabText">
        <item name="android:textColor">@color/actionbar_text</item>
    </style>
</resources>
```

## 支持 Android 2.1 和更高

當使用 Support 庫時，樣式 XML 文件應該是這樣的：

`res/values/themes.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 應用於程序或者活動的主題 -->
    <style name="CustomActionBarTheme"
        parent="@style/Theme.AppCompat">
        <item name="android:actionBarStyle">@style/MyActionBar</item>
        <item name="android:actionBarTabTextStyle">@style/MyActionBarTabText</item>
        <item name="android:actionMenuTextColor">@color/actionbar_text</item>

        <!-- 支持庫兼容 -->
        <item name=" actionBarStyle">@style/MyActionBar</item>
        <item name=" actionBarTabTextStyle">@style/MyActionBarTabText</item>
        <item name=" actionMenuTextColor">@color/actionbar_text</item>
    </style>

    <!-- ActionBar 樣式 -->
    <style name="MyActionBar"
        parent="@style/Widget.AppCompat.ActionBar">
```

```

<item name="android:titleTextStyle">@style/MyActionBarTitleText</item>

<!-- 支持庫兼容 -->
<item name="titleTextStyle">@style/MyActionBarTitleText</item>
</style>

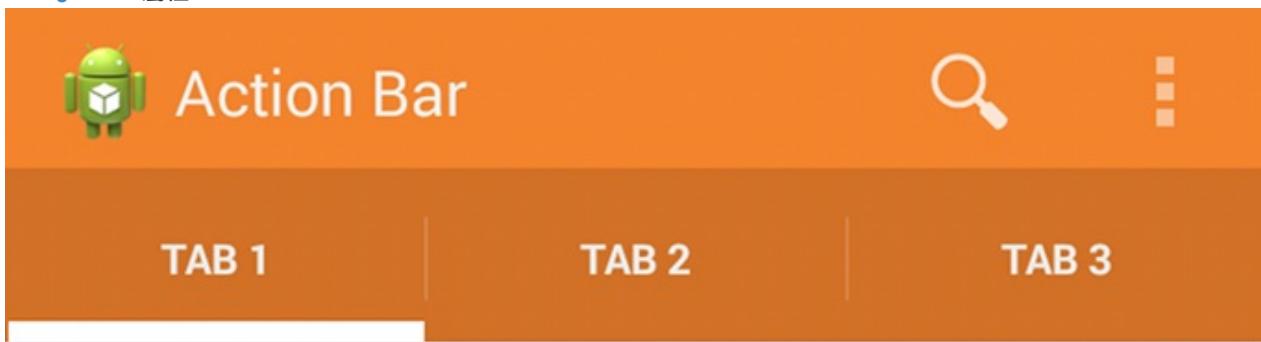
<!-- ActionBar 標題文本 -->
<style name="MyActionBarTitleText"
    parent="@style/TextAppearance.AppCompat.Widget.ActionBar.Title">
    <item name="android:textColor">@color/actionbar_text</item>
    <!-- 文本顏色屬性textColor是可以配合支持庫向後兼容的 -->
</style>

<!-- ActionBar Tab標籤文本樣式 -->
<style name="MyActionBarTabText"
    parent="@style/Widget.AppCompat.ActionBar.TabText">
    <item name="android:textColor">@color/actionbar_text</item>
    <!-- 文本顏色屬性textColor是可以配合支持庫向後兼容的 -->
</style>
</resources>

```

## 自定義 Tab Indicator

為 `activity` 創建一個自定義主題，通過重寫  `actionBarTabStyle` 屬性來改變 `navigation tabs` 使用的指示器。 `actionBarTabStyle` 屬性指向另一個樣式資源；在該樣式資源裏，通過指定一個`state-list drawable` 來重寫 `background` 屬性。



注意：一個`state-list drawable` 是重要的，它可以通過不同的背景來指出當前選擇的 tab 與其他 tab 的區別。更多關於如何創建一個 `drawable` 資源來處理多個按鈕狀態，請閱讀 [State List](#) 文檔。

例如，這是一個狀態列表 `drawable`，為一個 `action bar tab` 的多種不同狀態分別指定背景圖片：

`res/drawable/actionbar_tab_indicator.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

<!-- 按鈕沒有按下的狀態 -->

    <!-- 沒有焦點的狀態 -->
    <item android:state_focused="false" android:state_selected="false"
        android:state_pressed="false"
        android:drawable="@drawable/tab_unselected" />
    <item android:state_focused="false" android:state_selected="true"
        android:state_pressed="false"
        android:drawable="@drawable/tab_selected" />

    <!-- 有焦點的狀態 (例如D-Pad控制或者鼠標經過) -->
    <item android:state_focused="true" android:state_selected="false"
        android:state_pressed="false"
        android:drawable="@drawable/tab_unselected_focused" />
    <item android:state_focused="true" android:state_selected="true"
        android:state_pressed="false"
        android:drawable="@drawable/tab_selected_focused" />

</selector>

```

```
        android:drawable="@drawable/tab_selected_focused" />

<!-- 按鈕按下的狀態 -->

<!-- 沒有焦點的狀態 -->
<item android:state_focused="false" android:state_selected="false"
      android:state_pressed="true"
      android:drawable="@drawable/tab_unselected_pressed" />
<item android:state_focused="false" android:state_selected="true"
      android:state_pressed="true"
      android:drawable="@drawable/tab_selected_pressed" />

<!-- 有焦點的狀態 (例如D-Pad控制或者鼠標經過) -->
<item android:state_focused="true" android:state_selected="false"
      android:state_pressed="true"
      android:drawable="@drawable/tab_unselected_pressed" />
<item android:state_focused="true" android:state_selected="true"
      android:state_pressed="true"
      android:drawable="@drawable/tab_selected_pressed" />
</selector>
```

## 僅支持 Android 3.0 和更高

當僅支持 Android 3.0 和更高時，樣式 XML 文件應該是這樣的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 應用於程序或活動的主題 -->
    <style name="CustomActionBarTheme"
          parent="@style/Theme.Holo">
        <item name="android:actionBarTabStyle">@style/MyActionBarTabs</item>
    </style>

    <!-- ActionBar tabs 標籤樣式 -->
    <style name="MyActionBarTabs"
          parent="@style/Widget.Holo.ActionBar.TabView">
        <!-- 標籤指示器 -->
        <item name="android:background">@drawable/actionbar_tab_indicator</item>
    </style>
</resources>
```

## 支持 Android 2.1 和更高

當使用 Support 庫時，樣式 XML 文件應該是這樣的：

res/values/themes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 應用於程序或活動的主題 -->
    <style name="CustomActionBarTheme"
          parent="@style/Theme.AppCompat">
        <item name="android:actionBarTabStyle">@style/MyActionBarTabs</item>

        <!-- 支持庫兼容 -->
        <item name=" actionBarTabStyle">@style/MyActionBarTabs</item>
    </style>

    <!-- ActionBar tabs 樣式 -->
    <style name="MyActionBarTabs"
          parent="@style/Widget.AppCompat.ActionBar.TabView">
        <!-- 標籤指示器 -->
```

```
<item name="android:background">@drawable/actionbar_tab_indicator</item>

<!-- 支持庫兼容 -->
<item name="background">@drawable/actionbar_tab_indicator</item>
</style>
</resources>
```

## 更多資源

- 關於 action bar 的更多樣式屬性，請查看 [Action Bar 指南](#)
- 學習更多樣式的工作機制，請查看 [樣式和主題 指南](#)
- 全面的 action bar 樣式，請嘗試 [Android Action Bar 樣式生成器](#)

# ActionBar的覆蓋疊加

編寫:Vincent 4J - 原文:<http://developer.android.com/training/basics/actionbar/overlaying.html>

默認情況下，action bar 顯示在 `activity` 窗口的頂部，會稍微地減少其他佈局的有效空間。如果在用戶交互過程中要隱藏和顯示 action bar，可以通過調用 `ActionBar` 中的 `hide()` 和 `show()` 來實現。但是，這將導致 `activity` 基於新尺寸重新計算與繪製佈局。

為避免在 action bar 隱藏和顯示過程中調整佈局的大小，可以為 action bar 啓用疊加模式(overlay mode)。在疊加模式下，所有可用的空間都會被用來佈局就像 `ActionBar` 不存在一樣，並且 action bar 會疊加在佈局之上。這樣佈局頂部就會有點被遮擋，但當 action bar 隱藏或顯示時，系統不再需要調整佈局而是無縫過渡。

Note：如果希望 action bar 下面的佈局部分可見，可以創建一個背景部分透明的自定義式樣的 action bar，如圖 1 所示。關於如何定義 action bar 的背景，請查看 [自定義ActionBar的風格](#)。

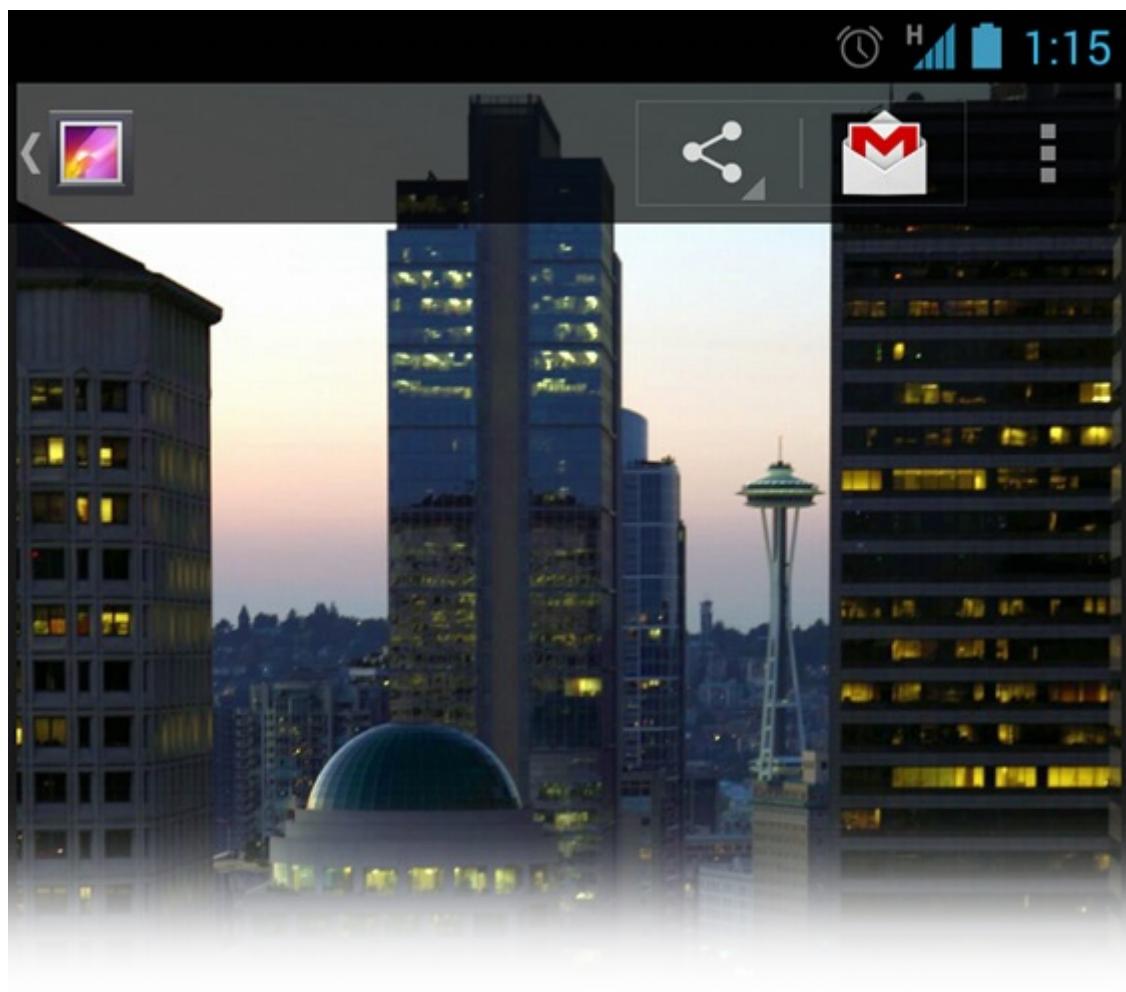


圖 1. 疊加模式下的 gallery action bar

## 啓用疊加模式(Overlay Mode)

要為 action bar 啓用疊加模式，需要自定義一個主題，該主題繼承於已經存在的 action bar 主題，並設置 `android:windowActionBarOverlay` 屬性的值為 `true`。

## 僅支持 Android 3.0 和以上

如果 `minSdkVersion` 為 11 或更高，自定義主題必須繼承 `Theme.Holo` 主題（或者其子主題）。例如：

```
<resources>
    <!-- 為程序或者活動應用的主題樣式 -->
    <style name="CustomActionBarTheme"
        parent="@android:style/Theme.Holo">
        <item name="android:windowActionBarOverlay">true</item>
    </style>
</resources>
```

## 支持 Android 2.1 和更高

如果為了兼容運行在 Android 3.0 以下版本的設備而使用了 Support 庫，自定義主題必須繼承 `Theme.AppCompat` 主題（或者其子主題）。例如：

```
<resources>
    <!-- 為程序或者活動應用的主題樣式 -->
    <style name="CustomActionBarTheme"
        parent="@android:style/Theme.AppCompat">
        <item name="android:windowActionBarOverlay">true</item>

        <!-- 兼容支持庫 -->
        <item name="windowActionBarOverlay">true</item>
    </style>
</resources>
```

注意，該主題包含兩種不同的 `windowActionBarOverlay` 式樣定義：一個帶 `android:` 前綴，另一個不帶。帶前綴的適用於包含該式樣的 Android 系統版本，不帶前綴的適用於通過從 Support 庫中讀取式樣的舊版本。

## 指定佈局的頂部邊距

當 action bar 啓用疊加模式時，它可能會遮擋住本應保持可見狀態的佈局。為了確保這些佈局始終位於 action bar 下部，可以使用 `actionBarSize` 屬性來指定頂部margin或padding的高度來到達。例如：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="?android:attr/actionBarSize">
    ...
</RelativeLayout>
```

如果在 action bar 中使用 Support 庫，需要移除 `android:` 前綴。例如：

```
<!-- 兼容支持庫 -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="?attr/actionBarSize">
    ...
</RelativeLayout>
```

在這種情況下，不帶前綴的 `?attr/actionBarSize` 適用於包括Android 3.0 和更高的所有版本。



# 兼容不同的設備

---

編寫:Lin-H - 原文:<http://developer.android.com/training/basics/supporting-devices/index.html>

全世界的Android設備有着各種各樣的大小和尺寸。通過各種各樣的設備類型，能使我們通過自己的app接觸到廣大的用戶羣體。為了能在各種Android平臺上使用，我們的app需要兼容各種不同的設備類型。某些例如語言，屏幕尺寸，Android的系統版本等重要的變量因素需要重點考慮。

本課程會教我們如何使用基礎的平臺功能，利用替代資源和其他功能，使app僅用一個app程序包(APK)，就能向用Android兼容設備的用戶提供最優的用戶體驗。

## Lessons

---

- [適配不同的語言](#)

學習如何使用字符串替代資源實現支持多國語言。

- [適配不同的屏幕](#)

學習如何根據不同尺寸分辨率的屏幕來優化用戶體驗。

- [適配不同的系統版本](#)

學習如何在使用新的用戶編程接口(API)時向下兼容舊版本Android。

# 適配不同的語言

編寫:Lin-H - 原文:<http://developer.android.com/training/basics/supporting-devices/languages.html>

把UI中的字符串存儲在外部文件，通過代碼提取，這是一種很好的做法。Android可以通過工程中的資源目錄輕鬆實現這一功能。

如果使用Android SDK Tools(詳見[創建Android項目\(Creating an Android Project\)](#))來創建工程，則在工程的根目錄會創建一個 `res/` 的目錄，目錄中包含所有資源類型的子目錄。其中包含工程的默認文件比如 `res/values/strings.xml`，用於保存字符串值。

## 創建區域設置目錄及字符串文件

為支持多國語言，在 `res/` 中創建一個額外的 `values` 目錄以連字符和ISO國家代碼結尾命名，比如 `values-es/` 是為語言代碼為"es"的區域設置的簡單的資源文件的目錄。Android會在運行時根據設備的區域設置，加載相應的資源。詳見[Providing Alternative Resources](#)。

若決定支持某種語言，則需要創建資源子目錄和字符串資源文件，例如:

```
MyProject/
  res/
    values/
      strings.xml
    values-es/
      strings.xml
    values-fr/
      strings.xml
```

添加不同區域語言的字符串值到相應的文件。

Android系統運行時會根據用戶設備當前的區域設置，使用相應的字符串資源。

例如，下面列舉了幾個不同語言對應不同的字符串資源文件。

英語(默認區域語言)，`/values/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="title">My Application</string>
  <string name="hello_world">Hello World!</string>
</resources>
```

西班牙語，`/values-es/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="title">Mi Aplicación</string>
  <string name="hello_world">Hola Mundo!</string>
</resources>
```

法語，`/values-fr/strings.xml`：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Note：可以在任何資源類型中使用區域修飾詞(或者任何配置修飾符)，比如為bitmap提供本地化的版本，更多信息見[Localization](#)。

## 使用字符串資源

我們可以在源代碼和其他XML文件中通過 `<string>` 元素的 `name` 屬性來引用自己的字符串資源。

在源代碼中可以通過 `R.string.<string_name>` 語法來引用一個字符串資源，很多方法都可以通過這種方式來接受字符串。

例如：

```
// Get a string resource from your app's Resources
String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

在其他XML文件中，每當XML屬性要接受一個字符串值時，你都可以通過 `@string/<string_name>` 語法來引用字符串資源。

例如：

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

# 適配不同的屏幕

編寫:Lin-H - 原文:<http://developer.android.com/training/basics/supporting-devices/screens.html>

Android用尺寸和分辨率這兩種常規屬性對不同的設備屏幕加以分類。我們應該想到自己的app會被安裝在各種屏幕尺寸和分辨率的設備中。這樣，app中就應該包含一些可選資源，針對不同的屏幕尺寸和分辨率，來優化其外觀。

- 有4種普遍尺寸：小(small)，普通(normal)，大(large)，超大(xlarge)
- 4種普遍分辨率：低精度(ldpi), 中精度(mdpi), 高精度(hdpi), 超高精度(xhdpi)

聲明針對不同屏幕所用的layout和bitmap，必須把這些可選資源放置在獨立的目錄中，這與適配不同語言時的做法類似。

同樣要注意屏幕的方向(橫向或縱向)也是一種需要考慮的屏幕尺寸變化，因此許多app會修改layout，來針對不同的屏幕方向優化用戶體驗。

## 創建不同的layout

為了針對不同的屏幕去優化用戶體驗，我們需要為每一種將要支持的屏幕尺寸創建唯一的XML文件。每一種layout需要保存在相應的資源目錄中，目錄以 <screen\_size> 為後綴命名。例如，對大尺寸屏幕(large screens)，一個唯一的layout文件應該保存在 res/layout-large/ 中。

Note:為了匹配合適的屏幕尺寸Android會自動地測量我們的layout文件。所以不需要因不同的屏幕尺寸去擔心UI元素的大小，而應該專注於layout結構對用戶體驗的影響。(比如關鍵視圖相對於同級視圖的尺寸或位置)

例如，這個工程包含一個默認layout和一個適配大屏幕的layout：

```
MyProject/
  res/
    layout/
      main.xml
    layout-large/
      main.xml
```

layout文件的名字必須完全一樣，為了對相應的屏幕尺寸提供最優的UI，文件的內容不同。

如平常一樣在app中簡單引用：

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

系統會根據app所運行的設備屏幕尺寸，在與之對應的layout目錄中加載layout。更多關於Android如何選擇恰當資源的信息，詳見[Providing Resources](#)。

另一個例子，這一個工程中有為適配橫向屏幕的layout:

```
MyProject/
```

```
res/
    layout/
        main.xml
    layout-land/
        main.xml
```

默認的，`layout/main.xml` 文件用作豎屏的layout。

如果想給橫屏提供一個特殊的layout，也適配於大屏幕，那麼則需要使用 `large` 和 `land` 修飾符。

```
MyProject/
    res/
        layout/          # default (portrait)
            main.xml
        layout-land/     # landscape
            main.xml
        layout-large/    # large (portrait)
            main.xml
        layout-large-land/ # large landscape
            main.xml
```

Note:Android 3.2及以上版本支持定義屏幕尺寸的高級方法，它允許我們根據屏幕最小長度和寬度，為各種屏幕尺寸指定與密度無關的layout資源。這節課程不會涉及這一新技術，更多信息詳見[Designing for Multiple Screens](#)。

## 創建不同的bitmap

我們應該為4種普遍分辨率:低，中，高，超高精度，都提供相適配的bitmap資源。這能使我們的app在所有屏幕分辨率中都能有良好的畫質和效果。

要生成這些圖像，應該從原始的矢量圖像資源着手，然後根據下列尺寸比例，生成各種密度下的圖像。

- `xhdpi: 2.0`
- `hdpi: 1.5`
- `mdpi: 1.0 (基準)`
- `ldpi: 0.75`

這意味着，如果針對`xhdpi`的設備生成了一張`200x200`的圖像，那麼應該為`hdpi`生成`150x150`,為`mdpi`生成`100x100`,和為`ldpi`生成`75x75`的圖片資源。

然後，將這些文件放入相應的drawable資源目錄中:

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

任何時候，當引用 `@drawable/awesomeimage` 時系統會根據屏幕的分辨率選擇恰當的bitmap。

Note:低密度(`ldpi`)資源是非必要的，當提供了`hdpi`的圖像，系統會把`hdpi`的圖像按比例縮小一半，去適配`ldpi`的

屏幕。

更多關於為app創建圖標assets的信息和指導，詳見[Iconography design](#)。

# 適配不同的系統版本

編寫:Lin-H - 原文:<http://developer.android.com/training/basics/supporting-devices/platforms.html>

新的Android版本會為我們的app提供更棒的APIs，但我們的app仍應支持舊版本的Android，直到更多的設備升級到新版本為止。這節課程將展示如何在利用新的APIs的同時仍支持舊版本Android。

**Platform Versions**的控制面板會定時更新，通過統計訪問Google Play Store的設備數量，來顯示運行每個版本的安卓設備的分佈。一般情況下，在更新app至最新Android版本時，最好先保證新版的app可以支持90%的設備使用。

Tip:為了能在幾個Android版本中都能提供最好的特性和功能，應該在我們的app中使用**Android Support Library**，它能使我們的app能在舊平臺上使用最近的幾個平臺的APIs。

## 指定最小和目標API級別

**AndroidManifest.xml**文件中描述了我們的app的細節及app支持哪些Android版本。具體來說，`<uses-sdk>` 元素中的 `minSdkVersion` 和 `targetSdkVersion` 屬性，標明在設計和測試app時，最低兼容API的級別和最高適用的API級別(這個最高的級別是需要通過我們的測試的)。例如：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

隨着新版本Android的發佈，一些風格和行為可能會改變，為了能使app能利用這些變化，而且能適配不同風格的用戶的設備，我們應該將 `targetSdkVersion` 的值儘量的設置與最新可用的Android版本匹配。

## 運行時檢查系統版本

Android在**Build**常量類中提供了對每一個版本的唯一代號，在我們的app中使用這些代號可以建立條件，保證依賴於高級別的API的代碼，只會在這些API在當前系統中可用時，纔會執行。

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Note:當解析XML資源時，Android會忽略當前設備不支持的XML屬性。所以我們可以安全地使用較新版本的XML屬性，而不需要擔心舊版本Android遇到這些代碼時會崩潰。例如如果我們設置 `targetSdkVersion="11"`，app會在Android 3.0或更高時默認包含**ActionBar**。然後添加menu items到action bar時，我們需要在自己的menu XML資源中設置 `android:showAsAction="ifRoom"`。在跨版本的XML文件中這麼做是安全的，因為舊版本的Android會簡單地忽略 `showAsAction` 屬性(就是這樣，你並不需要用到 `res/menu-v11/` 中單獨版本的文件)。

## 使用平飄風格和主題

Android提供了用戶體驗主題，為app提供基礎操作系統的外觀和體驗。這些主題可以在manifest文件中被應用於app中。通過使用內置的風格和主題，我們的app自然地隨着Android新版本的發佈，自動適配最新的外觀和體驗。

使activity看起來像對話框：

```
<activity android:theme="@android:style/Theme.Dialog">
```

使activity有一個透明背景：

```
<activity android:theme="@android:style/Theme.Translucent">
```

應用在 /res/values/styles.xml 中定義的自定義主題：

```
<activity android:theme="@style/CustomTheme">
```

使整個app應用一個主題(全部activities)在元素中添加 android:theme 屬性：

```
<application android:theme="@style/CustomTheme">
```

更多關於創建和使用主題，詳見[Styles and Themes](#)。

# 管理Activity的生命週期

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/activity-lifecycle/index.html>

當用戶進入，退出，回到你的App，在程序中的Activity 實例都經歷了生命週期中的不同狀態。例如，當你的activity第一次啓動的時候，它來到系統的前臺，開始接受用戶的焦點。在此期間，Android系統調用了一系列的生命週期中的方法。如果用戶執行了啓動另一個activity或者切換到另一個app(此時雖然當前activity不可見，但其實例與數據仍然存在)的操作，系統又會調用一些生命週期中的方法。

在生命週期的回調方法裏面，你可以聲明當用戶離開或者重新進入這個Activity所需要執行的操作。例如，如果你建立了一個streaming video player，在用戶切換到另外一個app的時候，你應該暫停video 並終止網絡連接。當用戶返回時，你可以重新建立網絡連接並允許用戶從同樣的位置恢復播放。

這一章會介紹一些Activity中重要的生命週期回調方法，如何使用那些方法使得程序符合用戶的期望且在activity不需要的時候不會導致系統資源的浪費。

完整的Demo示例：[ActivityLifecycle.zip](#)

## Lessons

- 啓動與銷燬Activity

學習關於activity生命週期的基礎知識，用戶如何啓動你的應用以及如何執行activity的創建。

- 暫停與恢復Activity

學習activity暫停發生時，你應該做哪些事情。

- 停止與重啓Activity

學習用戶離開你的activity與返回activity時會發生的事情。

- 重新創建Activity

學習當你的activity被銷燬時發生了什麼事情以及在有必要時如何重建你的activity。

# 啓動與銷燬Activity

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/activity-lifecycle/startling.html>

不像其他編程範式一樣：程序從 `main()` 方法開始啓動。Android系統根據生命週期的不同階段喚起對應的回調函數來執行代碼。系統存在啓動與銷燬一個activity的一套有序的回調函數。

本節課會介紹那些生命週期中最重要的回調函數，並演示如何處理啓動一個activity所涉及到的回調函數。

## 理解生命週期的回調

在一個activity的生命週期中，系統會像金字塔模型一樣去調用一系列的生命週期回調方法。Activity生命週期的每一個階段就像金字塔中的臺階。當系統創建了一個新的activity實例，每一個回調函數會向上一階移動activity狀態。處在金字塔頂端意味着當前activity處在前臺並且是用戶可以與它進行交互的狀態。

當用戶退出這個activity時，為了回收這個activity，系統會調用其它方法來向下一階移動activity狀態。在某些情況下，activity會隱藏在金字塔下等待(例如當用戶切換到其他app)，這個時候activity可以重新回到頂端(如果用戶回到這個activity)並且恢復用戶離開時的狀態。

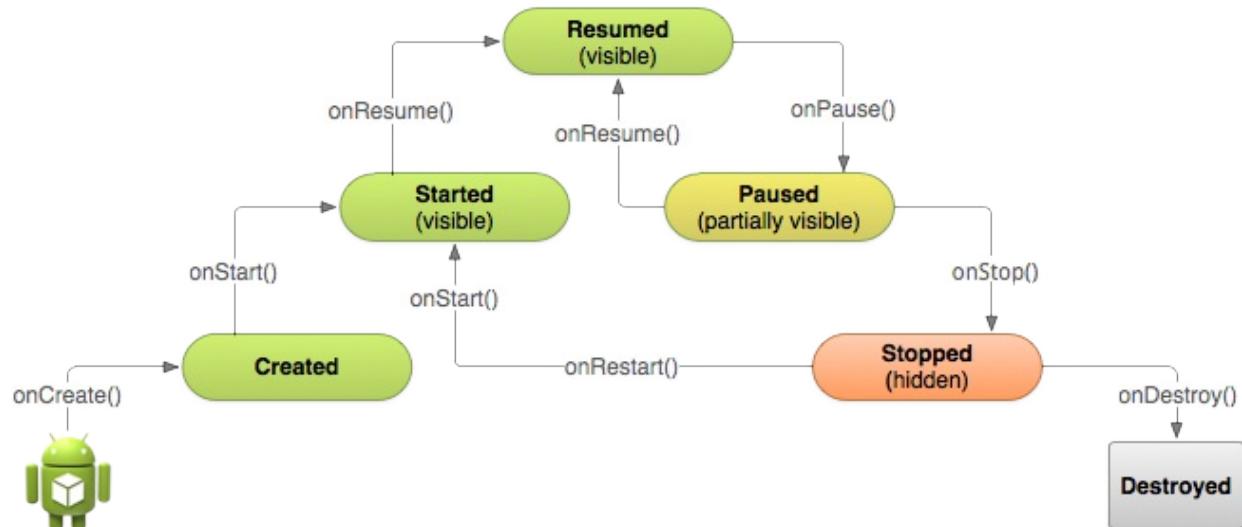


Figure 1. 下面這張圖講解了activity的生命週期：(這個金字塔模型要比之前Dev Guide裏面的生命週期圖更加容易理解，更加形象)

根據activity的複雜度，你也許不需要實現所有的生命週期方法。然而，你需要知道每一個方法的功能並確保你的app能夠像用戶期望的那樣執行。如何實現一個符合用戶期待的app，你需要注意下面幾點：

- 當使用你的app的時候，不會因為有來電通話或者切換到其他app而導致程序crash。
- 當用戶沒有激活某個組件的時候不要消耗寶貴的系統資源。
- 當離開你的app並且一段時間後返回，不要丟失用戶的使用進度。
- 當設備發送屏幕旋轉的時候，不會crash或者丟失用戶的使用進度。

在下面的課程中會介紹上圖所示的幾個生命狀態。然而，其中只有三個狀態是靜態的，這三個狀態下activity可以存在一段比較長的時間。(其它幾個狀態會很快就切換掉，停留的時間比較短暫)

- **Resumed**：在這個狀態，activity處在前臺，用戶可以與它進行交互。(通常也被理解為"running" 狀態)

- Paused：在這個狀態，activity被另外一個activity所遮蓋：另外的activity來到前臺，但是半透明的，不會覆蓋整個屏幕。被暫停的activity不會再接受用戶的輸入且不會執行任何代碼。(這裏的不會執行任何代碼並不代表了任何後臺線程都不會工作)
- Stopped：在這個狀態，activity完全被隱藏，不被用戶可見。可以認為是在後臺。當stopped，activity實例與它的所有狀態信息都會被保留，但是activity不能執行任何代碼。

其它狀態 (Created與Started)都是短暫的，系統快速的執行那些回調函數並通過執行下一階段的回調函數移動到下一個狀態。也就是說，在系統調用onCreate()，之後會迅速調用onStart()，之後再迅速執行onResume()。上面就是基本的activity生命週期。

## 指定程序首次啓動的Activity

當用戶從主界麵點擊你的程序圖標時，系統會調用你的app裏面的被聲明為"launcher" (or "main") activity中的onCreate()方法。這個Activity被用來當作你的程序的主要進入點。

你可以在AndroidManifest.xml中定義哪個activity作為你的主activity。

這個main activity必須在manifest使用包括 MAIN action 與 LAUNCHER category 的 <intent-filter> 標籤來聲明。例如：

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Note:當你使用Android SDK工具來創建Android工程時，工程中就包含了一個默認的聲明有這個filter的activity類。

如果你的程序中沒有一個activity聲明瞭MAIN action 或者LAUNCHER category，那麼在設備的主界面列表裏面不會呈現你的app圖標。

## 創建一個新的實例

大多數app都包括許多不同的activity，使得用戶可以執行不同的動作。不論這個activity是當用戶點擊應用圖標創建的main activtiy還是為了響應用戶行為而創建的其他activity，系統都會調用新的activity實例中的onCreate()方法。

你必須實現onCreate()方法來執行程序啓動所需要的基本邏輯。例如可以在onCreate()方法中定義UI以及實例化類成員變量。

例如：下面的onCreate()方法演示了為了建立一個activity所需要的一些基礎操作。如，聲明UI元素，定義成員變量，配置UI等。(onCreate裏面儘量少做事情，避免程序啓動太久都看不到界面)

```
TextView mTextView; // Member variable for text view in the layout

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

    // Set the user interface layout for this Activity
    // The layout file is defined in the project res/layout/main_activity.xml file
    setContentView(R.layout.main_activity);
```

```

// Initialize member TextView so we can manipulate it later
mTextView = (TextView) findViewById(R.id.text_message);

// Make sure we're running on Honeycomb or higher to use ActionBar APIs
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    // For the main activity, make sure the app icon in the action bar
    // does not behave as a button
    ActionBar actionBar = getActionBar();
    actionBar.setHomeButtonEnabled(false);
}
}

```

Caution : 用SDK\_INT來避免舊的系統調用了只在Android 2.0 ( API level 5 ) 或者更新的系統可用的方法（上述if條件中的代碼）。舊的系統調用了這些方法會拋出一個運行時異常。

一旦結束onCreate 操作，系統會迅速調用onStart() 與onResume()方法。你的activity不會在Created或者Started狀態停留。技術上來說，activity在onStart()被調用後會開始被用戶可見，但是 onResume()會迅速被執行使得activity停留在Resumed狀態，直到一些因素發生變化纔會改變這個狀態。例如接收到一個來電，用戶切換到另外一個activity，或者是設備屏幕關閉。

在後面的課程中，你將看到其他方法是如何使用的，onStart() 與 onResume()在用戶從Paused或 Stopped狀態中恢復的時候非常有用。

Note: onCreate() 方法包含了一個參數叫做savedInstanceState，這將會在後面的課程 - 重新創建activity的時候涉及到。

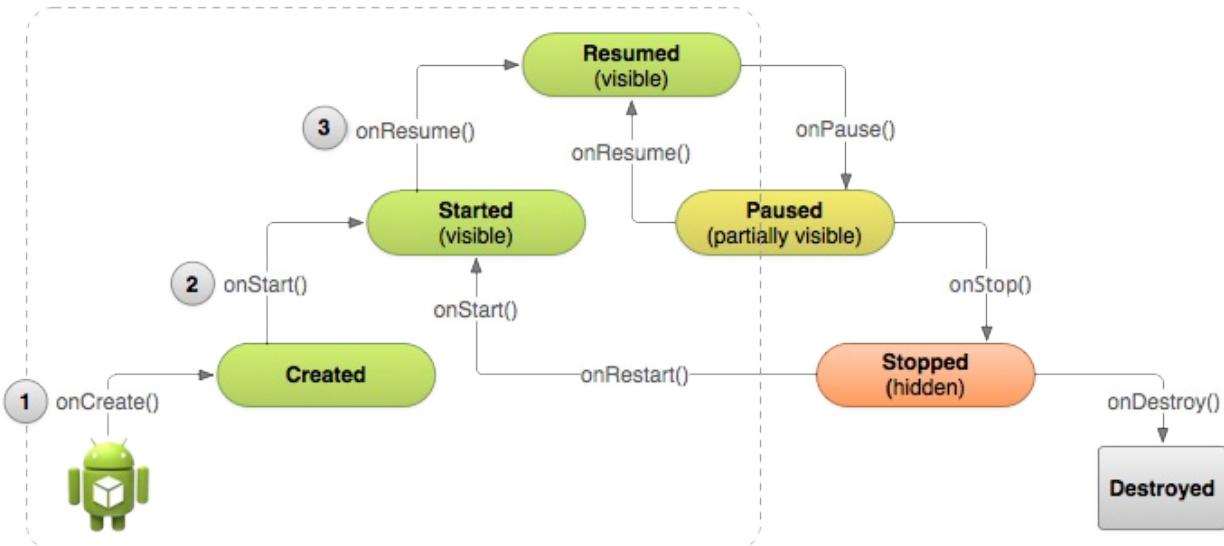


Figure 2. 上圖顯示了onCreate(), onStart() 和 onResume()是如何執行的。當這三個順序執行的回調函數完成後，activity會到達Resumed狀態。

## 銷燬Activity

activity的第一個生命週期回調函數是 **onCreate()**, 它最後一個回調是**onDestroy()**. 當系統收到需要將該activity徹底移除的信號時，系統會調用這個方法。

大多數 app並不需要實現這個方法，因為局部類的references會隨着activity的銷燬而銷燬，並且你的activity應該在onPause()與onStop()中執行清除activity資源的操作。然而，如果你的activity含有在onCreate調用時創建的後臺線程，或者是其他有可能導致內存泄漏的資源，你應該在OnDestroy()時殺死後臺線程，進行資源清理。

```
@Override  
public void onDestroy() {  
    super.onDestroy(); // Always call the superclass  
  
    // Stop method tracing that the activity started during onCreate()  
    android.os.Debug.stopMethodTracing();  
}
```

Note: 系統通常是在執行了onPause()與onStop() 之後再調用onDestroy() ，除非你的程序在onCreate()方法裏面就調用了finish()方法，。在某些情況下，例如你的[activity](#)只是做了一個臨時的邏輯跳轉的功能，它只是用來決定跳轉到哪一個[activity](#)，這樣的話，你需要在onCreate裏面去調用finish方法，這樣系統會直接就調用onDestory方法，其它生命週期的方法則不會被執行。

# 暫停與恢復Activity

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/activity-lifecycle/pausing.html>

在使用通常的app時，前端的activity有時候會被其他可見的組件而阻塞(obstructed)，這樣會導致當前的activity進入Pause狀態。例如，當打開一個半透明的activity時(例如以對話框的形式)，之前的activity會被暫停。只要之前的activity仍然被部分可見，這個activity就會一直處於Paused狀態。

然而，一旦之前的activity被完全阻塞並不可見，它則會進入Stop狀態(將在下一小節討論)。

當activity進入paused狀態，系統會調用activity中的onPause()方法，在這個方法裏面可以執行停止目前正在運行的任務的操作，比如暫停視頻播放或者是保存那些有可能需要長期保存的信息。如果用戶從暫停狀態回到當前activity，系統應該恢復那些數據並執行onResume()方法。

Note: 當你的activity調用onPause()，那可能意味著activity將被暫停一段時間，並且用戶很可能回到你的activity。然而，那也是用戶要離開你的activitiy的第一個信號。

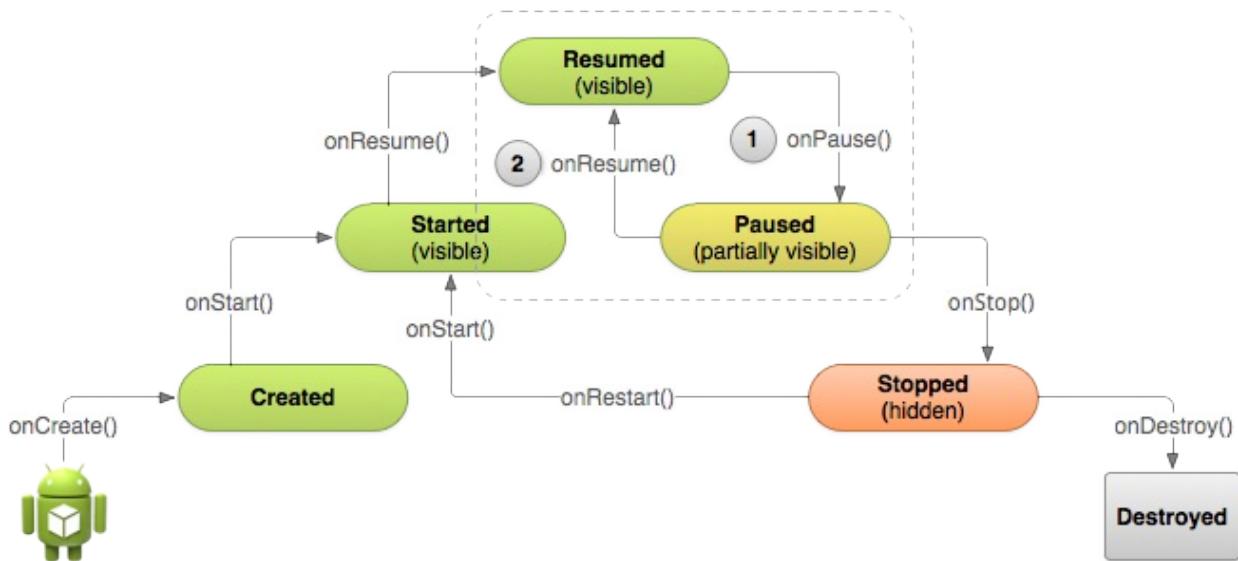


Figure 1. 當一個半透明的activity阻塞你的activity時，系統會調用onPause()方法並且這個activity會停留在Paused 狀態(1). 如果用戶在這個activity還是在Paused 狀態時回到這個activity，系統則會調用它的onResume() (2).

## 暫停Activity

當系統調用activity中的onPause()，從技術上講，那意味着activity仍然處於部分可見的狀態.但大多數時候，那意味着用戶正在離開這個activity並馬上會進入Stopped state. 你通常應該在onPause()回調方法裏面做下面的事情:

- 停止動畫或者是其他正在運行的操作，那些都會導致CPU的浪費.
- 提交沒有保存的改變，但是僅僅是在用戶離開時期待保存的內容(例如郵件草稿).
- 釋放系統資源，例如broadcast receivers, sensors (比如GPS), 或者是其他任何會影響到電量的資源。
- 例如，如果你的程序使用Camera, onPause()會是一個比較好的地方去做那些釋放資源的操作。

```
@Override  
public void onPause() {  
    super.onPause(); // Always call the superclass method first
```

```
// Release the Camera because we don't need it when paused  
// and other activities might need to use it.  
if (mCamera != null) {  
    mCamera.release()  
    mCamera = null;  
}  
}
```

通常，不應該使用onPause()來保存用戶改變的數據（例如填入表格中的個人信息）到永久存儲(File或者DB)上。僅僅當你確認用戶期待那些改變能夠被自動保存的時候（例如正在撰寫郵件草稿），你可以把那些數據存到永久存儲。然而，你應該避免在onPause()時執行CPU-intensive的工作，例如寫數據到DB，因為它會導致切換到下一個activity變得緩慢（你應該把那些heavy-load的工作放到onStop()去做）。

如果你的activity實際上是要被Stop，那麼你應該為了切換的順暢而減少在OnPause()方法裏面的工作量。

Note:當你的activity處於暫停狀態，Activity實例是駐留在內存中的，並且在activity恢復的時候重新調用。你不需要在恢復到Resumed狀態的一系列回調方法中重新初始化組件。

## 恢復activity

當用戶從Paused狀態恢復activity時，系統會調用onResume()方法。

請注意，系統每次調用這個方法時，activity都處於前臺，包括第一次創建的時候。所以，應該實現onResume()來初始化那些你在onPause方法裏面釋放掉的組件，並執行那些activity每次進入Resumed state都需要的初始化動作（例如開始動畫與初始化那些只有在獲取用戶焦點時才需要的組件）

下面的onResume()的例子是與上面的onPause()例子相對應的。

```
@Override  
public void onResume() {  
    super.onResume(); // Always call the superclass method first  
  
    // Get the Camera instance as the activity achieves full user focus  
    if (mCamera == null) {  
        initializeCamera(); // Local method to handle camera init  
    }  
}
```

# 停止與重啓Activity

編寫:kesenhoo - 原文: <http://developer.android.com/training/basics/activity-lifecycle/stopping.html>

恰當的停止與重啓你的activity是很重要的，在activity生命週期中，他們能確保用戶感知到程序的存在並不會丟失他們的進度。在下面一些關鍵的場景中會涉及到停止與重啓：

- 用戶打開最近使用app的菜單並從你的app切換到另外一個app，這個時候你的app是被停止的。如果用戶通過手機主界面的啓動程序圖標或者最近使用程序的窗口回到你的app，那麼你的activity會重啓。
- 用戶在你的app裏面執行啓動一個新的activity的操作，當前activity會在第二個activity被創建後stop。如果用戶點擊back按鈕，第一個activity會被重啓。
- 用戶在使用你的app時接收到一個來電通話。

Activity類提供了onStop()與onRestart()方法來允許在activity停止與重啓時進行調用。不像暫停狀態是部分阻塞UI，停止狀態是UI不再可見並且用戶的焦點轉移到另一個activity中。

Note: 因為系統在activity停止時會在內存中保存Activity的實例。有些時候你不需要實現onStop(),onRestart()甚至是onStart()方法。因為大多數的activity相對比較簡單，activity會自己停止與重啓，你只需要使用onPause()來停止正在運行的動作並斷開系統資源鏈接。

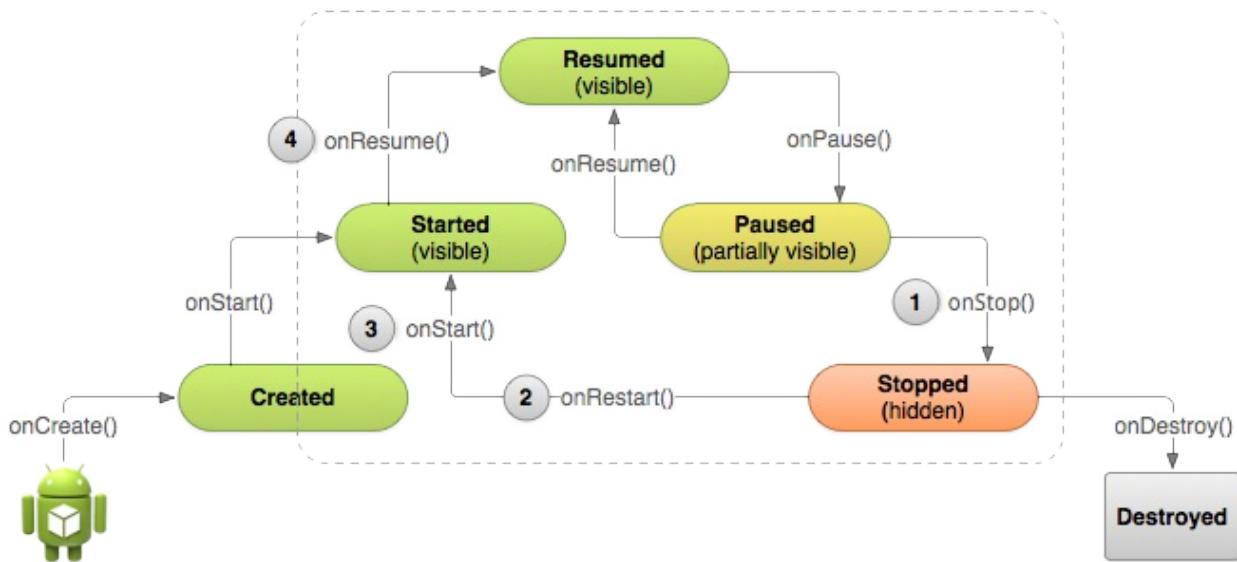


Figure 1. 上圖顯示：當用戶離開你的activity，系統會調用onStop()來停止activity (1)。這個時候如果用戶返回，系統會調用onRestart()(2)，之後會迅速調用onStart()(3)與onResume()(4)。請注意：無論什麼原因導致activity停止，系統總是會在onStop()之前調用onPause()方法。

## 停止activity

當activity調用onStop()方法，activity不再可見，並且應該釋放那些不再需要的所有資源。一旦你的activity停止了，系統會在不再需要這個activity時摧毀它的實例(和棧結構有關，通常back操作會導致前一個activity被銷燬)。在極端情況下，系統會直接殺死你的app進程，並且不執行activity的onDestroy()回調方法，因此你需要使用onStop()來釋放資源，從而避免內存泄漏。(這點需要注意)

儘管onPause()方法是在onStop()之前調用，你應該使用onStop()來執行那些CPU intensive的shut-down操作，例如往數據庫寫信息。

例如，下面是一個在onStop()的方法裏面保存筆記草稿到persistent storage的示例：

```
@Override  
protected void onStop() {  
    super.onStop(); // Always call the superclass method first  
  
    // Save the note's current draft, because the activity is stopping  
    // and we want to be sure the current note progress isn't lost.  
    ContentValues values = new ContentValues();  
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());  
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());  
  
    getContentResolver().update(  
        mUri, // The URI for the note to update.  
        values, // The map of column names and new values to apply to them.  
        null, // No SELECT criteria are used.  
        null // No WHERE columns are used.  
    );  
}
```

當activity已經停止，Activity對象會保存在內存中，並且在activity resume的時候重新被調用到。你不需要在恢復到Resumed state狀態前重新初始化那些被保存在內存中的組件。系統同樣保存了每一個在佈局中的視圖的當前狀態，如果用戶在EditText組件中輸入了text，它會被保存，因此不需要保存與恢復它。

Note: 即使系統會在activity stop的時候停止這個activity，它仍然會保存View對象的狀態(比如EditText中的文字)到一個Bundle中，並且在用戶返回這個activity時恢復他們(下一小節會介紹在activity銷燬與重新建立時如何使用Bundle來保存其他數據的狀態).

## 啓動與重啓activity

當你的activity從Stopped狀態回到前臺時，它會調用onRestart().系統再調用onStart()方法，onStart()方法會在每次你的activity可見時都會被調用。onRestart()方法則是只在activity從stopped狀態恢復時纔會被調用，因此你可以使用它來執行一些特殊的恢復(restoration)工作，請注意之前是被stopped而不是destroy。

使用onRestart()來恢復activity狀態是不太常見的，因此對於這個方法如何使用沒有任何的guidelines。然而，因為你的onStop()方法應該做清除所有activity資源的操作，你需要在重新啓動activtiy時重新實例化那些被清除的資源，同樣，你也需要在activity第一次創建時實例化那些資源。介於上面的原因，你應該使用onStart()作為onStop()所對應方法。因為系統會在創建activity與從停止狀態重啓activity時都會調用onStart().(這個地方的意思應該是說你在onStop裏面做了哪些清除的操作就應該在onStart裏面重新把那些清除掉的資源重新創建出來)

例如：因為用戶很可能在回到這個activity之前已經過了很長一段時間，所以onStart()方法是一個比較好的地方來驗證某些必須的系統特性是否可用。

```
@Override  
protected void onStart() {  
    super.onStart(); // Always call the superclass method first  
  
    // The activity is either being restarted or started for the first time  
    // so this is where we should make sure that GPS is enabled  
    LocationManager locationManager =  
        (LocationManager) getSystemService(Context.LOCATION_SERVICE);  
    boolean gpsEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);  
  
    if (!gpsEnabled) {  
        // Create a dialog here that requests the user to enable GPS, and use an intent  
        // with the android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS action  
        // to take the user to the Settings screen to enable GPS when they click "OK"  
    }  
}
```

```
@Override  
protected void onRestart() {  
    super.onRestart(); // Always call the superclass method first  
  
    // Activity being restarted from stopped state  
}
```

當系統Destory你的activity，它會為你的activity調用onDestroy()方法。因為我們會在onStop方法裏面做釋放資源的操作，那麼onDestory方法則是你最後去清除那些可能導致內存泄漏的地方。因此你需要確保那些線程都被destroyed並且所有的操作都被停止。

# 重新創建Activity

編寫:kesenhoo - 原文: <http://developer.android.com/training/basics/activity-lifecycle/recreating.html>

有幾個場景中，Activity是由於正常的程序行為而被Destory的。例如當用戶點擊返回按鈕或者是你的Activity通過調用`finish()`來發出停止信號。系統也有可能會在你的Activity處於stop狀態且長時間不被使用，或者是在前臺activity需要更多系統資源的時候把關閉後臺進程，這樣來獲取更多的內存。

當你的Activity是因為用戶點擊Back按鈕或者是activity通過調用`finish()`結束自己時，系統就丟失了對Activity實例的引用，因為前面的行為意味着不再需要這個activity了。然而，如果因為系統資源緊張而導致Activity的Destory，系統會在用戶回到這個Activity時有這個Activity存在過的記錄，系統會使用那些保存的記錄數據（描述了當Activity被Destory時的狀態）來重新創建一個新的Activity實例。那些被系統用來恢復之前狀態而保存的數據被叫做 "instance state"，它是一些存放在Bundle對象中的key-value pairs。（請注意這裏的描述，這對理解`onSaveInstanceState()`執行的時刻很重要）

**Caution:** 你的Activity會在每次旋轉屏幕時被destroyed與recreated。當屏幕改變方向時，系統會Destory與Recreate前臺的activity，因為屏幕配置被改變，你的Activity可能需要加載另一些替代的資源(例如layout)。

默認情況下，系統使用 Bundle 實例來保存每一個View(視圖)對象中的信息(例如輸入EditText 中的文本內容)。因此，如果你的Activity被destroyed與recreated，那麼layout的狀態信息會自動恢復到之前的狀態。然而，你的activity也許存在更多你想要恢復的狀態信息，例如記錄用戶Progress的成員變量(member variables)。

**Note:** 為了使Android系統能夠恢復Activity中的View的狀態，每個View都必須有一個唯一ID，由`android:id`定義。

為了讓你可以保存額外更多的數據到saved instance state。在Activity的生命週期裏面存在一個額外的回調函數，你必須重寫這個函數。這個回調函數並沒有在前面課程的圖片示例中顯示。這個方法是`onSaveInstanceState()`，當用戶離開你的Activity時，系統會調用它。當系統調用這個函數時，系統會在你的Activity被異常Destory時傳遞 Bundle 對象，這樣你可以增加額外的信息到Bundle中並保存到系統中。然後如果系統在Activity被Destory之後想重新創建這個Activity實例時，之前的那個Bundle對象會(系統)被傳遞到你的activity的`onRestoreInstanceState()`方法與`onCreate()`方法中。



Figure 2. 當系統開始停止你的Activity時，只有在Activity實例會需要重新創建的情況下纔會調用到`onSaveInstanceState()` (1)，在這個方法裏面可以指定額外的狀態數據到Bunde中。如果這個Activity被destroyed然後這個實例又需要被重新創建時，系統會傳遞在 (1) 中的狀態數據到`onCreate()` (2) 與`onRestoreInstanceState()`(3)。

(通常來說，跳轉到其他的activity或者是點擊Home都會導致當前的activity執行`onSaveInstanceState`，因為這種情況

下的activity都是有可能會被destory並且是需要保存狀態以便後續恢復使用的，而從跳轉的activity點擊back回到前一個activity，那麼跳轉前的activity是執行退棧的操作，所以這種情況下是不會執行onSaveInstanceState的，因為這個activity不可能存在需要重建的操作)

## 保存Activity狀態

當你的activity開始Stop，系統會調用 onSaveInstanceState()，你的Activity可以用鍵值對的集合來保存狀態信息。這個方法會默認保存Activity視圖的狀態信息，例如在 EditText 組件中的文本或者是 ListView 的滑動位置。

為了給Activity保存額外的狀態信息，你必須實現onSaveInstanceState()並增加key-value pairs到 Bundle 對象中，例如：

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

Caution: 總是需要調用 onSaveInstanceState() 方法的父類實現，這樣默認的父類實現才能保存視圖狀態的信息。

## 恢復Activity狀態

當你的Activity從Destory中重建。你可以從系統傳遞給你的Activity的Bundle中恢復保存的狀態。onCreate() 與 onRestoreInstanceState() 回調方法都接收到了同樣的Bundle，裏面包含了同樣的實例狀態信息。

因為 onCreate() 方法會在第一次創建新的Activity實例與重新創建之前被Destory的實例時都被調用，你必須在嘗試讀取 Bundle 對象前檢測它是否為null。如果它為null，系統則是創建一個新的Activity實例，而不是恢復之前被Destory的Activity。

下面是一個示例：演示在onCreate方法裏面恢復一些數據：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
}
```

你也可以選擇實現 onRestoreInstanceState()，而不是在onCreate方法裏面恢復數據。onRestoreInstanceState()

方法會在 `onStart()` 方法之後執行。系統僅僅會在存在需要恢復的狀態信息時纔會調用 `onRestoreInstanceState()`，因此你不需要檢查 `Bundle` 是否為 `null`。

```
public void onRestoreInstanceState(Bundle savedInstanceState) {  
    // Always call the superclass so it can restore the view hierarchy  
    super.onRestoreInstanceState(savedInstanceState);  
  
    // Restore state members from saved instance  
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);  
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);  
}
```

Caution: 與上面保存一樣，總是需要調用 `onRestoreInstanceState()` 方法的父類實現，這樣默認的父類實現才能保存視圖狀態的信息。如果想瞭解更多關於運行時狀態改變引起的 `recreate` 你的 `activity`。請參考 [Handling Runtime Changes](#).

# 使用Fragment建立動態UI

---

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/basics/fragments/index.html>

為了在Android上創建動態的、多窗口的用戶交互體驗，你需要將UI組件和Activity操作封裝成模塊進行使用，在activity中你可以對這些模塊進行切入切出操作。你可以用Fragment來創建這些模塊，Fragment就像一個嵌套的activity，擁有自己的佈局（layout）以及管理自己的生命週期。

如果一個fragment定義了自己的佈局，那麼在activity中它可以與其他的fragment生成不同的組合，從而為不同的屏幕尺寸生成不同的佈局（一個小的屏幕一次只放一個fragment，大的屏幕則可以兩個或以上的fragment）。

這一章將向你展示如何用fragment來創建動態的用戶體驗，以及在不同屏幕尺寸的設備上優化你的APP的用戶體驗。像運行着android1.6這樣老版本的設備，也都將繼續得到支持。

完整的Demo示例：[FragmentBasics.zip](#)

## Lessons

---

### [創建一個fragment](#)

學習如何創建一個fragment，以及實現它生命週期內的基本功能。

### [構建靈活的UI](#)

學習在APP內，對不同的屏幕尺寸用fragment構建不同的佈局。

### [與其他fragments交互](#)

學習fragment與activity以及其他fragment之間交互。

# 創建一個Fragment

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/basics/fragments/creating.html>

你可以把fragment想象成activity中一個模塊化的部分，它擁有自己的生命週期，接收自己的輸入事件，可以在activity運行過程中添加或者移除（有點像“子activity”，你可以在不同的activity裏面重複使用）。這一課教你繼承Support Library 中的Fragment，以使你的應用在Android1.6這樣的低版本上仍能保持兼容。

Note：如果的你的APP的最低API版本是11或以上，你不必使用Support Library，你可以直接使用框架裏面的Fragment和相關的API，這節課主要是講基於Support Library的API，Support Library有一個特殊的包名，有時候與平臺版本的API名字有些輕微的不一樣。

在開始這節課前，你必須先讓你的項目引用Support Library。如果你沒有使用過Support Library，你可以根據文檔Support Library Setup 來設置你的項目使用Support Library。當然，你也可以使用包含action bar的 v7 appcompat library。v7 appcompat library 兼容Android2.1(API level 7)，也包含了Fragment APIs。

## 創建一個Fragment類

創建一個fragment，首先需要繼承Fragment類，然後通過重寫生命週期的幾個關鍵方法來導入你APP的邏輯，就像activity一樣。

其中一個區別是當你創建Fragment的時候，你必須重寫onCreateView()回調方法來定義你的佈局。事實上，這是使Fragment運行起來，唯一一個需要你重寫的回調方法。比如，下面是一個自定義佈局的示例fragment.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

就像activity一樣，當fragment從activity添加或者移除、當activity生命週期發生變化時，fragment應該是實現生命週期回調來管理它的狀態。例如，當activity的onPause()被調用時，它裏面的所有fragment的onPause()方法也會被觸發。

更多關於fragment的聲明週期和回調方法，詳見[Fragments developer guide](#).

## 用XML將fragment添加到activity

fragments是可以重用的，模塊化的UI組件，每一個Fragment的實例都必須與一個FragmentActivity關聯。你可以在activity的XML佈局文件中定義每一個fragment來實現這種關聯。

Notes : FragmentActivity是Support Library提供的一個特殊activity，用來在API11版本以下的系統上處理fragment。如果你APP中的最低版本大於等於11，你可以使用普通的Activity。

下面是一個XML佈局的例子，當屏幕被認為是large(用目錄名稱中的 `large` 字符來區分)時，它在佈局中增加了兩個fragment.

`res/layout-large/news_articles.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment android:name="com.example.android.fragments.HeadlinesFragment"
        android:id="@+id/headlines_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFragment"
        android:id="@+id/article_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

Notes：更多關於不同屏幕尺寸創建不同佈局的信息，請閱讀[Supporting Different Screen Sizes](#)

然後將這個佈局文件用到你的activity中。

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

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

如果你用的是 `v7 appcompat library`，你的activity應該改為繼承`ActionBarActivity`，`ActionBarActivity`是`FragmentActivity`的一個子類（更多關於這方面的內容，請閱讀[Adding the Action Bar](#)）。

Note：當你用XML佈局文件的方式將Fragment添加進activity時，你的Fragment是不能被動態移除的。如果你想在用戶交互的時候把fragment切入與切出，你必須在activity啓動後，再將fragment添加進activity。這將在下節課講到。

# 建立靈活動態的UI

編寫:fastcome1985 - 原文:<http://developer.android.com/training/basics/fragments/fragment-ui.html>

如果你的APP設計成要支持範圍廣泛的屏幕尺寸時，在可利用的屏幕空間內，你可以通過在不同的佈局配置中重用你的fragment來優化你的用戶體驗。

比如，一個手機設備可能適合一次只有一個fragment的單面板用戶交互。相反，在更大屏幕尺寸的平板電腦上，你可能更想要兩個fragment並排在一起，用來向用戶展示更多信息。



Figure 1. 兩個fragments，在同一個activity不同屏幕尺寸中用不同的配置來展示。在大屏幕上，兩個fragment被並排放置，但是在手機上，一次只放置一個fragment，所以在用戶導航中，兩個fragment必須進行替換。

FragmentManager類提供了方法，讓你在activity運行時能夠對fragment進行添加，移除，替換，來實現動態的用戶體驗。

## 在activity運行時添加fragment

除了像上節課介紹的用 `<fragment>` 標簽在activity的佈局文件中定義fragment的方法外，你還可以在activity運行時動態添加fragment。如果你打算在activity的生命週期內替換fragment，這是必須的。

為了執行fragment的增加或者移除操作，你必須用 `FragmentManager` 創建一個`FragmentTransaction`對象，`FragmentTransaction`提供了用來增加、移除、替換以及其它一些操作的APIs。

如果你的activity允許移除或者替換fragment，那麼你應該在activity的`onCreate()`方法中添加初始的fragment(s)。

運用fragment（特別是那些你在運行時添加的）的一個很重要的規則就是在佈局中你必須有一個容器View，fragment的layout將會放在這個view裏面。

下面的這個佈局是上節課介紹的佈局的另一個版本，只不過這次的佈局一次只顯示一個fragment。為了從一個佈局替換為另外一個佈局，activity的佈局包含了一個空的 `FrameLayout`作為fragment的容器。

注意文件名與上節課的佈局一樣，但是文件目錄沒有 `large` 標識，所以下面這個佈局將被用在比large小的屏幕上，因為較小的屏幕無法同時放置兩個fragments。

res/layout/news\_articles.xml:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/fragment_container"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

在你的activity裏面，用Support Library APIs調用 `getSupportFragmentManager()`方法獲取`FragmentManager` 對象。然後調用 `beginTransaction()` 方法創建一個`FragmentTransaction`對象，並調用`add()`方法添加一個fragment。

你可以使用同一個 `FragmentTransaction`進行多次fragment事務。當你完成這些變化操作的時候，必須調用`commit()`方法。

例如，如何添加一個fragment到之前的layout中

```
import android.os.Bundle;  
import android.support.v4.app.FragmentActivity;  
  
public class MainActivity extends FragmentActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.news_articles);  
  
        // Check that the activity is using the layout version with  
        // the fragment_container FrameLayout  
        if (findViewById(R.id.fragment_container) != null) {  
  
            // However, if we're being restored from a previous state,  
            // then we don't need to do anything and should return or else  
            // we could end up with overlapping fragments.  
            if (savedInstanceState != null) {  
                return;  
            }  
  
            // Create a new Fragment to be placed in the activity layout  
            HeadlinesFragment firstFragment = new HeadlinesFragment();  
  
            // In case this activity was started with special instructions from an  
            // Intent, pass the Intent's extras to the fragment as arguments  
            firstFragment.setArguments(getIntent().getExtras());  
  
            // Add the fragment to the 'fragment_container' FrameLayout  
            getSupportFragmentManager().beginTransaction()  
                .add(R.id.fragment_container, firstFragment).commit();  
        }  
    }  
}
```

因為fragment是在activity運行時被添加進來的（不是在XML佈局中用 `<fragment>` 定義的），activity 可以移除這個fragment或者用另外一個來替換它。

## Fragment替換

替換fragment的過程與添加過程類似，只需要將`add()`方法替換爲 `replace()`方法。

記住當你執行fragment事務的時候，例如移除或者替換，你經常要適當地讓用戶可以向後導航與"撤銷"這次改變。爲了讓用戶向後導航fragment事務，你必須在`FragmentTransaction`提交前調用`addToBackStack()`方法。

Note：當你移除或者替換一個fragment並把它放入返回棧中時，被移除的fragment的生命週期是stopped(不是destroyed).當用戶返回重新恢復這個fragment,它的生命週期是restarts。如果你沒把fragment放入返回棧中，那麼當他被移除或者替換時，它的生命週期是destroyed。

下面是替換一個fragment的例子

```
// Create fragment and give it an argument specifying the article it should show
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack so the user can navigate back
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

[addToBackStack\(\)](#)方法提供了一個可選的String參數為事務指定了一個唯一的名字。這個名字不是必須的，除非你打算用[FragmentManager.BackStackEntry](#) APIs來進行一些高級的fragments操作。

# Fragments之間的交互

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/basics/fragments/communicating.html>

為了重用Fragment UI組件，你應該把每一個fragment都構建成完全的自包含的、模塊化的組件，定義他們自己的佈局與行爲。當你定義好這些模塊化的Fragment，你就可以讓他們關聯activity，使他們與application的邏輯結合起來，實現全局的複合的UI。

通常，你想fragment之間能相互交互，比如基於用戶事件改變fragment的內容。所有fragment之間的交互需要通過他們關聯的activity，兩個fragment之間不應該直接交互。

## 定義一個接口

為了讓fragment與activity交互，你可以在Fragment 類中定義一個接口，並且在activity中實現這個接口。Fragment在他們生命週期的onAttach()方法中獲取接口的實現，然後調用接口的方法來與Activity交互。

下面是一個fragment與activity交互的例子：

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Container Activity must implement this interface
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // This makes sure that the container activity has implemented
        // the callback interface. If not, it throws an exception
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                + " must implement OnHeadlineSelectedListener");
        }
    }

    ...
}
```

現在Fragment就可以通過調用 OnHeadlineSelectedListener 接口實例的 mCallback 中的 onArticleSelected()（也可以是其它方法）方法與activity傳遞消息。

舉個例子，在fragment中的下面的方法在用戶點擊列表條目時被調用，fragment 用回調接口來傳遞事件給父Activity.

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Send the event to the host activity
    mCallback.onArticleSelected(position);
}
```

# 實現接口

為了接收回調事件，宿主activity必須實現在Fragment中定義的接口。

舉個例子，下面的activity實現了上面例子中的接口。

```
public static class MainActivity extends Activity
    implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article
    }
}
```

# 傳消息給Fragment

宿主activity通過findFragmentById()方法來獲取fragment的實例，然後直接調用Fragment的public方法來向fragment傳遞消息。

例如，假設上面所示的activity可能包含另外一個fragment，這個fragment用來展示從上面的回調方法中返回的指定的數據。在這種情況下，activity可以把從回調方法中接收到的信息傳遞給這個展示數據的Fragment.

```
public static class MainActivity extends Activity
    implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article

        ArticleFragment articleFrag = (ArticleFragment)
            getSupportFragmentManager().findFragmentById(R.id.article_fragment);

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layout...

            // Call a method in the ArticleFragment to update its content
            articleFrag.updateArticleView(position);
        } else {
            // Otherwise, we're in the one-pane layout and must swap frags...

            // Create fragment and give it an argument for the selected article
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);

            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

            // Replace whatever is in the fragment_container view with this fragment,
            // and add the transaction to the back stack so the user can navigate back
            transaction.replace(R.id.fragment_container, newFragment);
            transaction.addToBackStack(null);

            // Commit the transaction
            transaction.commit();
        }
    }
}
```



# 數據保存

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/data-storage/index.html>

雖然可以在onPause()的時候保存一些信息以免用戶的使用進度被丟失，但是大多數Android app仍然是需要做保存數據的動作。大多數比較好的apps都需要保存用戶的設置信息，而且有一些apps必須維護大量的文件信息與DB信息。本章節將向你介紹Android中主要的數據存儲方法，包括：

## 保存到Preferences

學習使用Shared Preferences文件以Key-Value的方式保存簡要的信息。

## 保存到文件

學習保存基本的文件。

## 保存到數據庫

學習使用SQLite數據庫讀寫數據。

# 保存到Preference

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/data-storage/shared-preferences.html>

如果你有一個相對較小的key-value集合需要保存，你應該使用SharedPreferences APIs。 SharedPreferences 對象指向了一個保存key-value pairs的文件，併為讀寫他們提供了簡單的方法。每一個 SharedPreferences 文件均由 framework管理，其既可以是私有的，也可以是共享的。這節課會演示如何使用 SharedPreferences APIs 來存儲與檢索簡單的數據。

Note : SharedPreferences APIs 僅僅提供了讀寫key-value對的功能，請不要與Preference APIs相混淆。後者可以幫助你建立一個設置用戶配置的頁面（儘管它實際上是使用SharedPreferences 來實現保存用戶配置的）。如果想瞭解更多關於Preference APIs的信息，請參考Settings 指南。

## 獲取SharedPreference

你可以通過下面兩個方法之一來創建或者訪問shared preference 文件：

- `getSharedPreferences()` — 如果你需要多個通過名稱參數來區分的shared preference文件，名稱可以通過第一個參數來指定。你可以在你的app裏面通過任何一個Context 來執行這個方法。
- `getPreferences()` — 當你的activity僅僅需要一個shared preference文件時。因為這個方法會檢索activity下的默認的shared preference文件，並不需要提供文件名稱。

例如：下面的示例是在一個 Fragment 中被執行的，它會訪問名為 `R.string.preference_file_key` 的shared preference文件，並使用private模式來打開它，這種情況下，該文件僅能被你的app訪問了。

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
    getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

當命名你的shared preference文件時，你應該像 `"com.example.myapp.PREFERENCE_FILE_KEY"` 這樣來命名。

當然，如果你的activity僅僅需要一個shared preference文件時，你可以使用`getPreferences()`方法：

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

Caution: 如果你創建了一個MODE\_WORLD\_READABLE或者MODE\_WORLD\_WRITEABLE 模式的shared preference文件，那麼任何其他的app只要知道文件名，就可以訪問這個文件。

## 寫Shared Preference

為了寫 shared preferences 文件，需要通過執行`edit()`來創建一個 SharedPreferences.Editor 。

通過類似`putInt()`與`putString()`方法來傳遞keys與values。然後執行`commit()` 來提交改變。(後來有建議除非是出於線程同步的需要，否則請使用`apply()`方法來替代`commit()`，因為後者有可能會卡到UI Thread.)

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
```

```
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

## 讀Shared Preference

為了從shared preference中檢索讀取數據，可以通過類似 `getInt()` 與 `getString()`等方法來讀取。在那些方法裏面傳遞你想要獲取的value對應的key，並提供一個默認的value作為查找的key不存在時函數的返回值。如下：

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

# 保存到文件

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/data-storage/files.html>

Android使用與其他平臺類似的基於磁盤的文件系統(disk-based file systems)。這節課會描述如何在Android文件系統上使用 [File](#) 的讀寫APIs。

[File](#) 對象非常適合用於讀寫流式順序的數據。如圖片文件或是網絡中交換的數據。

這節課會演示在app中如何執行基本的文件操作任務。假定你已經對linux的文件系統與[java.io](#)中標準的I/O APIs有一定認識。

## 存儲在內部還是外部

所有的Android設備都有兩個文件存儲區域："internal" 與 "external" 存儲。那兩個名稱來自於早先的Android系統中，當時的大多設備都內置了不可變的內存 (internal storage)，然後再加上一個類似SD card (external storage) 這樣可以卸載的存儲部件。後來有一些設備把"internal" 與 "external" 的部分都做成不可卸載的內置存儲了，雖然如此，但是這一整塊還是從邏輯上有被劃分為"internal"與"external"的。只是現在不再以是否可以卸載來區分了。下面列出了兩者的區別：

Internal storage:

- 總是可用的
- 這裏的文件默認是只能被你的app所訪問的。
- 當用戶卸載你的app的時候，系統會把internal裏面的相關文件都清除乾淨。

Internal是在你想確保不被用戶與其他app所訪問的最佳存儲區域。

External storage:

- 並不總是可用的，因為用戶有時會通過USB存儲模式掛載外部存儲器，當取下掛載的這部分後，就無法對其進行訪問了。
- 是大家都可訪問的，因此你可能會失去保存在這裏的文件的訪問控制權。
- 當用戶卸載你的app時，系統僅僅會刪除保存在用 [getExternalFilesDir\(\)](#) 創建的目錄下的文件。

External是在你不需要嚴格的訪問權限並且你希望這些文件能夠被其他app所共享或者是允許用戶通過電腦訪問時的最佳存儲區域。

Tip: 儘管app是默認被安裝到internal storage的，你還是可以通過在程序的manifest文件中聲明[android:installLocation](#) 屬性來指定程序也可以被安裝到external storage。當某個程序的安裝文件很大且用戶的external storage空間大於internal storage時，他們會傾向與將該程序安裝到external storage。更多安裝信息，請參考[App Install Location](#)。

## 獲取External存儲的權限

為了寫數據到external storage，你必須在你的manifest文件中請求[WRITE\\_EXTERNAL\\_STORAGE](#)權限：

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

```
...  
</manifest>
```

Caution: 目前，所有的apps都可以在不指定某個專門的權限下做讀external storage的動作。但是，這在以後的版本中會有所改變。如果你的app只需要讀的權限(不是寫)，那麼你將需要聲明 **READ\_EXTERNAL\_STORAGE** 權限。為了確保你的app能持續地正常工作，你需要現在就聲明讀權限。

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

但是，如果你的程序有聲明寫的權限，那麼就默認有了讀的權限。

對於internal storage，你不需要聲明任何權限，因為你的應用默認就有權限讀寫它的internal storage目錄下的文件。

## 保存文件到Internal Storage

當保存文件到internal storage時，你可以通過執行下面兩個方法之一來獲取合適的目錄作為 **File** 的對象：

- **getFilesDir()**：返回一個**File**，代表了你的app的internal目錄。
- **getCacheDir()**：返回一個**File**，代表了你的app的internal緩存目錄。請確保這個目錄下的文件在一旦不再需要的時候能夠馬上被刪除，並對其大小進行合理限制，例如1MB。如果系統的內部存儲空間不夠，它會自行選擇刪除緩存文件。

你可以使用構造函數**File()** 在通過上述兩種方法創建的目錄下創建一個新的文件，如下：

```
File file = new File(context.getFilesDir(), filename);
```

同樣，你也可以執行**openFileOutput()** 來獲取一個 **FileOutputStream** 用於寫文件到internal目錄。如下：

```
String filename = "myfile";  
String string = "Hello world!";  
FileOutputStream outputStream;  
  
try {  
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);  
    outputStream.write(string.getBytes());  
    outputStream.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

如果，你需要緩存一些文件，你可以使用**createTempFile()**。例如：下面的方法從URL中抽取了一個文件名，然後再在程序的internal緩存目錄下創建了一個以這個文件名命名的文件。

```
public File getTempFile(Context context, String url) {  
    File file;  
    try {  
        String fileName = Uri.parse(url).getLastPathSegment();  
        file = File.createTempFile(fileName, null, context.getCacheDir());  
    } catch (IOException e) {  
        // Error while creating file
```

```
        }
        return file;
    }
```

Note: 你的app的internal storage 目錄是以你的app的包名作為標識存放在Android文件系統的特定目錄下 [data/data/com.example.xx]。從技術上講，如果你設置文件為可讀的，那麼其他app就可以讀取你的internal 文件。然而，其他app需要知道你的包名與文件名。若是你沒有設置為可讀或者可寫，其他app是沒有辦法讀寫的。因此只要你使用[MODE\\_PRIVATE](#)，那麼這些文件就不可能被其他app所訪問。

## 保存文件到External Storage

因為external storage可能是不可用的，比如遇到SD卡被拔出等情況時。因此你應該在訪問之前去檢查其是否可用。你可以通過執行 [getExternalStorageState\(\)](#)來查詢external storage的狀態。如果返回的狀態是[MEDIA\\_MOUNTED](#)，那麼你可以讀寫。示例如下：

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

儘管external storage對於用戶與其他app是可修改的，但是你可能會保存下面兩種類型的文件。

- **Public files** :這些文件對於用戶與其他app來說是public的。當用戶卸載你的app時，這些文件應該保留。例如，那些被你的app拍攝的圖片或者下載的文件。
- **Private files**: 這些文件應該是被你的app所擁有的，它們應該在你的app被卸載時刪除掉。儘管由於存儲在external storage，這些文件從技術上而言可以被用戶與其他app所訪問，但實際上那些文件對於其他app是沒有意義的。因此，當用戶卸載你的app時，系統會刪除你的app的private目錄。例如，那些被你的app下載的緩存文件。

如果你想要保存文件為public形式的，請使用[getExternalStoragePublicDirectory\(\)](#)方法來獲取一個 [File](#) 對象來表示存儲在external storage的目錄。這個方法會需要你帶有一個特定的參數來指定這些public的文件類型，以便於與其他public文件進行分類。參數類型包括[DIRECTORY\\_MUSIC](#) 或者 [DIRECTORY\\_PICTURES](#)。如下:

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

如果你想要保存文件爲私有的方式，你可以通過執行 `getExternalFilesDir()` 來獲取相應的目錄，並且傳遞一個指示文件類型的參數。每一個以這種方式創建的目錄都會被添加到包含所有你app的external storage文件的父目錄下。這個父目錄（包括目錄下的文件）會在用戶卸載你的app時被系統刪除。如下示例：

```
public File getAlbumStorageDir(Context context, String albumName) {
    // Get the directory for the app's private pictures directory.
    File file = new File(context.getExternalFilesDir(
        Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

如果剛開始的時候，沒有預定義的子目錄存放你的文件，你可以在 `getExternalFilesDir()`方法中傳遞 `null`。它會返回你的app在external storage下的private的根目錄。

請記住，`getExternalFilesDir()` 方法創建的目錄是位於另一個父目錄下，這個父目錄在app被卸載時會被系統刪除。如果你的文件想在app被刪除時仍然保留，請使用`getExternalStoragePublicDirectory()`。

不管你是使用 `getExternalStoragePublicDirectory()` 來存儲可以共享的文件，還是使用 `getExternalFilesDir()` 來儲存那些對與你的app來說是私有的文件。有一點很重要，那就是你要使用那些類似 `DIRECTORY_PICTURES` 的API的常量。那些目錄類型參數可以確保那些文件被系統正確的對待。例如，那些以 `DIRECTORY_RINGTONES` 類型保存的文件就會被系統的media scanner認爲是ringtone而不是音樂。

## 查詢剩餘空間

如果你事先知道你想要保存的文件大小，你可以通過執行`getFreeSpace()` or `getTotalSpace()` 來判斷是否有足夠的空間來保存文件，從而避免發生`IOException`。那些方法提供了當前可用的空間還有存儲系統的總容量。

然而，系統並不能保證你可以寫入通過 `getFreeSpace()` 查詢到的容量文件，如果查詢的剩餘容量比你的文件大小多幾MB，或者說文件系統使用率還不足90%，這樣則可以繼續進行寫的操作，否則你最好不要寫進去。

Note：你並沒有被強制要求在寫文件之前一定有要去檢查剩餘容量。你可以嘗試先做寫的動作，然後捕獲 `IOException`。這種做法僅適合於你事先並不知道你想要寫的文件的確切大小。例如，如果在把PNG圖片轉換成JPEG之前，你並不知道最終生成的圖片大小是多少。

## 刪除文件

你應該在不需要使用某些文件的時候，刪除它。刪除文件最直接的方法是直接執行文件的 `delete()` 方法。

```
myFile.delete();
```

如果文件是保存在internal storage，你可以通過 `Context` 來訪問並通過執行 `deleteFile()` 進行刪除

```
myContext.deleteFile(fileName);
```

Note: 當用戶卸載你的app時，android系統會刪除以下文件：

- 所有保存到internal storage的文件。

- 所有使用 `getExternalFilesDir()` 方式保存在 external storage 的文件。

然而，通常來說，你應該手動刪除所有通過 `getCacheDir()` 方式創建的緩存文件，以及那些不會再用到的文件。

# 保存到數據庫

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/data-storage/databases.html>

對於重複或者結構化的數據（如聯繫人信息）等保存到DB是個不錯的主意。這節課假定你已經熟悉SQL數據庫的操作。在Android上可能會使用到的APIs，可以從[android.database.sqlite](#)包中找到。

## 定義Schema與Contract

SQL中一箇中重要的概念是schema：一種DB結構的正式聲明。schema是從你創建DB的SQL語句中生成的。你可能會發現創建一個伴隨類（companion class）是很有益的，這個類稱為合約類（contract class），它用一種系統化並且自動生成文檔的方式，顯示指定了你的schema樣式。

Contract Class是一些常量的容器。它定義了例如URLs，表名，列名等。這個contract類允許你在同一個包下與其他類使用同樣的常量。它讓你只需要在一個地方修改列名，然後這個列名就可以自動傳遞給你整個code。

一個組織你的contract類的好方法是在你的類的根層級定義一些全局變量，然後為每一個table來創建內部類。

Note：通過實現 [BaseColumns](#) 的接口，你的內部類可以繼承到一個名為\_ID的主鍵，這個對於Android裏面的一些類似cursor adaptor類是很有必要的。這麼做不是必須的，但這樣能夠使得你的DB與Android的framework能夠很好的相容。

例如，下面的例子定義了表名與這個表的列名：

```
public final class FeedReaderContract {  
    // To prevent someone from accidentally instantiating the contract class,  
    // give it an empty constructor.  
    public FeedReaderContract() {}  
  
    /* Inner class that defines the table contents */  
    public static abstract class FeedEntry implements BaseColumns {  
        public static final String TABLE_NAME = "entry";  
        public static final String COLUMN_NAME_ENTRY_ID = "entryid";  
        public static final String COLUMN_NAME_TITLE = "title";  
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";  
        ...  
    }  
}
```

## 使用SQL Helper創建DB

當你定義好了你的DB的結構之後，你應該實現那些創建與維護db與table的方法。下面是一些典型的創建與刪除table的語句。

```
private static final String TEXT_TYPE = " TEXT";  
private static final String COMMA_SEP = ",";  
private static final String SQL_CREATE_ENTRIES =  
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +  
    FeedEntry._ID + " INTEGER PRIMARY KEY," +  
    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_SEP +  
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +  
    ... // Any other options for the CREATE command  
    " )";
```

```
private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

就像保存文件到設備的internal storage一樣，Android會保存db到你的程序的private的空間上。你的數據是受保護的，因為那些區域默認是私有的，不可被其他程序所訪問。

在SQLiteOpenHelper類中有一些很有用的APIs。當你使用這個類來做一些與你的db有關的操作時，系統會對那些有可能比較耗時的操作（例如創建與更新等）在真正需要的時候纔去執行，而不是在app剛啓動的時候就去做那些動作。你所需要做的僅僅是執行getWritableDatabase()或者getReadableDatabase()。

Note：因為那些操作可能是很耗時的，請確保你在background thread ( AsyncTask or IntentService ) 裏面去執行 getWritableDatabase() 或者 getReadableDatabase() 。

為了使用 SQLiteOpenHelper，你需要創建一個子類並重寫onCreate(), onUpgrade()與onOpen()等callback方法。你也許還需要實現onDowngrade()，但是這並不是必需的。

例如，下面是一個實現了SQLiteOpenHelper 類的例子：

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

為了訪問你的db，需要實例化你的 SQLiteOpenHelper的子類：

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
```

## 添加信息到DB

通過傳遞一個 ContentValues 對象到insert()方法將數據插入到DB：

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);
```

```
// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
    FeedEntry.TABLE_NAME,
    FeedEntry.COLUMN_NAME_NULLABLE,
    values);
```

`insert()` 方法的第一個參數是table名，第二個參數會使得系統自動對那些 `ContentValues` 沒有提供數據的列填充數據為 `null`，如果第二個參數傳遞的是`null`，那麼系統則不會對那些沒有提供數據的列進行填充。

## 從DB中讀取信息

為了從DB中讀取數據，需要使用`query()`方法， 傳遞你需要查詢的條件。查詢後會返回一個 `Cursor` 對象。

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_UPDATED,
    ...
};

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
    FeedEntry.TABLE_NAME, // The table to query
    projection,           // The columns to return
    selection,            // The columns for the WHERE clause
    selectionArgs,         // The values for the WHERE clause
    null,                 // don't group the rows
    null,                 // don't filter by row groups
    sortOrder             // The sort order
);
```

要查詢在cursor中的行，使用cursor的其中一個`move`方法，但必須在讀取值之前調用。一般來說你應該先調用 `moveToFirst()` 函數，將讀取位置置於結果集最開始的位置。對每一行，你可以使用cursor的其中一個`get`方法比如 `getString()` 或 `getLong()` 獲取列的值。對於每一個`get`方法必須傳遞你想要獲取的列的索引位置(index position)，索引位置可以通過調用 `getColumnIndex()` 或 `getColumnIndexOrThrow()` 獲得。

下面是演示如何從course對象中讀取數據信息：

```
cursor.moveToFirst();
long itemId = cursor.getLong(
    cursor.getColumnIndexOrThrow(FeedEntry._ID))
);
```

## 刪除DB中的信息

和查詢信息一樣，刪除數據同樣需要提供一些刪除標準。DB的API提供了一個防止SQL注入的機制來創建查詢與刪除標準。

SQL Injection：(隨着B/S模式應用開發的發展，使用這種模式編寫應用程序的程序員也越來越多。但是由於程

序員的水平及經驗也參差不齊，相當大一部分程序員在編寫代碼的時候，沒有對用戶輸入數據的合法性進行判斷，使應用程序存在安全隱患。用戶可以提交一段數據庫查詢代碼，根據程序返回的結果，獲得某些他想得知的數據，這就是所謂的SQL Injection，即SQL注入)

這個機制把查詢語句劃分為選項條款與選項參數兩部分。條款部分定義了查詢的列的特徵，參數部分用來測試是否符合前面的條款。(這裏翻譯的怪怪的，附上原文，The clause defines the columns to look at, and also allows you to combine column tests. The arguments are values to test against that are bound into the clause.) 因為處理的結果與通常的SQL語句不同，這樣可以避免SQL注入問題。

```
// Define 'where' part of query.  
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";  
// Specify arguments in placeholder order.  
String[] selectionArgs = { String.valueOf(rowId) };  
// Issue SQL statement.  
db.delete(table_name, selection, selectionArgs);
```

## 更新數據

當你需要修改DB中的某些數據時，使用 `update()` 方法。

更新操作結合了插入與刪除的語法。

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();  
  
// New value for one column  
ContentValues values = new ContentValues();  
values.put(FeedEntry.COLUMN_NAME_TITLE, title);  
  
// Which row to update, based on the ID  
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";  
String[] selectionArgs = { String.valueOf(rowId) };  
  
int count = db.update(  
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,  
    values,  
    selection,  
    selectionArgs);
```

# 與其他應用的交互

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/intents/index.html>

一個Android app通常都會有好幾個activities。每一個activity的界面都扮演者用戶接口的角色，允許用戶執行一些特殊任務（例如查看地圖或者是開始拍照等）。為了讓用戶從一個activity跳到另外一個activity，你的app必須使用Intent來定義你的app的意圖。當你使用startActivity()的方法，且參數是intent時，系統會使用這個 Intent 來定義並啓動合適的app組件。使用intents甚至還可以讓你的app啓動另一個app裏面的activity。

一個 Intent 可以顯式的指明需要啓動的模塊（用一個指定的Activity實例），也可以隱式的指明自己可以處理哪種類型的動作（比如拍一張照等）。

這一章節會演示如何使用Intent 來做一些與其他app之間的簡單交互。比如啓動另外一個app，從其他app接受數據，以及使得你的app能夠響應從其他發出的intent等。

## Lessons

---

### [Intent的發送\(Sending the User to Another App \)](#)

演示如何創建隱式的Intent來喚起能夠接收這個動作的App。

### [接收Activity返回的結果\(Getting a Result from an Activity\)](#)

演示如何啓動另外一個Activity並接收返回值。

### [Intent過濾\(Allowing Other Apps to Start Your Activity\)](#)

演示如何通過定義隱式的Intent的過濾器來使得你的應用能夠被其他應用喚起。

# Intent的發送

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/intents/sending.html>

Android中最重要的特徵之一就是可以利用一個帶有 `action` 的 `intent` 使得當前app能夠跳轉到其他的app。例如：如果你的app擁有一個地址想要顯示在地圖上，你並不需要在你的app裏面創建一個activity用來顯示地圖。你只需要使用 Intent來發出查看地址的請求。Android系統則會啓動能夠顯示地圖的程序來呈現那個地址。

正如在1.1章節:建立你的第一個App(Building Your First App)中所說的，你必須使用intent來在同一個app的兩個activity之間進行切換。在那種情況下通常是定義一個顯式(explicit)的intent，它指定了需要啓動組件的類名。然而，當你想要叫起不同的app來執行某個動作(比如查看地圖)，則必須使用隱式(implicit)的intent。

這節課會介紹如何為特殊的動作創建一個implicit intent，並使用它來啓動另外一個app去執行intent中的action。

## 建立隱式的Intent

Implicit intents並不會聲明需要啓動的組件的類名，而是聲明一個需要執行的動作。這個action指定了你想做的事情，例如查看，編輯，發送或者是獲取什麼。Intents通常會在發送action的同時附帶一些數據，例如你想要查看的地址或者是你想要發送的郵件信息。數據的類型取決於你想要創建的Intent，這些數據可能是Uri，或者是其他規定的數據類型，或者根本不需要數據也是有可能的。

如果你的數據是一個Uri，會有一個簡單的Intent() constructor用來定義action與data。

例如，下面是一個帶有指定電話號碼的intent。

```
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

當你的app通過執行startActivity()來啓動這個intent時，Phone app會使用之前的電話號碼來撥出這個電話。

下面是一些其他intent的例子：

- 查看地圖:

```
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
// Or map point based on latitude/longitude
// Uri location = Uri.parse("geo:37.42219,-122.08364?z=14"); // z param is zoom level
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

- 查看網頁:

```
Uri webpage = Uri.parse("http://www.android.com");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

另外一些需要 extra 數據的implicit intent。你可以使用 putExtra() 方法來添加那些數據。

默認的，系統會根據Uri數據類型來決定需要哪些合適的 MIME type。如果你沒有在intent中包含一個Uri，則通常需要使

用 `setType()` 方法來指定intent附帶的數據類型。設置MIME type 是為了指定哪些activity應該接受這個intent。例如：

- 發送一個帶附件的email:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
// The intent does not have a URI, so declare the "text/plain" MIME type
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jon@example.com"}); // recipients
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://path/to/email/attachment"));
// You can also attach multiple items by passing an ArrayList of Uris
```

- 創建一個日曆事件:

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT, Events.CONTENT_URI);
Calendar beginTime = Calendar.getInstance().set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance().set(2012, 0, 19, 10, 30);
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis());
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis());
calendarIntent.putExtra(Events.TITLE, "Ninja class");
calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

Note: 這個intent for Calendar的例子只使用於>=API Level 14。

創建一個日曆事件的代碼實際用的時候報錯。下面貼一段譯者對這部分測試編譯通過的代碼：

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT, CalendarContract.Events.CONTENT_URI);
Calendar beginTime = Calendar.getInstance();
beginTime.set(2015, 4, 23, 12, 0);
Calendar endTime = Calendar.getInstance();
endTime.set(2015, 4, 23, 14, 30);
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis());
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis());
calendarIntent.putExtra(CalendarContract.Events.TITLE, "Ninja class");
calendarIntent.putExtra(CalendarContract.Events.EVENT_LOCATION, "Secret dojo");
```

Note: 請儘可能具體地定義你的intent。例如，如果你想要使用ACTION\_VIEW的intent來顯示一張圖片，你還應該指定MIME type 為 `image/*`。這樣能夠阻止其他能夠“查看”其他數據類型的app（比如一個地圖app）被這個intent叫起。

## 驗證是否有App去接收這個Intent

儘管Android系統會確保每一個確定的intent會被系統內置的app（例如 Phone, Email, 或者 Calendar app）之一接收，但是你還是應該在觸發一個intent之前驗證是否有App接受這個intent。

Caution: 如果你觸發了一個intent，而且沒有任何一個app會去接收這個intent，那麼你的app會crash。

為了驗證是否有合適的activity會響應這個intent，需要執行`queryIntentActivities()` 來獲取到能夠接收這個intent的所有activity的list。如果返回的List非空，那麼你纔可以安全的使用這個intent。例如：

```
PackageManager packageManager = getPackageManager();
List activities = packageManager.queryIntentActivities(intent,
    PackageManager.MATCH_DEFAULT_ONLY);
boolean isIntentSafe = activities.size() > 0;
```

如果 `isIntentSafe` 是 `true`，那麼至少有一個app可以響應這個intent。如果是 `false` 則說明沒有app可以handle這個intent。

Note: 你必須在第一次使用之前做這個檢查，若是不可行，則應該關閉這個功能。如果你知道某個確切的app能夠handle這個intent，你也應該提供給用戶去下載這個app的鏈接。(詳細請看[link to your product on Google Play](#)).

## 使用Intent來啓動Activity

當你創建好了intent並且設置好了extra數據，通過執行`startActivity()`來發送到系統。如果系統確定有多個activity可以handle這個intent，它會顯示出一個dialog，讓用戶選擇啓動哪個app。如果系統發現只有一個app可以handle這個intent，那麼就會直接啓動這個app。

```
startActivity(intent);
```



下面是一個完整的例子，演示瞭如何創建一個intent來查看地圖，驗證有app可以handle這個intent，然後啓動它。

```
// Build the intent
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

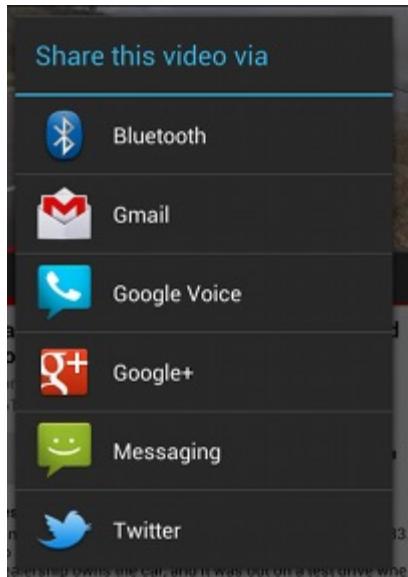
// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

## 顯示分享App的選擇界面

請注意，當你以`startActivity()`的形式傳遞一個intent，並且有多個app可以handle時，用戶可以在彈出dialog的時候，選擇默認啓動的app（通過勾選dialog下面的選擇框，如上圖所示）。這個功能對於用戶有特殊偏好的時候非常有用（例如用戶總是喜歡啓動某個app來查看網頁，總是喜歡啓動某個camera來拍照）。

然而，如果用戶希望每次都彈出選擇界面，而且每次都不確定會選擇哪個app啓動，例如分享功能，用戶選擇分享到哪個app都是不確定的，這個時候，需要強制彈出選擇的對話框。（這種情況下用戶不能選擇默認啓動的app）。



為了顯示chooser, 需要使用[createChooser\(\)](#)來創建Intent

```
Intent intent = new Intent(Intent.ACTION_SEND);
...
// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show chooser
Intent chooser = Intent.createChooser(intent, title);

// Verify the intent will resolve to at least one activity
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

這樣就列出了可以響應 `createChooser()` 中Intent的app，並且指定了標題。

# 接收Activity返回的結果

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/intents/result.html>

啓動另外一個activity並不一定是單向的。你也可以啓動另外一個activity然後接收一個result回來。為了接收這個result，你需要使用[startActivityForResult\(\)](#)，而不是[startActivity\(\)](#)。

例如，你的app可以啓動一個camera程序並接收拍到的照片作為result。或者你可以啓動People程序並獲取其中聯繫人的詳情作為result。

當然，被啓動的activity需要指定返回的result。它需要把這個result作為另外一個intent對象返回，你的activity需要在[onActivityResult\(\)](#)的回調方法裏面去接收result。

Note: 在執行 `startActivityForResult()` 時，你可以使用explicit 或者 implicit 的intent。當你啓動另外一個位於你的程序中的activity時，你應該使用explicit intent來確保你可以接收到期待的結果。

## 啓動Activity

對於[startActivityForResult\(\)](#)方法中的intent與之前介紹的並沒有什麼差異，只不過是需要在這個方法裏面多添加一個int類型的參數。

這個integer的參數叫做"request code"，用於標識你的請求。當你接收到result Intent時，可以從回調方法裏面的參數去判斷這個result是否是你想要的。

例如，下面是一個啓動activity來選擇聯繫人的例子：

```
static final int PICK_CONTACT_REQUEST = 1; // The request code
...
private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, Uri.parse("content://contacts"));
    pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user only contacts w/ phone numbers
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}
```

## 接收Result

當用戶完成了啓動之後activity操作之後，系統會調用你的activity的[onActivityResult\(\)](#)回調方法。這個方法有三個參數：

- 你通過[startActivityForResult\(\)](#)傳遞的request code。
- 第二個activity指定的result code。如果操作成功則是 `RESULT_OK`，如果用戶沒有操作成功，而是直接點擊回退或者其他什麼原因，那麼則是 `RESULT_CANCELED`
- 第三個參數則是intent，它包含了返回的result數據部分。

例如，下面是如何處理pick a contact的result的例子：對應上面的例子

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
```

```
// Make sure the request was successful
if (resultCode == RESULT_OK) {
    // The user picked a contact.
    // The Intent's data Uri identifies which contact was selected.

    // Do something with the contact here (bigger example below)
}
}
```

在這個例子中被返回的Intent中使用Uri的形式來表示返回的聯繫人。

為了正確的handle這些result，你必須瞭解那些result intent的格式。對於你自己程序裏面的返回result是比較簡單的。Apps都會有一些自己的api來指定特定的數據。例如，People app (Contacts app on some older versions) 總是返回一個URI來指定選擇的contact，Camera app 則是在 data 數據區返回一個 Bitmap （詳細介紹請見 [Capturing Photos](#)）。

## 讀取聯繫人數據

上面的代碼展示了如何獲取聯繫人的返回結果，但沒有說清楚如何從結果中讀取數據。因為這需要更多關於content providers的知識。但如果你想知道的話，下面是一段代碼，展示如何從被選的聯繫人中讀出電話號碼。

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request it is that we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // Get the URI that points to the selected contact
            Uri contactUri = data.getData();
            // We only need the NUMBER column, because there will be only one row in the result
            String[] projection = {Phone.NUMBER};

            // Perform the query on the contact to get the NUMBER column
            // We don't need a selection or sort order (there's only one result for the given URI)
            // CAUTION: The query() method should be called from a separate thread to avoid blocking
            // your app's UI thread. (For simplicity of the sample, this code doesn't do that.)
            // Consider using CursorLoader to perform the query.
            Cursor cursor = getContentResolver()
                .query(contactUri, projection, null, null, null);
            cursor.moveToFirst();

            // Retrieve the phone number from the NUMBER column
            int column = cursor.getColumnIndex(Phone.NUMBER);
            String number = cursor.getString(column);

            // Do something with the phone number...
        }
    }
}
```

Note: 在Android 2.3 (API level 9)之前對 contacts Provider 的請求(比如上面的代碼)，需要你聲明 READ\_CONTACTS 權限(更多詳見 [Security and Permissions](#))。但如果是Android 2.3以上的系統，通信錄或者聯繫人App返回一個result的時候，這些App會給予你的App讀取Contacts Provider的臨時權限。但這種臨時權限也僅限於特定的請求，所以你仍無法獲取除返回的Intent以外的聯繫人信息，除非聲明瞭 READ\_CONTACTS 權限。

# Intent過濾

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/intents/filters.html>

前兩節課主要講了從你的app啓動另外一個app。但如果你的app的功能對別的app也有用，那麼你的app應該做好響應的準備。例如，如果你創建了一個social app，它可以分享messages 或者 photos 給好友，那麼最好你的app能夠接收 ACTION\_SEND 的intent，這樣當用戶在其他app觸發分享功能的時候，你的app能夠出現在待選對話框。

為了使得其他的app能夠啓動你的activity，你需要在你的manifest文件的 `<activity>` 標簽下添加 `<intent-filter>` 的屬性。

當你的app被安裝到設備上時，系統可以識別你的intent filter並把這些信息記錄下來。當其他app使用implicit intent執行 `startActivity()` 或者 `startActivityForResult()` 時，系統會自動查找出那些可以響應這個intent的activity。

## 添加Intent Filter

為了儘可能確切的定義你的activity能夠handle哪些intent，每一個intent filter都應該儘可能詳盡的定義好action與data。

如果activity中的intent filter滿足以下intent對象的標準，系統就能夠把特定的intent發送給activity：

- Action:一個想要執行的動作的名稱。通常是系統已經定義好的值，例如 ACTION\_SEND 或者 ACTION\_VIEW。在intent filter中用 `<action>` 指定它的值，值的類型必須為字符串，而不是API中的常量(看下面的例子)
- Data:Intent附帶數據的描述。在intent filter中用 `<data>` 指定它的值，可以使用一個或者多個屬性，你可以只定義 MIME type或者是隻指定URI prefix，也可以只定義一個URI scheme，或者是他們綜合使用。

Note: 如果你不想handle Uri 類型的數據，那麼你應該指定 android:mimeType 屬性。例如 text/plain 或者 image/jpeg.

- Category:提供一個附加的方法來標識這個activity能夠handle的intent。通常與用戶的手勢或者是啓動位置有關。系統有支持幾種不同的categories,但是大多數都不怎麼用的到。而且，所有的implicit intents都默認是 CATEGORY\_DEFAULT 類型的。在intent filter中用 `<category>` 指定它的值。

在你的intent filter中，你可以在 `<intent-filter>` 元素中定義對應的XML元素來聲明你的activity使用何種標準。

例如，這個有intent filter的activity，當數據類型為文本或圖像時會處理 ACTION\_SEND 的intent。

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

每一個發送出來的intent只會包含一個action與data類型，但是handle這個intent的activity的 `<intent-filter>` 是可以聲明多個 `<action>`, `<category>` 與 `<data>` 的。

如果任何的兩對action與data是互相矛盾的，你應該創建不同的intent filter來指定特定的action與type。

例如，假設你的activity可以handle 文本與圖片，無論是 ACTION\_SEND 還是 ACTION\_SENDTO 的intent。在這種情況下，你必須為兩個action定義兩個不同的intent filter。因為 ACTION\_SENDTO intent 必須使用 Uri 類型來指定接收者使用 send 或 sendto 的地址。例如：

```
<activity android:name="ShareActivity">
    <!-- filter for sending text; accepts SENDTO action with sms URI schemes -->
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="sms" />
        <data android:scheme="smsto" />
    </intent-filter>
    <!-- filter for sending text or images; accepts SEND action and text or image data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Note: 為了接受implicit intents，你必須在你的intent filter中包含 CATEGORY\_DEFAULT 的 category。startActivity()和startActivityForResult()方法將所有intent視為聲明CATEGORY\_DEFAULT category。如果你沒有在你的intent filter中聲明CATEGORY\_DEFAULT，你的activity將無法對implicit intent 做出響應。

關於更多sending 與 receiving ACTION\_SEND intents來執行social sharing行為的，請查看[接收從其他App傳送來的數據](#)。

## 在你的Activity中Handle發送過來的Intent

為了決定採用哪個action，你可以讀取Intent的內容。

你可以執行[getIntent\(\)](#) 來獲取啓動你的activity的那個intent。你可以在activity生命週期的任何時候去執行這個方法，但最好是在 `onCreate()` 或者 `onStart()` 裏面去執行。

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

    setContentView(R.layout.main);

    // Get the intent that started this activity
    Intent intent = getIntent();
    Uri data = intent.getData();

    // Figure out what to do based on the intent type
    if (intent.getType().indexOf("image/") != -1) {
        // Handle intents with image data ...
    } else if (intent.getType().equals("text/plain")) {
        // Handle intents with text ...
    }
}
```

## 返回Result

如果你想返回一個result給啓動你的那個activity，簡單地調用[setResult\(\)](#)來指定result code與result intent。當你的操

作完成之後，用戶需要返回到原來的activity，通過執行`finish()`來關閉被叫起的activity。例如：

```
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION", Uri.parse("content://result_uri"));
setResult(Activity.RESULT_OK, result);
finish();
```

你必須總是指定一個result code。通常不是`RESULT_OK`就是`RESULT_CANCELED`。你可以通過Intent來添加需要返回的數據。

Note: 默認的result code是`RESULT_CANCELED`。因此，如果用戶在沒有完成操作之前點擊了back按鈕，那麼之前的activity接收到的result code就是"canceled"。

如果你只是純粹想要返回一個int來表示某些返回的result數據之一，你可以設置result code為任何大於0的數值。如果你返回的result只是一個int，那麼連intent都可以不需要返回了，你可以調用 `setResult()`然後只傳遞result code如下：

```
setResult(RESULT_COLOR_RED);
finish();
```

Note:我們沒有必要在意你的activity是被用`startActivity()`還是`startActivityForResult()`方法所叫起的。系統會自動去判斷該如何傳遞result。在不需要的result的case下，result會被自動忽略。

# Android分享操作(Building Apps with Content Sharing)

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/building-content-sharing.html>

這一系列課程會教你如何創建可以在不同的應用與設備之間進行分享的應用。

## 分享簡單的數據(Sharing Simple Data)

學習如何使得你的應用可以和其他應用進行交互。分享信息，接收信息，為用戶數據提供一個簡單並且可擴展的方式來執行分享操作。

## 分享文件(Sharing Files)

學習使用一個URI與臨時的訪問權限來提供安全的文件訪問。

## 使用NFC分享文件(Sharing Files with NFC)

學習使用NFC功能實現設備間的文件傳遞。

# 分享簡單的數據

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/sharing/index.html>

Android程序中很炫的一個功能是程序之間可以互相通信。為什麼要重新發明一個已經存在於另外一個程序中的功能呢，而且這個功能並非自己程序的核心部分。

這一章節會講述一些通常使用的方法來在不同程序之間通過使用Intent APIs與ActionProvider對象來發送與接受content。

## Lessons

---

- [給其他App發送簡單的數據 - Sending Simple Data to Other Apps](#)

學習如何使用intent發送text與binary數據給其他app。

- [接收從其他App返回的數據 - Receiving Simple Data from Other Apps](#)

學習如何通過Intent在你的app中接收來自其他app的text與binary數據。

- [給ActionBar增加分享功能 - Adding an Easy Share Action](#)

學習如何在你的Acitonbar上添加一個分享功能。

# 給其他App發送簡單的數據

編寫:kesenhoo - 原文:<http://developer.android.com/training/sharing/send.html>

當你構建一個intent，你必須指定這個intent需要觸發的actions。Android定義了一些actions，包括ACTION\_SEND，這個action表明着這個intent是用來從一個activity發送數據到另外一個activity的，甚至是跨進程之間的。

為了發送數據到另外一個activity，你需要做的是指定數據與數據的類型，系統會識別出能夠兼容接受的這些數據的activity。如果這些選擇有多個，則把這些activity顯示給用戶進行選擇；如果只有一個兼容選擇，則立即啓動該Activity。同樣的，你可以在manifest文件的Activity描述中添加接受的數據類型。

在不同的程序之間使用intent來發送與接受數據是在社交分享內容的時候最常用的方法。Intent使得用戶用最常用的程序進行快速簡單的分享信息。

注意:為ActionBar添加分享功能的最好方法是使用ShareActionProvider，它能夠在API level 14以上進行使用。ShareActionProvider會在第3課中進行詳細介紹。

## 分享文本內容(Send Text Content)

ACTION\_SEND最直接與最常用的地方是從一個Activity發送文本內容到另外一個Activity。例如，Android內置的瀏覽器可以把當前顯示頁面的URL作為文本內容分享到其他程序。這是非常有用的，通過郵件或者社交網絡來分享文章或者網址給好友。下面是一段Sample Code:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

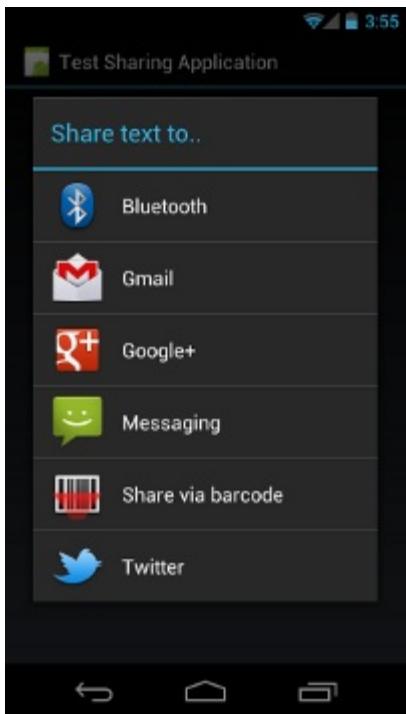
如果設備上有安裝某個能夠匹配ACTION\_SEND與MIME類型為text/plain程序，那麼Android系統會自動把他們都給篩選出來，並呈現Dialog給用戶進行選擇。如果你為intent調用了Intent.createChooser()，那麼Android總是會顯示可供選擇。這樣有一些好處：

- 即使用戶之前為這個intent設置了默認的action，選擇界面還是會被顯示。
- 如果沒有匹配的程序，Android會顯示系統信息。
- 你可以指定選擇界面的標題。

下面是更新後的代碼：

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent, getResources().getText(R.string.send_to)));
```

效果圖如下：



Optionally,你可以為intent設置一些標準的附加值，例如：EXTRA\_EMAIL, EXTRA\_CC, EXTRA\_BCC, EXTRA SUBJECT.然而，如果接收程序沒有針對那些做特殊的處理，則不會有對應的反應。

注意:一些e-mail程序，例如Gmail，對應接收的是EXTRA\_EMAIL與EXTRA\_CC，他們都是String類型的，可以使用putExtra(string,string[])方法來添加到intent裏面。

## 分享二進制內容(Send Binary Content)

分享二進制的數據需要結合設置特定的MIME類型，需要在 EXTRA\_STREAM`裏面放置數據的URI，下面有個分享圖片的例子，這個例子也可以修改用來分享任何類型的二進制數據：

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);
shareIntent.setType("image/jpeg");
startActivity(Intent.createChooser(shareIntent, getResources().getText(R.string.send_to)));
```

請注意下面的內容：

- 你可以使用`/*`這樣的方式來指定MIME類型，但是這僅僅會match到那些能夠處理一般數據類型的Activity(即一般的Activity無法詳盡所有的MIME類型)
- 接收的程序需要有訪問URI資源的權限。下面有一些方法來處理這個問題：
  - 將數據存儲在你的ContentProvider中，確保其他程序有訪問你的provider的權限。較好的提供訪問權限的方法是使用per-URI permissions，其對接收程序而言是只是暫時擁有該許可權限。類似於這樣創建ContentProvider的一種簡單的方法是使用FileProvider helper類。
  - 使用MediaStore系統。MediaStore系統主要用於音視頻及圖片的MIME類型。但在Android3.0之後，其也可以用於存儲非多媒體類型。

## 發送多塊內容(Send Multiple Pieces of Content)

為了同時分享多種不同類型的內容，需要使用 `ACTION_SEND_MULTIPLE` 與指定到那些數據的URLs列表。MIME類型會根據你分享的混合內容而不同。例如，如果你分享3張JPEG的圖片，那麼MIME類型仍然是 `image/jpeg` 。如果是不同圖片格式的話，應該是用 `image/*` 來匹配那些可以接收任何圖片類型的activity。如果你需要分享多種不同類型的數據，可以用 `/*` 來表示MIME。像前面描述的那樣，這取決於那些接收的程序解析並處理你的數據。下面是一個例子：

```
ArrayList<Uri> imageUrils = new ArrayList<Uri>();
imageUrils.add(imageUri1); // Add your image URIs here
imageUrils.add(imageUri2);

Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND_MULTIPLE);
shareIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, imageUrils);
shareIntent.setType("image/*");
startActivity(Intent.createChooser(shareIntent, "Share images to..."));
```

當然，請確保指定到數據的URLs能夠被接收程序所訪問(添加訪問權限)。

# 接收從其他App傳送來的數據

編寫:kesenhoo - 原文:<http://developer.android.com/training/sharing/receive.html>

就像你的程序能夠發送數據到其他程序一樣，其也能夠方便的接收其他程序發來的數據。需要考慮的是用戶與你的程序如何進行交互，以及你想要從其他程序接收哪些數據類型。例如，一個社交網絡程序會希望能夠從其他程序接受文本數據，像一個有趣的網址鏈接。Google+的Android客戶端會接受文本數據與單張或者多張圖片。用這個app，用戶可以簡單的從Gallery程序選擇一張圖片來啓動Google+進行發佈。

## 更新你的manifest文件(Update Your Manifest)

Intent filters通知Android系統一個程序願意接受的數據。像上一課一樣，你可以創建intent filters來表明程序能夠接收哪些action。下面是個例子，對三個activit分別指定接受單張圖片，文本與多張圖片。(這裏有不清楚Intent filter的，請參考[Intents and Intent Filters](#))

```
<activity android:name=".ui.MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

當另外一個程序嘗試通過創建一個intent並將其傳遞給startActivity來分享一些東西的時候，你的程序會被呈現在一個列表裏面讓用戶進行選擇。如果用戶選擇了你的程序，相應的activity就應該被調用開啓，這個時候就是你如何處理獲取到的數據的問題了。

## 處理接受到的數據(Handle the Incoming Content)

為了處理從Intent帶過來的數據，可以通過調用getIntent()方法來獲取到Intent對象。一旦你拿到這個對象，你可以對裏面的數據進行判斷，從而決定下一步應該做什麼。請記住，如果一個activity可以被其他的程序啓動，你需要在檢查intent的時候考慮這種情況(是被其他程序而調用啓動的)。

```
void onCreate (Bundle savedInstanceState) {
    ...
    // Get intent, action and MIME type
    Intent intent = getIntent();
    String action = intent.getAction();
    String type = intent.getType();

    if (Intent.ACTION_SEND.equals(action) && type != null) {
        if ("text/plain".equals(type)) {
            handleSendText(intent); // Handle text being sent
        } else if (type.startsWith("image/")) {
            handleSendImage(intent); // Handle single image being sent
    }
}
```

```
        }
    } else if (Intent.ACTION_SEND_MULTIPLE.equals(action) && type != null) {
        if (type.startsWith("image/")) {
            handleSendMultipleImages(intent); // Handle multiple images being sent
        }
    } else {
        // Handle other intents, such as being started from the home screen
    }
    ...
}

void handleSendText(Intent intent) {
    String sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    if (sharedText != null) {
        // Update UI to reflect text being shared
    }
}

void handleSendImage(Intent intent) {
    Uri imageUri = (Uri) intent.getParcelableExtra(Intent.EXTRA_STREAM);
    if (imageUri != null) {
        // Update UI to reflect image being shared
    }
}

void handleSendMultipleImages(Intent intent) {
    ArrayList<Uri> imageUrils = intent.getParcelableArrayListExtra(Intent.EXTRA_STREAM);
    if (imageUrils != null) {
        // Update UI to reflect multiple images being shared
    }
}
```

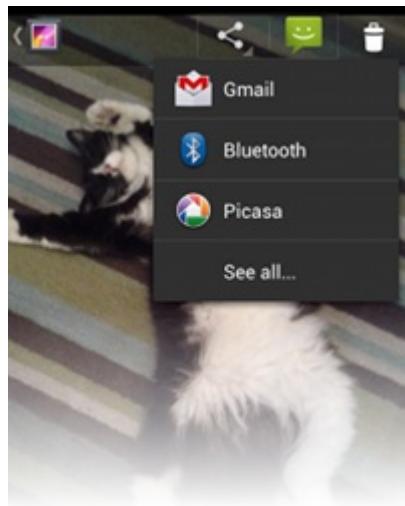
請注意，因為你無法知道其他程序發送過來的數據內容是文本還是其他的數據，因此你需要避免在UI線程裏面去處理那些獲取到的數據。更新UI可以像更新EditText一樣簡單，也可以是更加複雜一點的操作，例如過濾出感興趣的圖片。這完全取決於你的應用接下來要做些什麼。

---

# 添加一個簡便的分享功能

編寫:kesenhoo - 原文:<http://developer.android.com/training/sharing/shareaction.html>

這一課會介紹在ActionBar中添加一個高效率且比較友好的Share功能，會使用到ActionProvider(在Android 4.0上才被引進)。它會handle出現share功能的appearance與behavior。在ShareActionProvider的例子裏面，你只需要提供一個share intent，剩下的就交給ShareActionProvider來做。



## 更新菜單聲明(Update Menu Declarations)

使用ShareActionProvider的第一步，在你的menu resources對應item中定義 android:actionProviderClass 屬性。

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_share"
        android:showAsAction="ifRoom"
        android:title="Share"
        android:actionProviderClass="android.widget.ShareActionProvider" />
    ...
</menu>
```

這表明瞭這個item的appearance與function需要與ShareActionProvider匹配。然而，你還是需要告訴provider你想分享的內容。

## Set the Share Intent(設置分享的intent)

為了能夠實現ShareActionProvider的功能，你必須提供給它一個intent。這個share intent應該像第一課講的那樣，帶有 ACTION\_SEND 和附加數據(例如 EXTRA\_TEXT 與 EXTRA\_STREAM )的。如何使用ShareActionProvider，請看下面的例子：

```
private ShareActionProvider mShareActionProvider;
...
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate menu resource file.
    getMenuInflater().inflate(R.menu.share_menu, menu);
```

```
// Locate MenuItem with ShareActionProvider
MenuItem item = menu.findItem(R.id.menu_item_share);

// Fetch and store ShareActionProvider
mShareActionProvider = (ShareActionProvider) item.getActionProvider();

// Return true to display menu
return true;
}

// Call to update the share intent
private void setShareIntent(Intent shareIntent) {
    if (mShareActionProvider != null) {
        mShareActionProvider.setShareIntent(shareIntent);
    }
}
```

你也許在創建菜單的時候僅僅需要設置一次share intent就滿足需求了，或者說你可能想先設置share intent，然後根據UI的變化來對intent進行更新。例如，當你在Gallery裏面全圖查看照片的時候，share intent會在你切換圖片的時候進行改變。想要查看更多關於ShareActionProvider的內容，請查看[Action Bar](#)。

# 分享文件

---

編寫:jdneo - 原文:<http://developer.android.com/training/secure-file-sharing/index.html>

一個應用程序經常需要向其他應用程序提供一個甚至多個文件。例如，當我們用圖片編輯器編輯圖片時，被編輯的圖片往往由圖庫應用程序所提供；再比如，文件管理器會允許用戶在外部存儲的不同區域之間複製粘貼文件。這裏，我們提出一種讓應用程序可以分享文件的方法：即令發送文件的應用程序對索取文件的應用程序所發出的文件請求進行響應。

在所有情況下，將文件從你的應用程序發送至其它應用程序的唯一的安全方法是向接收文件的應用程序發送這個文件的content URI，並對該URI授予臨時的訪問權限。具有URI臨時訪問權限的content URI是安全的，因為他們僅應用於接收這個URI的應用程序，並且它們會自動過期。Android的FileProvider組件提供了getUriForFile()方法來創建一個文件的content URI。

如果你希望在應用之間僅共享少量的文本或者數字之類的數據，你應該發送一個包含該數據的Intent。要學習如何通過Intent發送簡單數據，可以閱讀：[Sharing Simple Data](#)。

本節課程主要介紹如何使用Android的FileProvider組件所創建的content URI來安全地在應用之間共享文件。當然，要做到這一點，同時還需要授予給接收文件的應用程序訪問這些content URI的臨時訪問權限。

## Lessons

---

- [建立文件分享](#)

學習如何配置你的應用程序使得它們可以分享文件。

- [分享文件](#)

學習分享文件的三個步驟：

- 生成文件的content URI；
- 授予URI的臨時訪問權限；
- 將URI發送給接收文件的應用程序。

- [請求分享一個文件](#)

學習如何向其他應用程序請求文件，如何接收該文件的content URI，以及如何使用content URI打開該文件。

- [獲取文件信息](#)

學習應用程序如何通過FileProvider提供的content URI獲取文件的信息：例如MIME類型，文件大小等。

# 建立文件分享

編寫:jdneo - 原文:<http://developer.android.com/training/secure-file-sharing/setup-sharing.html>

為了將文件安全地從你的應用程序發送給其它應用程序，你需要對你的應用進行配置來提供安全的文件句柄（Content URI的形式）。Android的FileProvider組件會基於你在XML文件中的具體配置，為文件創建Content URI。這節課會向你展示如何在你的應用程序中添加FileProvider的默認實現，以及如何指定你要共享的文件。

Note:FileProvider類隸屬於v4 Support Library庫。關於如何在你的應用程序中包含該庫，可以閱讀：[Support Library Setup](#)。

## 指定FileProvider

為你的應用程序定義一個FileProvider需要在你的Manifest清單文件中定義一個字段，這個字段指明瞭需要使用的創建Content URI的Authority。除此之外，還需要一個XML文件的文件名，這個XML文件指定了你的應用可以共享的目錄路徑。

下面的例子展示了如何在清單文件中添加 `<provider>` 標籤，來指定FileProvider類，Authority和XML文件名：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <application
        ...
        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:grantUriPermissions="true"
            android:exported="false">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/filepaths" />
        </provider>
        ...
    </application>
</manifest>
```

這裏，`android:authorities`屬性字段指定了你希望使用的Authority，該Authority針對於FileProvider所生成的content URI。在這個例子中，這個Authority是“com.example.myapp.fileprovider”。對於你自己的應用，要用你的應用程序包名（`android:package`的值）之後繼續追加“fileprovider”來指定Authority。要學習更多關於Authority的知識，可以閱讀：[Content URLs](#)，以及[android:authorities](#)。

`<provider>` 下的子標籤 `<meta-data>` 指向了一個XML文件，該文件指定了你希望共享的目錄路徑。“`android:resource`” 屬性字段是這個文件的路徑和名字（無“.xml”後綴）。該文件的內容將在下一節討論。

## 指定可共享目錄路徑

一旦你在你的Manifest清單文件中為你的應用添加了FileProvider，你需要指定你希望共享文件的目錄路徑。為了指定這個路徑，我們首先在“res/xml/”下創建文件“filepaths.xml”。在這個文件中，為每一個想要共享目錄添加一個XML標籤。下面的是一個“res/xml/filepaths.xml”的內容樣例。這個例子也說明了如何在你的內部存儲區域共享一個“files/”目錄的子目錄：

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

在這個例子中，`<files-path>` 標籤共享的是在你的應用的內部存儲中“files/”目錄下的目錄。“path”屬性字段指出了該子目錄為“files/”目錄下的子目錄“images/”。 “name”屬性字段告知[FileProvider](#)向在“files/images/”子目錄中的文件的Content URI添加路徑分段（path segment）標記：“myimages”。

`<paths>` 標籤可以有多個子標籤，每一個子標籤用來指定不同的共享目錄。除了`<files-path>` 標籤，你還可以使`<external-path>` 來共享位於外部存儲的目錄；另外，`<cache-path>` 標籤用來共享在你的內部緩存目錄下的子目錄。學習更多關於指定共享目錄的子標籤的知識，可以閱讀：[FileProvider](#)。

Note: XML文件是你定義共享目錄的唯一方式，你不可以用代碼的形式添加目錄。

現在你有一個完整的[FileProvider](#)聲明，它在你應用程序的內部存儲中“files/”目錄下，或者是在“files/”中的子目錄下，創建文件的Content URI。當你的應用為一個文件創建了Content URI，該Content URI將會包含下列信息：

- `<provider>` 標籤中指定的Authority（“com.example.myapp.fileprovider”）；
- 路徑“myimages/”；
- 文件的名字。

例如，如果你根據這節課的例子定義了一個[FileProvider](#)，然後你需要一個文件“default\_image.jpg”的Content URI，[FileProvider](#)會返回如下URI：

```
content://com.example.myapp.fileprovider/myimages/default_image.jpg
```

# 分享文件

編寫:jdneo - 原文:<http://developer.android.com/training/secure-file-sharing/sharing-file.html>

在上一節課中，我們對應用程序進行了配置，使得它可以使用Content URI來共享文件了，現在你可以響應其他應用程序的文件請求。一種響應這些請求的方法是在服務端應用程序提供一個文件選擇接口，它可以由其他應用激活。這種方法可以允許客戶端應用程序讓用戶從服務端應用程序選擇一個文件，然後接收這個文件的Content URI。

這節課將會向你展示如何在你的應用中創建一個用來選擇文件的Activity，來響應這些索取文件的請求。

## 接收文件請求

為了從客戶端應用程序接收一個文件索取請求，然後以Content URI的形式進行響應，你的應用程序應該提供一個選擇文件的Activity。客戶端應用程序通過調用startActivityForResult()來啓動這個Activity。該方法包含了一個Intent參數，它具有ACTION\_PICK這一Action。當客戶端應用程序調用了startActivityForResult()，你的應用可以向客戶端應用程序返回一個結果，該結果即用戶所選擇的文件所對應的Content URI。

學習如何在客戶端應用程序實現文件索取請求，可以閱讀：[請求分享一個文件](#)。

## 創建一個選擇文件的Activity

為了配置一個選擇文件的Activity，我們首先需要在Manifest清單文件中定義你的Activity，在其Intent過濾器中，匹配ACTION\_PICK這一Action，以及CATEGORY\_DEFAULT和CATEGORY\_OPENABLE這兩種Category。另外，還需要為你的應用程序設置MIME類型過濾器，來表明你的應用程序可以向其他應用程序提供哪種類型的文件。下面的這段代碼展示瞭如何在清單文件中定義新的Activity和Intent過濾器：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    ...
    <application>
        ...
        <activity
            android:name=".FileSelectActivity"
            android:label="File Selector" >
            <intent-filter>
                <action
                    android:name="android.intent.action.PICK"/>
                <category
                    android:name="android.intent.category.DEFAULT"/>
                <category
                    android:name="android.intent.category.OPENABLE"/>
                <data android:mimeType="text/plain"/>
                <data android:mimeType="image/*"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## 在代碼中定義文件選擇Activity

下面，定義一個Activity子類，它用來顯示在你內部存儲的“files/images/”目錄下可以獲得的文件，然後允許用戶選擇期望的文件。下面的代碼展示瞭如何定義這個Activity，並令其響應用戶的選擇：

```
public class MainActivity extends Activity {
    // The path to the root of this app's internal storage
```

```

private File mPrivateRootDir;
// The path to the "images" subdirectory
private File mImagesDir;
// Array of files in the images subdirectory
File[] mImageFiles;
// Array of filenames corresponding to mImageFiles
String[] mImageFilenames;
// Initialize the Activity
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Set up an Intent to send back to apps that request a file
    mResultIntent =
        new Intent("com.example.myapp.ACTION_RETURN_FILE");
    // Get the files/ subdirectory of internal storage
    mPrivateRootDir = getFilesDir();
    // Get the files/images subdirectory;
    mImagesDir = new File(mPrivateRootDir, "images");
    // Get the files in the images subdirectory
    mImageFiles = mImagesDir.listFiles();
    // Set the Activity's result to null to begin with
    setResult(Activity.RESULT_CANCELED, null);
    /*
     * Display the file names in the ListView mFileListView.
     * Back the ListView with the array mImageFilenames, which
     * you can create by iterating through mImageFiles and
     * calling File.getAbsolutePath() for each File
     */
    ...
}
...
}

```

## 響應一個文件選擇

一旦用戶選擇了一個想要共享的文件，你的應用程序必須明確哪個文件被選擇了，然後為這個文件生成一個對應的Content URI。如果我們的Activity在ListView中顯示了可獲得文件的清單，那麼當用戶點擊了一個文件名時，系統會調用方法onItemClick()，在該方法中你可以獲取被選擇的文件。

在onItemClick()中，根據被選中文件的文件名獲取一個File對象，然後將它作為參數傳遞給getUriFromFile()，另外還需傳入的參數是你在<provider>標籤中為FileProvider所指定的Authority，函數返回的Content URI包含了相應的Authority，一個對應於文件目錄的路徑標記（如在XML meta-data中定義的），以及包含擴展名的文件名。有關FileProvider如何基於XML meta-data將目錄路徑與路徑標記進行匹配的知識，可以閱讀：[指定可共享目錄路徑](#)。

下面的例子展示瞭如何檢測選中的文件並且獲得它的Content URI：

```

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in the ListView
    mFileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            /*
             * When a filename in the ListView is clicked, get its
             * content URI and send it to the requesting app
             */
            public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
                /*
                 * Get a File for the selected file name.
                 * Assume that the file names are in the
                 * mImageFilename array.
                 */
            }
        }
    );
}

```

```

        File requestFile = new File(mImageFilename[position]);
        /*
         * Most file-related method calls need to be in
         * try-catch blocks.
         */
        // Use the FileProvider to get a content URI
        try {
            fileUri = FileProvider.getUriForFile(
                MainActivity.this,
                "com.example.myapp.fileprovider",
                requestFile);
        } catch (IllegalArgumentException e) {
            Log.e("File Selector",
                  "The selected file can't be shared: " +
                  clickedFilename);
        }
        ...
    });
    ...
}

```

記住，你能生成的那些Content URI所對應的文件，是那些在meta-data文件中包含 `<paths>` 標籤的（即你定義的）目錄內的文件，這方面知識在[Specify Sharable Directories](#)中已經討論過。如果你調用[getUriForFile\(\)](#)方法所要獲取的文件不在你指定的目錄中，你會收到一個[IllegalArgumentException](#)。

## 爲文件授權

現在已經有了你想要共享給其他應用程序的文件所對應的Content URI，你需要允許客戶端應用程序訪問這個文件。爲了達到這一目的，可以通過將Content URI添加至一個[Intent](#)中，然後爲該[Intent](#)設置權限標記。你所授予的權限是臨時的，並且當接收文件的應用程序的任務棧終止後，會自動過期。

下面的例子展示瞭如何爲文件設置讀權限：

```

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks in the ListView
    mFileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
                ...
                if (fileUri != null) {
                    // Grant temporary read permission to the content URI
                    mResultIntent.addFlags(
                        Intent.FLAG_GRANT_READ_URI_PERMISSION);
                }
                ...
            }
        });
    ...
}

```

**Caution :** 只有調用[setFlags\(\)](#)來爲你的文件授予臨時被訪問權限纔是唯一的方法。儘量避免對文件的Content URI調用[Context.grantUriPermission\(\)](#)，因爲通過該方法授予的權限，你只能通過調用[Context.revokeUriPermission\(\)](#)來撤銷。

## 與請求應用共享文件

為了向請求文件的應用程序提供其需要的文件，我們將包含了Content URI和相應權限的Intent傳遞給 setResult()。當你定義的Activity結束後，系統會把這個包含了Content URI的Intent傳遞給客戶端應用程序。下面的例子展示了其中的核心步驟：

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Define a listener that responds to clicks on a file in the ListView
    mList.setAdapter(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView,
                View view,
                int position,
                long rowId) {
            ...
            if (fileUri != null) {
                ...
                // Put the Uri and MIME type in the result Intent
                mResultIntent.setDataAndType(
                    fileUri,
                    getContentResolver().getType(fileUri));
                // Set the result
                MainActivity.this.setResult(Activity.RESULT_OK,
                    mResultIntent);
            } else {
                mResultIntent.setDataAndType(null, "");
                MainActivity.this.setResult(RESULT_CANCELED,
                    mResultIntent);
            }
        }
    });
}
```

當用戶選擇好文件後，我們應該向用戶提供一個能夠立即回到客戶端應用程序的方法。一種實現的方法是向用戶提供一個勾選框或者一個完成按鈕。可以使用按鈕的android:onClick屬性字段為它關聯一個方法。在該方法中，調用finish()。例如：

```
public void onDoneClick(View v) {
    // Associate a method with the Done button
    finish();
}
```

# 請求分享一個文件

編寫:jdneo - 原文:<http://developer.android.com/training/secure-file-sharing/request-file.html>

當一個應用程序希望訪問由其它應用程序所共享的文件時，請求應用程序（即客戶端）經常會向其它應用程序（服務端）發送一個文件請求。在大多數情況下，這個請求會導致在服務端應用程序中啓動一個Activity，在該Activity中會顯示可以共享的文件。當服務端應用程序向客戶端應用程序返回了文件的Content URI後，用戶即選擇了文件。

這節課將向你展示一個客戶端應用程序應該如何向服務端應用程序請求一個文件，接收服務端應用程序發來的Content URI，然後使用這個Content URI打開這個文件。

## 發送一個文件請求

為了向服務端應用程序發送文件請求，在客戶端應用程序中，需要調用startActivityForResult()方法，同時傳遞給這個方法一個Intent參數，它包含了客戶端應用程序能處理的某個Action，比如ACTION\_PICK；以及一個MIME類型。

例如，下面的代碼展示瞭如何向服務端應用程序發送一個Intent，來啓動在分享文件中提到的Activity：

```
public class MainActivity extends Activity {
    private Intent mRequestFileIntent;
    private ParcelFileDescriptor mInputPFD;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mRequestFileIntent = new Intent(Intent.ACTION_PICK);
        mRequestFileIntent.setType("image/jpg");
        ...
    }
    ...
    protected void requestFile() {
        /**
         * When the user requests a file, send an Intent to the
         * server app.
         * files.
         */
        startActivityForResult(mRequestFileIntent, 0);
        ...
    }
    ...
}
```

## 訪問請求的文件

當服務端應用程序向客戶端應用程序發回包含Content URI的Intent時，這個Intent會傳遞給客戶端應用程序重寫的onActivityResult()方法當中。一旦客戶端應用程序有了文件的Content URI，它就可以通過獲取其FileDescriptor來訪問文件了。

文件的安全問題在這一過程中不用過多擔心，因為客戶端應用程序所收到的所有數據只有文件的Content URI而已。由於URI不包含目錄路徑信息，客戶端應用程序無法查詢出或者打開任何服務端應用程序的其他文件。客戶端應用程序僅僅獲取了這個文件的訪問渠道以及由服務端應用程序授予的訪問權限。同時訪問權限是臨時的，一旦這個客戶端應用的任務棧結束了，這個文件將不再被除服務端應用程序之外的其他應用程序訪問。

下面的例子展示了客戶端應用程序應該如何處理發自服務端應用程序的Intent，以及客戶端應用程序如何使用Content URI獲取FileDescriptor：

```
/*
 * When the Activity of the app that hosts files sets a result and calls
 * finish(), this method is invoked. The returned Intent contains the
 * content URI of a selected file. The result code indicates if the
 * selection worked or not.
 */
@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent returnIntent) {
    // If the selection didn't work
    if (resultCode != RESULT_OK) {
        // Exit without doing anything else
        return;
    } else {
        // Get the file's content URI from the incoming Intent
        Uri returnUri = returnIntent.getData();
        /*
         * Try to open the file for "read" access using the
         * returned URI. If the file isn't found, write to the
         * error log and return.
         */
        try {
            /*
             * Get the content resolver instance for this context, and use it
             * to get a ParcelFileDescriptor for the file.
             */
            mInputPFD = getContentResolver().openFileDescriptor(returnUri, "r");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Log.e("MainActivity", "File not found.");
            return;
        }
        // Get a regular file descriptor for the file
        FileDescriptor fd = mInputPFD.getFileDescriptor();
        ...
    }
}
```

`openFileDescriptor()`方法會返回一個文件的`ParcelFileDescriptor`對象。從這個對象中，客戶端應用程序可以獲取`FileDescriptor`對象，然後用戶就可以利用這個對象讀取這個文件了。

# 獲取文件信息

編寫:jdneo - 原文:<http://developer.android.com/training/secure-file-sharing/retrieve-info.html>

當一個客戶端應用程序擁有了文件的Content URI之後，它就可以獲取該文件並進行下一步的工作了，但是在此之前，客戶端應用程序還可以向服務端應用程序索取關於文件的信息，包括文件的數據類型和文件大小等等。數據類型可以幫助客戶端應用程序確定自己能否處理該文件，文件大小能幫助客戶端應用程序為文件設置合理的緩衝區。

這節課將展示如何通過查詢服務端應用程序的FileProvider來獲取文件的MIME類型和文件大小。

## 獲取文件的MIME類型

客戶端應用程序可以通過文件的數據類型判斷自己應該如何處理這個文件的內容。為了得到Content URI所對應的文件數據類型，客戶端應用程序可以調用ContentResolver.getType()方法。這個方法返回了文件的MIME類型。默認情況下，一個FileProvider通過文件的後綴名來確定其MIME類型。

下面的例子展示了當服務端應用程序將Content URI返回給客戶端應用程序後，客戶端應用程序應該如何獲取文件的MIME類型：

```
...
/*
 * Get the file's content URI from the incoming Intent, then
 * get the file's MIME type
 */
Uri returnUri = returnIntent.getData();
String mimeType = getContentResolver().getType(returnUri);
...
```

## 獲取文件名和文件大小

FileProvider類有一個query()方法的默認實現，它返回一個Cursor對象，該Cursor對象包含了Content URI所關聯的文件的名稱和大小。默認的實現返回下面兩列信息：

### DISPLAY\_NAME

文件的文件名，它是一個String類型。這個值和File.getName()所返回的值是一樣的。

### SIZE

文件的大小，以字節為單位，它是一個long類型。這個值和File.length()所返回的值是一樣的。

客戶端應用可以通過將query()的除了Content URI之外的其他參數都設置為“null”，來同時獲取文件的名稱（DISPLAY\_NAME）和大小（SIZE）。例如，下面的代碼獲取一個文件的名稱和大小，然後在兩個TextView中將他們顯示出來：

```
...
/*
 * Get the file's content URI from the incoming Intent,
 * then query the server app to get the file's display name
 * and size.
*/
```

```
Uri returnUri = returnIntent.getData();
Cursor returnCursor =
        getContentResolver().query(returnUri, null, null, null, null);
/*
 * Get the column indexes of the data in the Cursor,
 * move to the first row in the Cursor, get the data,
 * and display it.
 */
int nameIndex = returnCursor.getColumnIndex(OpenableColumns.DISPLAY_NAME);
int sizeIndex = returnCursor.getColumnIndex(OpenableColumns.SIZE);
returnCursor.moveToFirst();
TextView nameView = (TextView) findViewById(R.id.filename_text);
TextView sizeView = (TextView) findViewById(R.id.filesize_text);
nameView.setText(returnCursor.getString(nameIndex));
sizeView.setText(Long.toString(returnCursor.getLong(sizeIndex)));
...
...
```

# 使用NFC分享文件

---

編寫:jdneo - 原文:<http://developer.android.com/training/beam-files/index.html>

Android允許你通過Android Beam文件傳輸功能在設備之間傳送大文件。這個功能具有簡單的API，同時，它允許用戶僅需要進行一些簡單的觸控操作就能啓動文件傳輸過程。Android Beam會自動地將文件從一臺設備拷貝至另一臺設備中，並在完成時告知用戶。

Android Beam文件傳輸API可以用來處理規模較大的數據，而在Android4.0（API Level 14）引入的Android BeamNDEF傳輸API則用來處理規模較小的數據，比如：URI或者消息數據等。另外，Android Beam僅僅是Android NFC框架提供的衆多特性之一，它允許你從NFC標籤中讀取NDEF消息。想要學習更多有關Android Beam的知識，可以閱讀：[Beaming NDEF Messages to Other Devices](#)。想要學習更多有關NFC框架的知識，可以閱讀：[Near Field Communication](#)。

## Lessons

---

- [發送文件給其他設備](#)

學習如何配置你的應用程序，使其可以發送文件給其他設備。

- [接收其他設備的文件](#)

學習如何配置你的應用程序，使其可以接收其他設備發送的文件。

# 發送文件給其他設備

編寫:jdneo - 原文:<http://developer.android.com/training/beam-files/sending-files.html>

這節課將展示如何通過Android Beam文件傳輸向另一臺設備發送大文件。要發送文件，首先需要聲明使用NFC和外部存儲的權限，你需要測試一下你的設備是否支持NFC，這樣，你才能夠將文件的URI提供給Android Beam文件傳輸。

使用Android Beam文件傳輸功能必須滿足下列要求：

1. 用Android Beam文件傳輸功能傳輸大文件只能在Android 4.1（API Level 16）及以上版本的Android系統中使用。
2. 你希望傳送的文件必須放置於外部存儲。學習更多關於外部存儲的知識，可以閱讀：[Using the External Storage](#)。
3. 每個你希望傳送的文件必須是全局可讀的。你可以通過[File.setReadable\(true, false\)](#)來為文件設置相應的讀權限。
4. 你必須提供待傳輸文件的File URI。Android Beam文件傳輸無法處理由[FileProvider.getUriForFile](#)生成的Content URI。

## 在清單文件中聲明

首先，編輯你的Manifest清單文件來聲明你的應用程序所需要的權限和功能。

### 聲明權限

為了允許你的應用程序使用Android Beam文件傳輸控制NFC從外部存儲發送文件，你必須在應用程序的Manifest清單文件中聲明下面的權限：

#### NFC

允許你的應用程序通過NFC發送數據。為了聲明該權限，添加下面的標籤作為一個 `<manifest>` 標籤的子標籤：

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

#### READ\_EXTERNAL\_STORAGE

允許你的應用讀取外部存儲。為了聲明該權限，添加下面的標籤作為一個 `<manifest>` 標籤的子標籤：

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

Note：對於Android 4.2.2（API Level 17）及之前版本的系統，這個權限不是必需的。在後續版本的系統中，若應用程序需要讀取外部存儲，可能會需要申明該權限。為了保證將來程序穩定性，建議在該權限申明變成必需的之前，就在清單文件中聲明好。

### 指定NFC功能

指定你的應用程序使用NFC，添加 `<uses-feature>` 標籤作為一個 `<manifest>` 標籤的子標籤。設置 `android:required` 屬性字段為 `true`，這樣可以使得你的應用程序只有在NFC可以使用時，才能運行。

下面的代碼展示瞭如何指定 `<uses-feature>` 標籤：

```
<uses-feature  
    android:name="android.hardware.nfc"  
    android:required="true" />
```

注意，如果你的應用程序將NFC作為一個可選的功能，但期望在NFC不可使用時程序還能繼續執行，你就應該設置 `android:required` 屬性字段為 `false`，然後在代碼中測試NFC的可用性。

## 指定Android Beam文件傳輸

由於Android Beam文件傳輸只能在Android 4.1（API Level 16）及以上的平臺使用，如果你的應用將Android Beam文件傳輸作為一個不可缺少的核心模塊，那麼你必須指定 `<uses-sdk>` 標籤為：`android:minSdkVersion="16"`。或者，你可以將`android:minSdkVersion`設置為其它值，然後在代碼中測試平臺版本，這部分內容將在下一節中展開。

## 測試設備是否支持Android Beam文件傳輸

為了在Manifest清單文件中，指定NFC是可選的，你應該使用下面的標籤：

```
<uses-feature android:name="android.hardware.nfc" android:required="false" />
```

如果設置了`android:required="false"`，那麼你必須要在代碼中測試設備是否支持NFC和Android Beam文件傳輸。

為了在代碼中測試Android Beam文件傳輸，我們先通過`PackageManager.hasSystemFeature()`和參數`FEATURE_NFC`，來測試設備是否支持NFC。下一步，通過`SDK_INT`的值測試系統版本是否支持Android Beam文件傳輸。如果設備支持Android Beam文件傳輸，那麼獲得一個NFC控制器的實例，它能允許你與NFC硬件進行通信，如下所示：

```
public class MainActivity extends Activity {  
    ...  
    NfcAdapter mNfcAdapter;  
    // Flag to indicate that Android Beam is available  
    boolean mAndroidBeamAvailable = false;  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        ...  
        // NFC isn't available on the device  
        if (!PackageManager.hasSystemFeature(PackageManager.FEATURE_NFC)) {  
            /*  
             * Disable NFC features here.  
             * For example, disable menu items or buttons that activate  
             * NFC-related features  
             */  
            ...  
            // Android Beam file transfer isn't supported  
        } else if (Build.VERSION.SDK_INT <  
                  Build.VERSION_CODES.JELLY_BEAN_MR1) {  
            // If Android Beam isn't available, don't continue.  
            mAndroidBeamAvailable = false;  
            /*  
             * Disable Android Beam file transfer features here.  
             */  
            ...  
            // Android Beam file transfer is available, continue  
        } else {  
            mNfcAdapter = NfcAdapter.getDefaultAdapter(this);  
            ...
```

```
        }
    }
    ...
}
```

## 創建一個提供文件的回調函數

一旦你確認了設備支持Android Beam文件傳輸，那麼可以添加一個回調函數，當Android Beam文件傳輸監測到用戶希望向另一個支持NFC的設備發送文件時，系統會調用它。在該回調函數中，返回一個Uri對象數組，Android Beam文件傳輸將URI對應的文件拷貝給要接收這些文件的設備。

要添加這個回調函數，我們需要實現NfcAdapter.CreateBeamUrisCallback接口，和它的方法：createBeamUris()，下面是一個例子：

```
public class MainActivity extends Activity {
    ...
    // List of URIs to provide to Android Beam
    private Uri[] mFileUris = new Uri[10];
    ...
    /**
     * Callback that Android Beam file transfer calls to get
     * files to share
     */
    private class FileUriCallback implements
        NfcAdapter.CreateBeamUrisCallback {
        public FileUriCallback() {
        }
        /**
         * Create content URIs as needed to share with another device
         */
        @Override
        public Uri[] createBeamUris(NfcEvent event) {
            return mFileUris;
        }
    }
    ...
}
```

一旦你實現了這個接口，通過調用setBeamPushUrisCallback()將回調函數提供給Android Beam文件傳輸。下面是一個例子：

```
public class MainActivity extends Activity {
    ...
    // Instance that returns available files from this app
    private FileUriCallback mFileUriCallback;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Android Beam file transfer is available, continue
        ...
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        /*
         * Instantiate a new FileUriCallback to handle requests for
         * URIs
         */
        mFileUriCallback = new FileUriCallback();
        // Set the dynamic callback for URI requests.
        mNfcAdapter.setBeamPushUrisCallback(mFileUriCallback, this);
        ...
    }
    ...
}
```

Note：你也可以將Uri對象數組通過應用程序的NfcAdapter實例，直接提供給NFC框架。如果你能在NFC觸碰事件發生之前，定義這些URI，那麼你可以選擇使用這個方法。要學習關於這個方法的知識，可以閱讀：[NfcAdapter.setBeamPushUris\(\)](#)。

## 定義要發送的文件

為了將一個或更多個文件發送給其他支持NFC的設備，需要為每一個文件獲取一個File URI（一個具有文件格式（file scheme）的URI），然後將它們添加至一個Uri對象數組中。要傳輸一個文件，你必須也有讀該文件的權限。例如，下面的例子展示的是你如何根據文件名獲取它的File URI，然後將URI添加至數組當中：

```
/*
 * Create a list of URIs, get a File,
 * and set its permissions
 */
private Uri[] mFileUris = new Uri[10];
String transferFile = "transferimage.jpg";
File extDir = getExternalFilesDir(null);
File requestFile = new File(extDir, transferFile);
requestFile.setReadable(true, false);
// Get a URI for the File and add it to the list of URIs
fileUri = Uri.fromFile(requestFile);
if (fileUri != null) {
    mFileUris[0] = fileUri;
} else {
    Log.e("My Activity", "No File URI available for file.");
}
```

# 接收其他設備的文件

編寫:jdneo - 原文:<http://developer.android.com/training/beam-files/receive-files.html>

Android Beam文件傳輸將文件拷貝至接收設備上的一個特殊目錄。同時使用Android Media Scanner掃描拷貝的文件，並在MediaStore provider中為媒體文件添加對應的條目記錄。這節課將向你展示當文件拷貝完成時要如何響應，以及在接收設備上應該如何定位拷貝的文件。

## 響應請求並顯示數據

當Android Beam文件傳輸將文件拷貝至接收設備後，它會發佈一個通知，該通知包含有一個Intent，該Intent擁有所有：ACTION\_VIEW這一Action，首個被傳輸文件的MIME類型，以及一個指向第一個文件的URI。當用戶點擊了這個通知後，Intent會被發送至系統。為了讓你的應用程序能夠響應這個Intent，我們需要為響應的Activity所對應的<activity>標籤添加一個<intent-filter>標籤，在<intent-filter>標籤中，添加下面的子標籤：

```
<action android:name="android.intent.action.VIEW" />
```

該標籤用來匹配從通知發出的Intent，這些Intent具有ACTION\_VIEW這一Action。

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

該標籤用來匹配不含有顯式Category的Intent對象。

```
<data android:mimeType="mime-type" />
```

匹配一個MIME類型。僅僅指定那些你的應用能夠處理的類型。

例如，下面的例子展示了如何添加一個intent filter來激活你的activity：

```
<activity
    android:name="com.example.android.nfctransfer.ViewActivity"
    android:label="Android Beam Viewer" >
    ...
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        ...
    </intent-filter>
</activity>
```

Note：不僅僅只有Android Beam文件傳輸會發送含有ACTION\_VIEW這一Action的Intent。在接收設備上的其它應用也有可能會發送含有該Action的intent。我們馬上會進一步討論這一問題。

## 請求文件讀權限

如果要讀取Android Beam文件傳輸所拷貝到設備上的文件，需要READ\_EXTERNAL\_STORAGE權限。例如：

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

如果你希望將文件拷貝至應用程序自己的存儲區，那麼需要的權限改為WRITE\_EXTERNAL\_STORAGE，另

外`WRITE_EXTERNAL_STORAGE`權限包含了`READ_EXTERNAL_STORAGE`權限。

Note：對於Android 4.2.2（API Level 17）及之前版本的系統，`READ_EXTERNAL_STORAGE`權限僅在用戶選擇要讀文件時纔是強制需要的。而在今後的版本中會在所有情況下都需要該權限。為了保證未來應用程序在的穩定性，建議在Manifest清單文件中聲明該權限。

由於你的應用對於其自身的內部存儲區域具有控制權，所以若要將文件拷貝至應用程序自身的的內部存儲區域，寫權限是不需要聲明的。

## 獲取拷貝文件的目錄

Android Beam文件傳輸一次性將所有文件拷貝到目標設備的一個目錄中，Android Beam文件傳輸通知所發出的Intent中包含有URI，該URI指向了第一個被傳輸的文件。然而，你的應用程序也有可能接收到除了Android Beam文件傳輸之外的某個來源所發出的含有`ACTION_VIEW`這一Action的Intent。為了明確你應該如何處理接收的Intent，你需要檢查它的Scheme和Authority。

可以調用`Uri.getScheme()`獲得URI的Scheme，下面的代碼展示瞭如何確定Scheme並對URI進行相應的處理：

```
public class MainActivity extends Activity {  
    ...  
    // A File object containing the path to the transferred files  
    private File mParentPath;  
    // Incoming Intent  
    private Intent mIntent;  
    ...  
    /*  
     * Called from onNewIntent() for a SINGLE_TOP Activity  
     * or onCreate() for a new Activity. For onNewIntent(),  
     * remember to call setIntent() to store the most  
     * current Intent  
     */  
    private void handleViewIntent() {  
        ...  
        // Get the Intent action  
        mIntent = getIntent();  
        String action = mIntent.getAction();  
        /*  
         * For ACTION_VIEW, the Activity is being asked to display data.  
         * Get the URI.  
         */  
        if (TextUtils.equals(action, Intent.ACTION_VIEW)) {  
            // Get the URI from the Intent  
            Uri beamUri = mIntent.getData();  
            /*  
             * Test for the type of URI, by getting its scheme value  
             */  
            if (TextUtils.equals(beamUri.getScheme(), "file")) {  
                mParentPath = handleFileUri(beamUri);  
            } else if (TextUtils.equals(  
                beamUri.getScheme(), "content")) {  
                mParentPath = handleContentUri(beamUri);  
            }  
        }  
        ...  
    }  
    ...  
}
```

## 從File URI中獲取目錄

如果接收的Intent包含一個File URI，則該URI包含了一個文件的絕對文件名，它包括了完整的路徑和文件名。對於

Android Beam文件傳輸來說，目錄路徑指向了其它被傳輸文件的位置（如果有其它傳輸文件的話），要獲得這個目錄路徑，需要取得URI的路徑部分（URI中除去“file:”前綴的部分），根據路徑創建一個File對象，然後獲取這個File的父目錄：

```
...
public String handleFileUri(Uri beamUri) {
    // Get the path part of the URI
    String fileName = beamUri.getPath();
    // Create a File object for this filename
    File copiedFile = new File(fileName);
    // Get a string containing the file's parent directory
    return copiedFile.getParent();
}
...
```

## 從Content URI獲取目錄

如果接收的Intent包含一個Content URI，這個URI可能指向的是存儲於MediaStore Content Provider的目錄和文件名。你可以通過檢測URI的Authority值來判斷它是否是來自於MediaStore的Content URI。一個MediaStore的Content URI可能來自Android Beam文件傳輸也可能來自其它應用程序，但不管怎麼樣，你都能根據該Content URI獲得一個目錄路徑和文件名。

你也可以接收一個含有ACTION\_VIEW這一Action的Intent，它包含的Content URI針對於Content Provider，而不是MediaStore，在這種情況下，這個Content URI不包含MediaStore的Authority，且這個URI一般不指向一個目錄。

Note：對於Android Beam文件傳輸，接收在含有ACTION\_VIEW的Intent中的Content URI時，如果第一個接收的文件，其MIME類型為“audio/”，“image/”或者“video/\*”，Android Beam文件傳輸會在它存儲傳輸文件的目錄內運行Media Scanner，以此為媒體文件添加索引。同時Media Scanner將結果寫入MediaStore的Content Provider，之後它將第一個文件的Content URI回遞給Android Beam文件傳輸。這個Content URI就是你在通知Intent中所接收到的。要獲得第一個文件的目錄，你需要使用該Content URI從MediaStore中獲取它。

## 確定Content Provider

為了明確你能從Content URI中獲取文件目錄，你可以通過調用Uri.getAuthority()獲取URI的Authority，以此確定與該URI相關聯的Content Provider。其結果有兩個可能的值：

### MediaStore.AUTHORITY

表明這個URI關聯了被MediaStore記錄的一個文件或者多個文件。可以從MediaStore中獲取文件的全名，目錄名就自然可以從文件全名中獲取。

### 其他值

來自其他Content Provider的Content URI。可以顯示與該Content URI相關聯的數據，但是不要嘗試去獲取文件目錄。

要從MediaStore的Content URI中獲取目錄，我們需要執行一個查詢操作，它將Uri參數指定為收到的ContentURI，將MediaColumns.DATA列作為投影（Projection）。返回的Cursor對象包含了URI所代表的文件的完整路徑和文件名。該目錄路徑下還包含了由Android Beam文件傳輸傳送到該設備上的其它文件。

下面的代碼展示了你要如何測試Content URI的Authority，並獲取傳輸文件的路徑和文件名：

```
...
public String handleContentUri(Uri beamUri) {
```

```
// Position of the filename in the query Cursor
int filenameIndex;
// File object for the filename
File copiedFile;
// The filename stored in MediaStore
String fileName;
// Test the authority of the URI
if (!TextUtils.equals(beamUri.getAuthority(), MediaStore.AUTHORITY)) {
    /*
     * Handle content URIs for other content providers
     */
    // For a MediaStore content URI
} else {
    // Get the column that contains the file name
    String[] projection = { MediaStore.MediaColumns.DATA };
    Cursor pathCursor =
        getContentResolver().query(beamUri, projection,
        null, null, null);
    // Check for a valid cursor
    if (pathCursor != null &&
        pathCursor.moveToFirst()) {
        // Get the column index in the Cursor
        filenameIndex = pathCursor.getColumnIndex(
            MediaStore.MediaColumns.DATA);
        // Get the full file name including path
        fileName = pathCursor.getString(filenameIndex);
        // Create a File object for the filename
        copiedFile = new File(fileName);
        // Return the parent directory of the file
        return new File(copiedFile.getParent());
    } else {
        // The query didn't work; return null
        return null;
    }
}
...
}
```

要學習更多關於從Content Provider獲取數據的知識，可以閱讀：[Retrieving Data from the Provider](#)。

# Android多媒體

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/building-multimedia.html>

下面的這些課程會教你如何創建更加符合用戶期待的富媒體的應用。

## 管理音頻播放(Managing Audio Playback)

如何響應音頻硬件按鈕的點擊事件，在播放音頻的時候請求audio focus，以及如何正確的響應audio focus的改變。

## 拍照(Capturing Photos)

如何利用以及存在的相機應用進行拍照，如何直接控制相機硬件實現你自己的相機應用。

## 打印(Printing Content)

如何打印照片，HTML文檔，自定義的文檔。

# 管理音頻播放

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/managing-audio/index.html>

如果我們的應用能夠播放音頻，那麼讓用戶能夠以自己預期的方式控制音頻是很重要的。為了保證良好的用戶體驗，我們應該讓應用能夠管理當前的音頻焦點，因為這樣才能確保多個應用不會在同一時刻一起播放音頻。

在學習本系列課程中，我們將會創建可以對音量按鈕進行響應的應用，該應用會在播放音頻的時候請求獲取音頻焦點，並且在當前音頻焦點被系統或其他應用所改變的時候，做出正確的響應。

## Lessons

---

- [控制音量與音頻播放\(Controlling Your App's Volume and Playback\)](#)

學習如何確保用戶能通過硬件或軟件音量控制器調節應用的音量（通常這些控制器上還具有播放、停止、暫停、跳過以及回放等功能按鍵）。

- [管理音頻焦點\(Managing Audio Focus\)](#)

由於可能會有多個應用具有播放音頻的功能，考慮他們如何交互非常重要。為了防止多個音樂應用同時播放音頻，Android使用音頻焦點（Audio Focus）來控制音頻的播放。在這節課中可以學習如何請求音頻焦點，監聽音頻焦點的丟失，以及在這種情況發生時應該如何做出響應。

- [兼容音頻輸出設備\(Dealing with Audio Output Hardware\)](#)

音頻有多種輸出設備，在這節課中可以學習如何找出播放音頻的設備，以及處理播放時耳機被拔出的情況。

# 控制音量與音頻播放

編寫:kesenhoo - 原文:<http://developer.android.com/training/managing-audio/volume-playback.html>

良好的用戶體驗應該是可預期且可控的。如果我們的應用可以播放音頻，那麼顯然我們需要做到能夠通過硬件按鈕，軟件按鈕，藍牙耳麥等來控制音量。同樣地，我們需要能夠對應用的音頻流進行播放（Play），停止（Stop），暫停（Pause），跳過（Skip），以及回放（Previous）等動作，並且並確保其正確性。

## 鑑別使用的是哪個音頻流(Identify Which Audio Stream to Use)

為了創建一個良好的音頻體驗，我們首先需要知道應用會使用到哪些音頻流。Android為播放音樂，鬧鈴，通知鈴，來電聲音，系統聲音，打電話聲音與撥號聲音分別維護了一個獨立的音頻流。這樣做的主要目的是讓用戶能夠單獨地控制不同的種類的音頻。上述音頻種類中，大多數都是被系統限制。例如，除非你的應用需要做替換鬧鐘的鈴聲的操作，不然的話你只能通過STREAM\_MUSIC來播放你的音頻。

## 使用硬件音量鍵來控制應用的音量(Use Hardware Volume Keys to Control Your App's Audio Volume)

默認情況下，按下音量控制鍵會調節當前被激活的音頻流，如果我們的應用當前沒有播放任何聲音，那麼按下音量鍵會調節響鈴的音量。對於遊戲或者音樂播放器而言，即使是在歌曲之間無聲音的狀態，或是當前遊戲處於無聲的狀態，用戶按下音量鍵的操作通常都意味着他們希望調節遊戲或者音樂的音量。你可能希望通過監聽音量鍵被按下的事件，來調節音頻流的音量。其實我們不必這樣做。Android提供了setVolumeControlStream()方法來直接控制指定的音頻流。在鑑別出應用會使用哪個音頻流之後，我們需要在應用生命週期的早期階段調用該方法，因為該方法只需要在Activity整個生命週期中調用一次，通常，我們可以在負責控制多媒體的Activity或者Fragment的onCreate()方法中調用它。這樣能確保不管應用當前是否可見，音頻控制的功能都能符合用戶的預期。

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

自此之後，不管目標Activity或Fragment是否可見，按下設備的音量鍵都能夠影響我們指定的音頻流（在這個例子中，音頻流是"music"）。

## 使用硬件的播放控制按鍵來控制應用的音頻播放(Use Hardware Playback Control Keys to Control Your App's Audio Playback)

許多線控或者無線耳機都會有許多媒體播放控制按鈕，例如：播放，停止，暫停，跳過，以及回放等。無論用戶按下設備上任意一個控制按鈕，系統都會廣播一個帶有ACTION\_MEDIA\_BUTTON的Intent。為了正確地響應這些操作，需要在Manifest文件中註冊一個針對該Action的BroadcastReceiver，如下所示：

```
<receiver android:name=".RemoteControlReceiver">
    <intent-filter>
        <action android:name="android.intent.action.MEDIA_BUTTON" />
    </intent-filter>
</receiver>
```

在Receiver的實現中，需要判斷這個廣播來自於哪一個按鈕，Intent通過EXTRA\_KEY\_EVENT這一Key包含了該信息，另外，KeyEvent類包含了一系列諸如 KEYCODE\_MEDIA\_\* 的靜態變量來表示不同的媒體按鈕，例如KEYCODE\_MEDIA\_PLAY\_PAUSE 與 KEYCODE\_MEDIA\_NEXT。

```
public class RemoteControlReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction())) {
            KeyEvent event = (KeyEvent)intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode()) {
                // Handle key press.
            }
        }
    }
}
```

因為可能會有多個程序在監聽與媒體按鈕相關的事件，所以我們必須在代碼中控制應用接收相關事件的時機。下面的例子顯示瞭如何使用AudioManager來為我們的應用註冊監聽與取消監聽媒體按鈕事件，當Receiver被註冊上時，它將是唯一一個能夠響應媒體按鈕廣播的Receiver。

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...
// Start listening for button presses
am.registerMediaButtonEventReceiver(RemoteControlReceiver);
...
// Stop listening for button presses
am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
```

通常，應用需要在他們失去焦點或者不可見的時候（比如在onStop()方法裏面）取消註冊監聽。但是對於媒體播放應用來說並沒有那麼簡單，實際上，在應用不可見（不能通過可見的UI控件進行控制）的時候，仍然能夠響應媒體播放按鈕事件是極其重要的。為了實現這一點，有一個更好的方法，我們可以在程序獲取與失去音頻焦點的時候註冊與取消對音頻按鈕事件的監聽。這個內容會在後面的課程中詳細講解。

# 管理音頻焦點

編寫:kesenhoo - 原文:<http://developer.android.com/training/managing-audio/audio-focus.html>

由於可能會有多個應用可以播放音頻，所以我們應當考慮一下他們應該如何交互。為了防止多個音樂播放應用同時播放音頻，Android使用音頻焦點（Audio Focus）來控制音頻的播放——即只有獲取到音頻焦點的應用才能夠播放音頻。

在我們的應用開始播放音頻之前，它需要先請求音頻焦點，然後再獲取到音頻焦點。另外，它還需要知道如何監聽失去音頻焦點的事件並對此做出合適的響應。

## 請求獲取音頻焦點(Request the Audio Focus)

在我們的應用開始播放音頻之前，它需要獲取將要使用的音頻流的音頻焦點。通過使用[requestAudioFocus\(\)](#)方法可以獲取我們希望得到的音頻流焦點。如果請求成功，該方法會返回[AUDIOFOCUS\\_REQUEST\\_GRANTED](#)。

另外我們必須指定正在使用的音頻流，而且需要確定所請求的音頻焦點是短暫的（Transient）還是永久的（Permanent）。

- 短暫的焦點鎖定：當計劃播放一個短暫的音頻時使用（比如播放導航指示）。
- 永久的焦點鎖定：當計劃播放一個較長但時長可預期的音頻時使用（比如播放音樂）。

下面的代碼片段是一個在播放音樂時請求永久音頻焦點的例子，我們必須在開始播放之前立即請求音頻焦點，比如在用戶點擊播放或者遊戲中下一關的背景音樂開始前。

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...
// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                                  AudioManager.STREAM_MUSIC,
                                  // Request permanent focus.
                                  AudioManager.AUDIOFOCUS_GAIN);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.registerMediaButtonEventReceiver(RemoteControlReceiver);
    // Start playback.
}
```

一旦結束了播放，需要確保調用了[abandonAudioFocus\(\)](#)方法。這樣相當於告知系統我們不再需要獲取焦點並且註銷所關聯的[AudioManager.OnAudioFocusChangeListener](#)監聽器。對於另一種釋放短暫音頻焦點的情況，這會允許任何被我們打斷的應用可以繼續播放。

```
// Abandon audio focus when playback complete
am.abandonAudioFocus(afChangeListener);
```

當請求短暫音頻焦點的時候，我們可以選擇是否開啓“Ducking”。通常情況下，一個應用在失去音頻焦點時會立即關閉它的播放聲音。如果我們選擇在請求短暫音頻焦點的時候開啓了Ducking，那意味着其它應用可以繼續播放，僅僅是在這一刻降低自己的音量，直到重新獲取到音頻焦點後恢復正常音量（譯註：也就是說，不用理會這個短暫焦點的請求，這並不會打斷目前正在播放的音頻。比如在播放音樂的時候突然出現一個短暫的短信提示聲音，此時僅僅是把歌曲的音量暫時調低，使得用戶能夠聽到短信提示聲，在此之後便立馬恢復正常播放）。

```

// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
        // Use the music stream.
        AudioManager.STREAM_MUSIC,
        // Request permanent focus.
        AudioManager.AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    // Start playback.
}

```

Ducking對於那些間歇性使用音頻焦點的應用來說特別合適，比如語音導航。

如果有另一個應用像上述那樣請求音頻焦點，它所請求的永久音頻焦點或者短暫音頻焦點（支持Ducking或不支持Ducking），都會被你在請求獲取音頻焦點時所註冊的監聽器接收到。

## 處理失去音頻焦點(Handle the Loss of Audio Focus)

如果應用A請求獲取了音頻焦點，那麼在應用B請求獲取音頻焦點的時候，A獲取到的焦點就會失去。如何響應失去焦點事件，取決於失去焦點的方式。

在音頻焦點的監聽器裏面，當接受到描述焦點改變的事件時會觸發[onAudioFocusChange\(\)](#)回調方法。如之前提到的，獲取焦點有三種類型，我們同樣會有三種失去焦點的類型：永久失去，短暫失去，允許Ducking的短暫失去。

- 失去短暫焦點：通常在失去短暫焦點的情況下，我們會暫停當前音頻的播放或者降低音量，同時需要準備在重新獲取到焦點之後恢復播放。
- 失去永久焦點：假設另外一個應用開始播放音樂，那麼我們的應用就應該有效地將自己停止。在實際場景當中，這意味着停止播放，移除媒體按鈕監聽，允許新的音頻播放器可以唯一地監聽那些按鈕事件，並且放棄自己的音頻焦點。此時，如果想要恢復自己的音頻播放，我們需要等待某種特定用戶行為發生（例如按下了我們應用當中的播放按鈕）。

在下面的代碼片段當中，如果焦點的失去是短暫型的，我們將音頻播放對象暫停，並在重新獲取到焦點後進行恢復。如果是永久型的焦點失去事件，那麼我們的媒體按鈕監聽器會被註銷，並且不再監聽音頻焦點的改變。

```

OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT
            // Pause playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Resume playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_LOSS) {
            am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
            am.abandonAudioFocus(afChangeListener);
            // Stop playback
        }
    }
};

```

在上面失去短暫焦點的例子中，如果允許Ducking，那麼除了暫停當前的播放之外，我們還可以選擇使用“Ducking”。

## Duck!

在使用Ducking時，正常播放的歌曲會降低音量來凸顯這個短暫的音頻聲音，這樣既讓這個短暫的聲音比較突出，又不

至於打斷正常的聲音。

下面的代碼片段讓我們的播放器在暫時失去音頻焦點時降低音量，並在重新獲得音頻焦點之後恢復原來音量。

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK) {
            // Lower the volume
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Raise it back to normal
        }
    }
};
```

音頻焦點的失去是我們需要響應的最重要的事件廣播之一，但除此之外還有很多其他重要的廣播需要我們正確地做出響應。系統會廣播一系列的Intent來向你告知用戶在使用音頻過程當中的各種變化。下節課會演示如何監聽這些廣播並提升用戶的整體體驗。

# 兼容音頻輸出設備

編寫:kesenhoo - 原文:<http://developer.android.com/training/managing-audio/audio-output.html>

當用戶想要通過Android設備欣賞音樂的時候，他可以有多種選擇，大多數設備擁有內置的揚聲器，有線耳機，也有其它很多設備支持藍牙連接，有些甚至還支持A2DP藍牙音頻傳輸模型協定。（譯註：A2DP全名是Advanced Audio Distribution Profile 藍牙音頻傳輸模型協定！A2DP是能夠採用耳機內的芯片來堆棧數據，達到聲音的高清晰度。有A2DP的耳機就是藍牙立體聲耳機。聲音能達到44.1kHz，一般的耳機只能達到8kHz。如果手機支持藍牙，只要裝載A2DP協議，就能使用A2DP耳機了。還有消費者看到技術參數提到藍牙V1.0 V1.1 V1.2 V2.0 - 這些是指藍牙的技術版本，是指通過藍牙傳輸的速度，他們是否支持A2DP具體要看藍牙產品製造商是否使用這個技術。來自[百度百科](#)）

## 檢測目前正在使用的硬件設備(Check What Hardware is Being Used)

使用不同的硬件播放聲音會影響到應用的行為。可以使用[AudioManager](#)來查詢當前音頻是輸出到揚聲器，有線耳機還是藍牙上，如下所示：

```
if (isBluetoothA2dpOn()) {  
    // Adjust output for Bluetooth.  
} else if (isSpeakerphoneOn()) {  
    // Adjust output for Speakerphone.  
} else if (isWiredHeadsetOn()) {  
    // Adjust output for headsets  
} else {  
    // If audio plays and noone can hear it, is it still playing?  
}
```

## 處理音頻輸出設備的改變(Handle Changes in the Audio Output Hardware)

當有線耳機被拔出或者藍牙設備斷開連接的時候，音頻流會自動輸出到內置的揚聲器上。假設播放聲音很大，這個時候突然轉到揚聲器播放會顯得非常嘈雜。

幸運的是，系統會在這種情況下廣播帶有[ACTION\\_AUDIO\\_BECOMING\\_NOISY](#)的Intent。無論何時播放音頻，我們都應該註冊一個[BroadcastReceiver](#)來監聽這個Intent。在使用音樂播放器時，用戶通常會希望此時能夠暫停當前歌曲的播放。而在遊戲當中，用戶通常會希望可以減低音量。

```
private class NoisyAudioStreamReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        if (AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intent.getAction())) {  
            // Pause the playback  
        }  
    }  
  
    private IntentFilter intentFilter = new IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);  
  
    private void startPlayback() {  
        registerReceiver(myNoisyAudioStreamReceiver(), intentFilter);  
    }  
  
    private void stopPlayback() {
```

```
    unregisterReceiver(myNoisyAudioStreamReceiver);  
}
```

# 拍照

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/camera/index.html>

在多媒體技術還未流行之時，我們的世界並不像現在這樣多姿多彩。還記得Gopher嗎？（Gopher是計算機上的一個工具軟件，是Internet提供的一種由菜單式驅動的信息查詢工具，採用客戶機/服務器模式）。如果我們希望將我們的應用變成用戶生活的一部分，那麼我們應該給用戶提供一種方式，讓他們可以將自己的生活融入到我們的應用中來。通過相機，我們的應用可以讓用戶擴展他們所看到的事物：生成唯一的頭像，通過相機玩尋找殭屍的交互性遊戲，亦或者是分享他們的某些經歷。

這一章節，我們會學習如何簡單地藉助於已經存在的相機應用，完成一些特定的功能。在後面的課程中，我們還會更加深入地學習如何直接控制相機硬件。

樣例代碼

[PhotointentActivity.zip](#)

## Lessons

---

- [輕鬆拍攝照片](#)

用僅僅幾行代碼調用其他應用拍照。

- [輕鬆錄製視頻](#)

用僅僅幾行代碼調用其他應用錄像。

- [控制相機](#)

直接控制相機硬件，實現你自己的相機應用。

# 輕鬆拍攝照片

編寫:kesenhoo - 原文:<http://developer.android.com/training/camera/photobasics.html>

這節課將講解如何使用已有的相機應用拍攝照片。

假設我們正在實現一個基於人羣的氣象服務，通過應用客戶端拍下的天氣圖片匯聚在一起，可以組成全球氣象圖。整合圖片只是應用的一小部分，我們想要通過最簡單的方式獲取圖片，而不是重新設計並實現一個具有相機功能的組件。幸運的是，通常來說，大多數Android設備都已經安裝了至少一款相機程序。在這節課中，我們會學習如何利用已有的相機應用拍攝照片。

## 請求使用相機權限

如果拍照是應用的必要功能，那麼應該令它在Google Play中僅對有相機的設備可見。為了讓用戶知道我們的應用需要依賴相機，在Manifest清單文件中添加 `<uses-feature>` 標籤：

```
<manifest ...>
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

如果我們的應用使用相機，但相機並不是應用的正常運行所必不可少的組件，可以將 `android:required` 設置為 `"false"`。這樣的話，Google Play 也會允許沒有相機的設備下載該應用。當然我們有必要在使用相機之前通過調用`hasSystemFeature(PackageManager.FEATURE_CAMERA)`方法來檢查設備上是否有相機。如果沒有，我們應該禁用和相機相關的功能！

## 使用相機應用程序進行拍照

利用一個描述了執行目的Intent對象，Android可以將某些執行任務委託給其他應用。整個過程包含三部分：Intent 本身，一個函數調用來啓動外部的 Activity，當焦點返回到我們的Activity時，處理返回圖像數據的代碼。

下面的函數通過發送一個Intent來捕獲照片：

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

注意在調用`startActivityForResult()`方法之前，先調用`resolveActivity()`，這個方法會返回能處理該Intent的第一個Activity（譯註：即檢查有沒有能處理這個Intent的Activity）。執行這個檢查非常重要，因為如果在調用`startActivityForResult()`時，沒有應用能處理你的Intent，應用將會崩潰。所以只要返回結果不為null，使用該Intent就是安全的。

## 獲取縮略圖

拍攝照片並不是應用的最終目的，我們還想要從相機應用那裏取回拍攝的照片，並對它執行某些操作。

Android的相機應用會把拍好的照片編碼為縮小的Bitmap，使用extra value的方式添加到返回的Intent當中，並傳送給onActivityResult()，對應的Key為 "data"。下面的代碼展示的是如何獲取這一圖片並顯示在ImageView上。

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {  
        Bundle extras = data.getExtras();  
        Bitmap imageBitmap = (Bitmap) extras.get("data");  
        mImageView.setImageBitmap(imageBitmap);  
    }  
}
```

Note: 這張從 "data" 中取出的縮略圖適用於作為圖標，但其他作用會比較有限。而處理一張全尺寸圖片需要做更多的工作。

## 保存全尺寸照片

如果我們提供了一個File對象給Android的相機程序，它會保存這張全尺寸照片到給定的路徑下。另外，我們必須提供存儲圖片所需要的含有後綴名形式的文件名。

一般而言，用戶使用設備相機所拍攝的任何照片都應該被存放在設備的公共外部存儲中，這樣它們就能被所有的應用訪問。將DIRECTORY\_PICTURES作為參數，傳遞給getExternalStoragePublicDirectory()方法，可以返回適用於存儲公共圖片的目錄。由於該方法提供的目錄被所有應用共享，因此對該目錄進行讀寫操作分別需要READ\_EXTERNAL\_STORAGE和WRITE\_EXTERNAL\_STORAGE權限。另外，因為寫權限隱含了讀權限，所以如果需要外部存儲的寫權限，那麼僅僅需要請求一項權限就可以了：

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

然而，如果希望照片對我們的應用而言是私有的，那麼可以使用getExternalFilesDir()提供的目錄。在Android 4.3及以下版本的系統中，寫這個目錄需要WRITE\_EXTERNAL\_STORAGE權限。從Android 4.4開始，該目錄將無法被其他應用訪問，所以該權限就不再需要了，你可以通過添加maxSdkVersion屬性，聲明只在低版本的Android設備上請求這個權限。

```
<manifest ...>  
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"  
        android:maxSdkVersion="18" />  
    ...  
</manifest>
```

Note: 所有存儲在getExternalFilesDir()提供的目錄中的文件會在用戶卸載你的app後被刪除。

一旦選定了存儲文件的目錄，我們還需要設計一個保證文件名不會衝突的命名規則。當然我們還可以將路徑存儲在一個成員變量裏以備在將來使用。下面的例子使用日期時間戳作為新照片的文件名：

```
String mCurrentPhotoPath;
```

```

private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
    File image = File.createTempFile(
        imageFileName, /* prefix */
        ".jpg", /* suffix */
        storageDir /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    mCurrentPhotoPath = "file:" + image.getAbsolutePath();
    return image;
}

```

有了上面的方法，我們就可以給新照片創建文件對象了，現在我們可以像這樣創建並觸發一個Intent：

```

static final int REQUEST_TAKE_PHOTO = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            ...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                Uri.fromFile(photoFile));
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO);
        }
    }
}

```

## 將照片添加到相冊中

由於我們通過Intent創建了一張照片，因此圖片的存儲位置我們是知道的。對其他人來說，也許查看我們的照片最簡單的方式是通過系統的Media Provider。

Note: 如果將圖片存儲在`getExternalFilesDir()`提供的目錄中，Media Scanner將無法訪問到我們的文件，因為它們隸屬於應用的私有數據。

下面的例子演示瞭如何觸發系統的Media Scanner，將我們的照片添加到Media Provider的數據庫中，這樣就可以使得Android相冊程序與其他程序能夠讀取到這些照片。

```

private void galleryAddPic() {
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}

```

## 解碼一幅縮放圖片

在有限的內存下，管理許多全尺寸的圖片會很棘手。如果發現應用在展示了少量圖片後消耗了所有內存，我們可以通過縮放圖片到目標視圖尺寸，之後再載入到內存中的方法，來顯著降低內存的使用，下面的例子演示了這個技術：

```
private void setPic() {
    // Get the dimensions of the View
    int targetW = mImageView.getWidth();
    int targetH = mImageView.getHeight();

    // Get the dimensions of the bitmap
    BitmapFactory.Options bmOptions = new BitmapFactory.Options();
    bmOptions.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
    int photow = bmOptions.outWidth;
    int photoH = bmOptions.outHeight;

    // Determine how much to scale down the image
    int scaleFactor = Math.min(photow/targetW, photoH/targetH);

    // Decode the image file into a Bitmap sized to fill the View
    bmOptions.inJustDecodeBounds = false;
    bmOptions.inSampleSize = scaleFactor;
    bmOptions.inPurgeable = true;

    Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
    mImageView.setImageBitmap(bitmap);
}
```

# 輕鬆錄製視頻

編寫:kesenhoo - 原文:<http://developer.android.com/training/camera/videobasics.html>

這節課會介紹如何使用已有的相機應用來錄製視頻。

假設在我們應用的所有功能當中，整合視頻只是其中的一小部分，我們想要以最簡單的方法錄製視頻，而不是重新實現一個攝像機組件。幸運的是，大多數Android設備已經安裝了一個能錄製視頻的相機應用。在本節課當中，我們將會讓它為我們完成這一任務。

## 請求相機權限

為了讓用戶知道我們的應用依賴照相機，在Manifest清單文件中添加 `<uses-feature>` 標籤：

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera"
                  android:required="true" />
    ...
</manifest>
```

如果應用使用相機，但相機並不是應用正常運行所必不可少的組件，可以將 `android:required` 設置為 "false"。這樣的話，Google Play 也會允許沒有相機的設備下載該應用。當然我們有必要在使用相機之前通過調用`hasSystemFeature(PackageManager.FEATURE_CAMERA)`方法來檢查設備上是否有相機。如果沒有，那麼和相機相關的功能應該禁用！

## 使用相機程序來錄製視頻

利用一個描述了執行目的的Intent對象，Android可以將某些執行任務委託給其他應用。整個過程包含三部分：Intent本身，一個函數調用來啓動外部的 Activity，當焦點返回到Activity時，處理返回圖像數據的代碼。

下面的函數將會發送一個Intent來錄製視頻

```
static final int REQUEST_VIDEO_CAPTURE = 1;

private void dispatchTakeVideoIntent() {
    Intent takeVideoIntent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    if (takeVideoIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takeVideoIntent, REQUEST_VIDEO_CAPTURE);
    }
}
```

注意在調用`startActivityForResult()`方法之前，先調用`resolveActivity()`，這個方法會返回能處理該Intent的第一個Activity（譯註：即檢查有沒有能處理這個Intent的Activity）。執行這個檢查非常重要，因為如果在調用`startActivityForResult()`時，沒有應用能處理你的Intent，應用將會崩潰。所以只要返回結果不為null，使用該Intent就是安全的。

## 查看視頻

Android的相機程序會把指向視頻存儲地址的Uri添加到Intent中，並傳送給onActivityResult()方法。下面的代碼獲取該視頻並顯示到一個VideoView當中：

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (requestCode == REQUEST_VIDEO_CAPTURE && resultCode == RESULT_OK) {  
        Uri videoUri = intent.getData();  
        mVideoView.setVideoURI(videoUri);  
    }  
}
```

# 控制相機

編寫:kesenhoo - :<http://developer.android.com/training/camera/cameradirect.html>

在這一節課，我們會討論如何通過使用Android框架所提供的API來直接控制相機硬件。

直接控制相機，比起向已有的相機應用請求圖片或視頻，要複雜一些。這節課將會講解如何創建一個特殊的相機應用或將相機整合在我們的應用當中。

## 打開相機對象

獲取一個 Camera 對象是直接控制相機的第一步。正如Android自帶的相機程序一樣，比較好的訪問相機的方式是在onCreate()方法裏面另起一個線程來打開相機。這種辦法可以避免因為啓動時間較長導致UI線程被阻塞。另外還有一種更好的方法：可以把打開相機的操作延遲到onResume()方法裏面去執行，這樣可以使得代碼更容易重用，還能保持控制流程更為簡單。

如果我們在執行Camera.open()方法的時候相機正在被另外一個應用使用，那麼函數會拋出一個Exception，我們可以利用 try 語句塊進行捕獲：

```
private boolean safeCameraOpen(int id) {
    boolean qOpened = false;

    try {
        releaseCameraAndPreview();
        mCamera = Camera.open(id);
        qOpened = (mCamera != null);
    } catch (Exception e) {
        Log.e(getString(R.string.app_name), "failed to open Camera");
        e.printStackTrace();
    }

    return qOpened;
}

private void releaseCameraAndPreview() {
    mPreview.setCamera(null);
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

自從API Level 9開始，相機框架可以支持多個相機。如果使用舊的API，在調用open()時不傳入參數，那麼我們會獲取後置攝像頭。

## 創建相機預覽界面

拍照通常需要向用戶提供一個預覽界面來顯示待拍攝的事物。我們可以使用SurfaceView來展現照相機採集的圖像。

### Preview類

我們需要使用Preview類來顯示預覽界面。這個類需要實現 android.view.SurfaceHolder.Callback 接口，用這個接口把相機硬件獲取的數據傳遞給應用。

```
class Preview extends ViewGroup implements SurfaceHolder.Callback {  
  
    SurfaceView mSurfaceView;  
    SurfaceHolder mHolder;  
  
    Preview(Context context) {  
        super(context);  
  
        mSurfaceView = new SurfaceView(context);  
        addView(mSurfaceView);  
  
        // Install a SurfaceHolder.Callback so we get notified when the  
        // underlying surface is created and destroyed.  
        mHolder = mSurfaceView.getHolder();  
        mHolder.addCallback(this);  
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);  
    }  
    ...  
}
```

Preview類必須在實時圖像預覽開始之前傳遞給Camera對象。

## 設置和啓動Preview

一個Camera實例與它相關的Preview必須以特定的順序來創建，其中Camera對象首先被創建。在下面的示例中，初始化Camera的動作被封裝了起來，這樣，無論用戶想對Camera做什麼樣的改變，`Camera.startPreview()`都會被`setCamera()`調用。另外，Preview對象必須在`surfaceChanged()`這一回調方法裏面重新啓用（`restart()`）。

```
public void setCamera(Camera camera) {  
    if (mCamera == camera) { return; }  
  
    stopPreviewAndFreeCamera();  
  
    mCamera = camera;  
  
    if (mCamera != null) {  
        List<Size> localSizes = mCamera.getParameters().getSupportedPreviewSizes();  
        mSupportedPreviewSizes = localSizes;  
        requestLayout();  
  
        try {  
            mCamera.setPreviewDisplay(mHolder);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
        // Important: Call startPreview() to start updating the preview  
        // surface. Preview must be started before you can take a picture.  
        mCamera.startPreview();  
    }  
}
```

## 修改相機設置

相機設置可以改變拍照的方式，從縮放級別到曝光補償等。下面的例子僅僅演示瞭如何改變預覽大小，更多設置請參考相機應用的源代碼。

```
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {  
    // Now that the size is known, set up the camera parameters and begin  
    // the preview.  
    Camera.Parameters parameters = mCamera.getParameters();
```

```
parameters.setPreviewSize(mPreviewSize.width, mPreviewSize.height);
requestLayout();
mCamera.setParameters(parameters);

// Important: Call startPreview() to start updating the preview surface.
// Preview must be started before you can take a picture.
mCamera.startPreview();
}
```

## 設置預覽方向

大多數相機程序會鎖定預覽為橫屏狀態，因為該方向是相機傳感器的自然方向。當然這一設定並不會阻止我們去拍豎屏的照片，因為設備的方向信息會被記錄在EXIF頭中。[setCameraDisplayOrientation\(\)](#)方法可以讓你在不影響照片拍攝過程的情況下，改變預覽的方向。然而，對於Android API Level 14及以下版本的系統，在改變方向之前，我們必須先停止預覽，然後再去重啓它。

## 拍攝照片

只要預覽開始之後，可以使用[Camera.takePicture\(\)](#)方法拍攝照片。我們可以創建[Camera.PictureCallback](#)與[Camera.ShutterCallback](#)對象並將他們傳遞到[Camera.takePicture\(\)](#)中。

如果我們想要進行連拍，可以創建一個[Camera.PreviewCallback](#)並實現[onPreviewFrame\(\)](#)方法。我們可以拍攝選中的預覽幀，或是為調用[takePicture\(\)](#)建立一個延遲。

## 重啓Preview

在拍攝好圖片後，我們必須在用戶拍下一張圖片之前重啓預覽。下面的示例使用快門按鈕來實現重啓。

```
@Override
public void onClick(View v) {
    switch(mPreviewState) {
        case K_STATE_FROZEN:
            mCamera.startPreview();
            mPreviewState = K_STATE_PREVIEW;
            break;

        default:
            mCamera.takePicture( null, rawCallback, null );
            mPreviewState = K_STATE_BUSY;
    } // switch
    shutterBtnConfig();
}
```

## 停止預覽並釋放相機

當應用使用好相機後，我們有必要進行清理操作。特別地，我們必須釋放[Camera](#)對象，不然的話可能會引起其他應用崩潰，包括我們自己應用的新實例。

那麼何時應該停止預覽並釋放相機呢？在預覽的Surface被摧毀之後，可以做停止預覽與釋放相機的操作。如下面Preview類中的方法所做的那樣：

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the preview.
```

```
    if (mCamera != null) {
        // Call stopPreview() to stop updating the preview surface.
        mCamera.stopPreview();
    }

    /**
     * When this function returns, mCamera will be null.
     */
    private void stopPreviewAndFreeCamera() {

        if (mCamera != null) {
            // Call stopPreview() to stop updating the preview surface.
            mCamera.stopPreview();

            // Important: Call release() to release the camera for use by other
            // applications. Applications should release the camera immediately
            // during onPause() and re-open() it during onResume().
            mCamera.release();

            mCamera = null;
        }
    }
}
```

在這節課的前部分中，這一些系列的動作也是 `setCamera()` 方法的一部分，因此初始化一個相機的動作，總是從停止預覽開始的。

# 打印

---

編寫:jdneo - 原文:<http://developer.android.com/training/printing/index.html>

Android用戶經常需要在設備上單獨地閱覽信息，但有時候也需要為了分享信息而不得不給其他人看自己設備的屏幕，這顯然不是分享信息的好辦法。如果我們可以通過Android應用把希望分享的信息打印出來，這將給用戶提供一種從應用獲取更多信息的好辦法，更何況這麼做還能將信息分享給其他那些不使用我們的應用的人。另外，打印服務還能創建信息的快照（生成PDF文件），而這一切不需要打印設備，無線網絡連接，也不會消耗過多電量。

在Android 4.4（API Level 19）及更高版本的系統中，框架提供了直接從Android應用程序打印圖片和文字的服務。這系列課程將展示如何啓用打印：包括打印圖片，HTML頁面以及創建自定義的打印文檔。

## Lessons

---

- [打印照片](#)

這節課將展示如何打印一幅圖像。

- [打印HTML文檔](#)

這節課將展示如何打印一個HTML文檔。

- [打印自定義文檔](#)

這節課將展示如何連接到Android打印管理器，創建一個打印適配器並建立要打印的內容。

# 打印照片

編寫:jdneo - 原文:<http://developer.android.com/training/printing/photos.html>

拍攝並分享照片是移動設備最流行的用法之一。如果我們的應用拍攝了照片，並期望可以展示他們，或者允許用戶共享照片，那麼我們就應該考慮讓應用可以打印出這些照片來。Android Support Library提供了一個方便的函數，通過這一函數，僅僅使用很少量的代碼和一些簡單的打印佈局配置集，就能夠進行照片打印。

這堂課將展示如何使用v4 support library中的PrintHelper類打印一幅圖片。

## 打印一幅圖片

Android Support Library中的PrintHelper類提供了一種打印圖片的簡單方法。該類有一個單一的佈局選項：`setScaleMode()`，它允許我們使用下面的兩個選項之一：

- `SCALE_MODE_FIT`：該選項會調整圖像的大小，這樣整個圖像就會在打印有效區域內全部顯示出來（等比例縮放至長和寬都包含在紙張頁面內）。
- `SCALE_MODE_FILL`：該選項同樣會等比例地調整圖像的大小使圖像充滿整個打印有效區域，即讓圖像充滿整個紙張頁面。這就意味着如果選擇這個選項，那麼圖片的一部分（頂部和底部，或者左側和右側）將無法打印出來。如果不設置圖像的打印佈局選項，該模式將是默認的圖像拉伸方式。

這兩個`setScaleMode()`的圖像佈局選項都會保持圖像原有的長寬比。下面的代碼展示瞭如何創建一個PrintHelper類的實例，設置佈局選項，並開始打印進程：

```
private void doPhotoPrint() {
    PrintHelper photoPrinter = new PrintHelper(getActivity());
    photoPrinter.setScaleMode(PrintHelper.SCALE_MODE_FIT);
    Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
        R.drawable.droids);
    photoPrinter.printBitmap("droids.jpg - test print", bitmap);
}
```

該方法可以作為一個菜單項的Action來被調用。注意對於那些不一定被設備支持的菜單項（比如有些設備可能無法支持打印），應該放置在“更多菜單（overflow menu）”中。要獲取有關這方面的更多知識，可以閱讀：[Action Bar](#)。

在`printBitmap()`被調用之後，我們的應用就不再需要進行其他的操作了。之後Android打印界面就會出現，允許用戶選擇一個打印機和它的打印選項。用戶可以打印圖像或者取消這一次操作。如果用戶選擇了打印圖像，那麼一個打印任務將會被創建，同時在系統的通知欄中會顯示一個打印提醒通知。

如果希望在打印輸出中包含更多的內容，而不僅僅是一張圖片，那麼就必須構造一個打印文檔。這方面知識將會在後面的兩節課程中展開。

# 打印HTML文檔

編寫:jdneo - 原文:<http://developer.android.com/training/printing/html-docs.html>

如果要在Android上打印比一副照片更豐富的內容，我們需要將文本和圖片組合在一個待打印的文檔中。Android框架提供了一種使用HTML語言來構建文檔並進行打印的方法，它使用的代碼數量是很小的。

[WebView](#)類在Android 4.4（API Level 19）中得到了更新，使得它可以打印HTML內容。該類允許我們加載一個本地HTML資源或者從網頁下載一個頁面，創建一個打印任務，並把它交給Android打印服務。

這節課將展示如何快速地構建一個包含有文本和圖片的HTML文檔，以及如何使用[WebView](#)打印該文檔。

## 加載一個HTML文檔

用[WebView](#)打印一個HTML文檔，會涉及到加載一個HTML資源，或者用一個字符串構建HTML文檔。這一節將描述如何構建一個HTML的字符串並將它加載到[WebView](#)中，以備打印。

該View對象一般被用來作為一個[Activity](#)佈局的一部分。然而，如果應用當前並沒有使用[WebView](#)，我們可以創建一個該類的實例，以進行打印。創建該自定義View的主要步驟是：

1. 在HTML資源加載完畢後，創建一個[WebViewClient](#)用來啓動一個打印任務。
2. 加載HTML資源至[WebView](#)對象中。

下面的代碼展示瞭如何創建一個簡單的[WebViewClient](#)並且加載一個動態創建的HTML文檔：

```
private WebView mWebView;

private void doWebViewPrint() {
    // Create a WebView object specifically for printing
    WebView webView = new WebView(getActivity());
    webView.setWebViewClient(new WebViewClient() {

        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            return false;
        }

        @Override
        public void onPageFinished(WebView view, String url) {
            Log.i(TAG, "page finished loading " + url);
            createWebPrintJob(view);
            mWebView = null;
        }
    });
}

// Generate an HTML document on the fly:
String htmlDocument = "<html><body><h1>Test Content</h1><p>Testing, " +
    "testing, testing...</p></body></html>";
webView.loadDataWithBaseURL(null, htmlDocument, "text/HTML", "UTF-8", null);

// Keep a reference to WebView object until you pass the PrintDocumentAdapter
// to the PrintManager
mWebView = webView;
}
```

Note：請確保在[WebViewClient](#))中的[onPageFinished\(\)](#)方法內調用創建打印任務的方法。如果沒有等到頁面加載完畢就進行打印，打印的輸出可能會不完整或空白，甚至可能會失敗。

Note：在上面的樣例代碼中，保留了一個WebView對象實例的引用，這樣能夠確保它不會在打印任務創建之前就被垃圾回收器所回收。在編寫代碼時請務必這樣做，否則打印的進程可能會無法繼續執行。

如果我們希望頁面中包含圖像，將這個圖像文件放置在你的工程的“assets/”目錄中，並指定一個基URL（Base URL），並將它作為loadDataWithBaseUrl()方法的第一個參數，就像下面所顯示的一樣：

```
webView.loadDataWithBaseUrl("file:///android_asset/images/", htmlBody,  
    "text/HTML", "UTF-8", null);
```

我們也可以加載一個需要打印的網頁，具體做法是將loadDataWithBaseUrl()方法替換為loadUrl()，如下所示：

```
// Print an existing web page (remember to request INTERNET permission!):  
webView.loadUrl("http://developer.android.com/about/index.html");
```

當使用WebView創建打印文檔時，你要注意下面的一些限制：

- 不能為文檔添加頁眉和頁腳，包括頁號。
- HTML文檔的打印選項不包含選擇打印的頁數範圍，例如：對於一個10頁的HTML文檔，只打印2到4頁是不可以的。
- 一個WebView的實例只能在同一時間處理一個打印任務。
- 若一個HTML文檔包含CSS打印屬性，比如一個landscape屬性，這是不被支持的。
- 不能通過一個HTML文檔中的JavaScript腳本來激活打印。

Note：一旦在佈局中包含的WebView對象將文檔加載完畢後，就可以打印WebView對象的內容了。

如果希望創建一個更加自定義化的打印輸出並希望可以完全控制打印頁面上繪製的內容，可以學習下一節課程：[打印自定義文檔](#)

## 創建一個打印任務

在創建了WebView並加載了我們的HTML內容之後，應用就已經幾乎完成了屬於它的任務。接下來，我們需要訪問PrintManager，創建一個打印適配器，並在最後創建一個打印任務。下面的代碼展示瞭如何執行這些步驟：

```
private void createWebPrintJob(WebView webView) {  
  
    // Get a PrintManager instance  
    PrintManager printManager = (PrintManager) getActivity()  
        .getSystemService(Context.PRINT_SERVICE);  
  
    // Get a print adapter instance  
    PrintDocumentAdapter printAdapter = webView.createPrintDocumentAdapter();  
  
    // Create a print job with name and adapter instance  
    String jobName = getString(R.string.app_name) + " Document";  
    PrintJob printJob = printManager.print(jobName, printAdapter,  
        new PrintAttributes.Builder().build());  
  
    // Save the job object for later status checking  
    mPrintJobs.add(printJob);  
}
```

這個例子保存了一個PrintJob對象的實例，以供我們的應用將來使用，當然這是不必須的。我們的應用可以使用這個對象來跟蹤打印任務執行時的進度。如果希望監控應用中的打印任務是否完成，是否失敗或者是否被用戶取消，這個方法

非常有用。另外，我們不需要創建一個應用內置的通知，因為打印框架會自動的創建一個該打印任務的系統通知。

# 打印自定義文檔

編寫:jdneo - 原文:<http://developer.android.com/training/printing/custom-docs.html>

對於有些應用，比如繪圖應用，頁面佈局應用和其它一些關注於圖像輸出的應用，創造出精美的打印頁面將是它的核心功能。在這種情況下，僅僅打印一幅圖片或一個HTML文檔就不夠了。這類應用的打印輸出需要精確地控制每一個會在頁面中顯示的對象，包括字體，文本流，分頁符，頁眉，頁腳和一些圖像元素等等。

想要創建一個完全自定義的打印文檔，需要投入比之前討論的方法更多的編程精力。我們必須構建可以和打印框架相互通信的組件，調整打印參數，繪製頁面元素並管理多個頁面的打印。

這節課將展示如何連接打印管理器，創建一個打印適配器以及如何構建出需要打印的內容。

## 連接打印管理器

當我們的應用直接管理打印進程時，在收到來自用戶的打印請求後，第一步要做的是連接Android打印框架並獲取一個PrintManager類的實例。該類允許我們初始化一個打印任務並開始打印任務的生命週期。下面的代碼展示瞭如何獲得打印管理器並開始打印進程。

```
private void doPrint() {
    // Get a PrintManager instance
    PrintManager printManager = (PrintManager) getActivity()
        .getSystemService(Context.PRINT_SERVICE);

    // Set job name, which will be displayed in the print queue
    String jobName = getActivity().getString(R.string.app_name) + " Document";

    // Start a print job, passing in a PrintDocumentAdapter implementation
    // to handle the generation of a print document
    printManager.print(jobName, new MyPrintDocumentAdapter(getActivity()),
        null); //
}
```

上面的代碼展示瞭如何命名一個打印任務以及如何設置一個PrintDocumentAdapter類的實例，它負責處理打印生命週期的每一步。打印適配器的實現會在下一節中進行討論。

Note : `print()`方法的最後一個參數接收一個PrintAttributes對象。我們可以使用這個參數向打印框架進行一些打印設置，以及基於前一個打印週期的預設，從而改善用戶體驗。我們也可以使用這個參數對打印內容進行一些更符合實際情況的設置，比如當打印一幅照片時，設置打印的方向與照片方向一致。

## 創建打印適配器

打印適配器負責與Android打印框架交互並處理打印過程的每一步。這個過程需要用戶在創建打印文檔前選擇打印機和打印選項。由於用戶可以選擇不同性能的打印機，不同的頁面尺寸或不同的頁面方向，因此這些選項可能會影響最終的打印效果。當這些選項配置好之後，打印框架會尋求適配器進行佈局並生成一個打印文檔，以此作為打印的前期準備。一旦用戶點擊了打印按鈕，框架會將最終的打印文檔傳遞給Print Provider進行打印輸出。在打印過程中，用戶可以選擇取消打印，所以打印適配器必須監聽並響應取消打印的請求。

PrintDocumentAdapter抽象類負責處理打印的生命週期，它有四個主要的回調方法。我們必須在打印適配器中實現這些方法，以此來正確地和Android打印框架進行交互：

- **onStart()**：一旦打印進程開始，該方法就將被調用。如果我們的應用有任何一次性的準備任務要執行，比如獲取一個要打印數據的快照，那麼讓它們在此處執行。在你的適配器中，這個回調方法不是必須實現的。
- **onLayout()**：每當用戶改變了影響打印輸出的設置時（比如改變了頁面的尺寸，或者頁面的方向）該函數將會被調用，以此給我們的應用一個機會去重新計算打印頁面的佈局。另外，該方法必須返回打印文檔包含多少頁面。
- **onWrite()**：該方法調用後，會將打印頁面渲染成一個待打印的文件。該方法可以在**onLayout()**方法被調用後調用一次或多次。
- **onFinish()**：一旦打印進程結束後，該方法將會被調用。如果我們的應用有任何一次性銷燬任務要執行，讓這些任務在該方法內執行。這個回調方法不是必須實現的。

下面將介紹如何實現 `onLayout()` 以及 `onWrite()` 方法，他們是打印適配器的核心功能。

Note：這些適配器的回調方法會在應用的主線程上被調用。如果這些方法的實現在執行時可能需要花費大量的時間，那麼應該將他們放在另一個線程裏執行。例如：我們可以將佈局或者寫入打印文檔的操作封裝在一個 `AsyncTask` 對象中。

## 計算打印文檔信息

在實現 `PrintDocumentAdapter` 類時，我們的應用必須能夠指定出所創建文檔的類型，計算出打印任務所需要打印的總頁數，並提供打印頁面的尺寸信息。在實現適配器的 `onLayout()` 方法時，我們執行這些計算，並提供與理想的輸出相關的一些信息，這些信息可以在 `PrintDocumentInfo` 類中獲取，包括頁數和內容類型。下面的例子展示了 `PrintDocumentAdapter` 中 `onLayout()` 方法的基本實現：

```
@Override
public void onLayout(PrintAttributes oldAttributes,
                     PrintAttributes newAttributes,
                     CancellationSignal cancellationSignal,
                     LayoutResultCallback callback,
                     Bundle metadata) {
    // Create a new PdfDocument with the requested page attributes
    mPdfDocument = new PrintedPdfDocument(getActivity(), newAttributes);

    // Respond to cancellation request
    if (cancellationSignal.isCancelled() ) {
        callback.onLayoutCancelled();
        return;
    }

    // Compute the expected number of printed pages
    int pages = computePageCount(newAttributes);

    if (pages > 0) {
        // Return print information to print framework
        PrintDocumentInfo info = new PrintDocumentInfo
            .Builder("print_output.pdf")
            .setContentType(PrintDocumentInfo.CONTENT_TYPE_DOCUMENT)
            .setPageCount(pages);
            .build();

        // Content layout reflow is complete
        callback.onLayoutFinished(info, true);
    } else {
        // Otherwise report an error to the print framework
        callback.onLayoutFailed("Page count calculation failed.");
    }
}
```

`onLayout()` 方法的執行結果有三種：完成，取消或失敗（計算佈局無法順利完成時會失敗）。我們必須通過調用 `PrintDocumentAdapter.LayoutResultCallback` 對象中的適當方法來指出這些結果中的一個。

Note：`onLayoutFinished()` 方法的布爾類型參數明確了這個佈局內容是否和上一次打印請求相比發生了改變。恰當地設定了這個參數將避免打印框架不必要的調用 `onWrite()` 方法，緩存之前的打印文檔，提升執行性能。

`onLayout()`的主要任務是計算打印文檔的頁數，並將它作為打印參數交給打印機。如何計算頁數則高度依賴於應用是如何對打印頁面進行佈局的。下面的代碼展示了頁數是如何根據打印方向確定的：

```
private int computeItemCount(PrintAttributes printAttributes) {
    int itemsPerPage = 4; // default item count for portrait mode

    MediaSize pageSize = printAttributes.getMediaSize();
    if (!pageSize.isPortrait()) {
        // Six items per page in landscape orientation
        itemsPerPage = 6;
    }

    // Determine number of print items
    int itemCount = getItemCount();

    return (int) Math.ceil(itemCount / itemsPerPage);
}
```

## 將打印文檔寫入文件

當需要將打印內容輸出到一個文件時，Android打印框架會調用`PrintDocumentAdapter`類的`onWrite()`方法。這個方法的參數指定了哪些頁面要被寫入以及要使用的輸出文件。該方法的實現必須將每一個請求頁的內容渲染成一個含有多個頁面的PDF文件。當這個過程結束以後，你需要調用callback對象的`onWriteFinished()`方法。

Note：Android打印框架可能會在每次調用`onLayout()`後，調用`onWrite()`方法一次甚至更多次。請務必牢記：當打印內容的佈局沒有變化時，可以將`onLayoutFinished()`方法的布爾參數設置為“false”，以此避免對打印文檔進行不必要的重寫操作。

Note：`onLayoutFinished()`方法的布爾類型參數明確了這個佈局內容是否和上一次打印請求相比發生了改變。恰當地設定了這個參數將避免打印框架不必要的調用`onLayout()`方法，緩存之前的打印文檔，提升執行性能。

下面的代碼展示了使用`PrintedPdfDocument`類創建了PDF文件的基本原理：

```
@Override
public void onWrite(final PageRange[] pageRanges,
                    final ParcelFileDescriptor destination,
                    final CancellationSignal cancellationSignal,
                    final WriteResultCallback callback) {
    // Iterate over each page of the document,
    // check if it's in the output range.
    for (int i = 0; i < totalPages; i++) {
        // Check to see if this page is in the output range.
        if (containsPage(pageRanges, i)) {
            // If so, add it to writtenPagesArray. writtenPagesArray.size()
            // is used to compute the next output page index.
            writtenPagesArray.append(writtenPagesArray.size(), i);
            PdfDocument.Page page = mPdfDocument.startPage(i);

            // check for cancellation
            if (cancellationSignal.isCancelled()) {
                callback.onWriteCancelled();
                mPdfDocument.close();
                mPdfDocument = null;
                return;
            }

            // Draw page content for printing
            drawPage(page);

            // Rendering is complete, so page can be finalized.
            mPdfDocument.finishPage(page);
        }
    }
}
```

```

// Write PDF document to file
try {
    mPdfDocument.writeTo(new FileOutputStream(
        destination.getFileDescriptor()));
} catch (IOException e) {
    callback.onWriteFailed(e.toString());
    return;
} finally {
    mPdfDocument.close();
    mPdfDocument = null;
}
PageRange[] writtenPages = computeWrittenPages();
// Signal the print framework the document is complete
callback.onWriteFinished(writtenPages);

...
}

```

代碼中將PDF頁面遞交給了`drawPage()`方法，這個方法會在下一部分介紹。

就佈局而言，`onWrite()`方法的執行可以有三種結果：完成，取消或者失敗（內容無法被寫入）。我們必須通過調用`PrintDocumentAdapter.WriteResultCallback`對象中的適當方法來指明這些結果中的一個。

Note：渲染打印文檔是一個可能耗費大量資源的操作。為了避免阻塞應用的主UI線程，我們應該考慮將頁面的渲染和寫操作放在另一個線程中執行，比如在`AsyncTask`中執行。關於更多異步任務線程的知識，可以閱讀：[Processes and Threads](#)。

## 繪製PDF頁面內容

當我們的應用進行打印時，應用必鬱生成一個PDF文檔並將它傳遞給Android打印框架以進行打印。我們可以使用任何PDF生成庫來協助完成這個操作。本節將展示如何使用`PrintedPdfDocument`類將打印內容生成爲PDF頁面。

`PrintedPdfDocument`類使用一個`Canvas`對象來在PDF頁面上繪製元素，這一點和在`activity`佈局上進行繪製很類似。我們可以在打印頁面上使用`Canvas`類提供的相關繪圖方法繪製頁面元素。下面的代碼展示瞭如何使用這些方法在PDF頁面上繪製一些簡單的元素：

```

private void drawPage(PdfDocument.Page page) {
    Canvas canvas = page.getCanvas();

    // units are in points (1/72 of an inch)
    int titleBaseLine = 72;
    int leftMargin = 54;

    Paint paint = new Paint();
    paint.setColor(Color.BLACK);
    paint.setTextSize(36);
    canvas.drawText("Test Title", leftMargin, titleBaseLine, paint);

    paint.setTextSize(11);
    canvas.drawText("Test paragraph", leftMargin, titleBaseLine + 25, paint);

    paint.setColor(Color.BLUE);
    canvas.drawRect(100, 100, 172, 172, paint);
}

```

當使用`Canvas`在一個PDF頁面上繪圖時，元素通過單位“點（point）”來指定大小，一個點相當於七十二分之一英寸。在編寫程序時，請確保使用該測量單位來指定頁面上的元素大小。在定位繪製的元素時，座標系的原點（即 $(0,0)$ 點）在頁面的最左上角。

Tip：雖然Canvas對象允許我們將打印元素放置在一個PDF文檔的邊緣，但許多打印機無法在紙張的邊緣打印。所以當我們使用這個類構建一個打印文檔時，需要考慮到那些無法打印的邊緣區域。

# Android圖像與動畫

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/building-graphics.html>

這些課程教你如何使用圖形完成任務，這會使你的app在競爭中佔優勢。如果你想創建超越基本用戶界面的漂亮的視覺體驗，這些課程會幫助你做到。

## 高效顯示Bitmap(Displaying Bitmaps Efficiently)

如何在加載並處理bitmaps的同時保持用戶界面響應，防止超出內存限制。

## 使用OpenGL ES顯示圖像(Displaying Graphics with OpenGL ES)

如何使用Android app framework繪製OpenGL圖形並響應觸摸。

## 添加動畫(Adding Animations)

如何給你的用戶界面添加過渡動畫。

# 高效顯示Bitmap

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/index.html>

這一章節會介紹一些處理與加載Bitmap對象的常用方法，這些技術能夠使得程序的UI不會被阻塞，並且可以避免程序超出內存限制。如果我們不注意這些，Bitmaps會迅速的消耗掉可用內存從而導致程序崩潰，出現下面的異常: `java.lang.OutOfMemoryError: bitmap size exceeds VM budget.`

在Android應用中加載Bitmaps的操作是需要特別小心處理的，有下面幾個方面的原因:

- 移動設備的系統資源有限。Android設備對於單個程序至少需要16MB的內存。[Android Compatibility Definition Document \(CDD\)](#), Section 3.7. Virtual Machine Compatibility 中給出了對於不同大小與密度的屏幕的最低內存需求。應用應該在這個最低內存限制下去優化程序的效率。當然，大多數設備的都有更高的限制需求。
- Bitmap會消耗很多內存，特別是對於類似照片等內容更加豐富的圖片。例如，[Galaxy Nexus](#)的照相機能夠拍攝 2592x1936 pixels (5 MB)的圖片。如果bitmap的圖像配置是使用ARGB\_8888 (從Android 2.3開始的默認配置)，那麼加載這張照片到內存大約需要19MB( $2592 \times 1936 \times 4$  bytes) 的空間，從而迅速消耗掉該應用的剩餘內存空間。
- Android應用的UI通常會在一次操作中立即加載許多張Bitmaps。例如在[ListView](#), [GridView](#) 與 [ViewPager](#) 等控件中通常會需要一次加載許多張Bitmaps，而且需要預先加載一些沒有在屏幕上顯示的內容，為用戶滑動的顯示做準備。

## 參考資料

- [DEMO : DisplayingBitmaps.zip](#)
- [VIDEO : Bitmap Allocation](#)
- [VIDEO : Making App Beautiful - Part 4 - Performance Tuning](#)

## 章節課程

- [高效的加載大圖\(Loading Large Bitmaps Efficiently\)](#)

這節課會帶領你學習如何解析很大的Bitmaps並且避免超出程序的內存限制。

- [非UI線程處理Bitmap\(Processing Bitmaps Off the UI Thread\)](#)

處理Bitmap（裁剪，下載等操作）不能執行在主線程。這節課會帶領你學習如何使用AsyncTask在後臺線程對Bitmap進行處理，並解釋如何處理並髮帶來的問題。

- [緩存Bitmaps\(Caching Bitmaps\)](#)

這節課會帶領你學習如何使用內存與磁盤緩存來提升加載多張Bitmaps時的響應速度與流暢度。

- [管理Bitmap的內存使用\(Managing Bitmap Memory\)](#)

這節課會介紹如何管理Bitmap的內存佔用，以此來提升程序的性能。

- [在UI上顯示Bitmap\(Displaying Bitmaps in Your UI\)](#)

這節課會綜合之前章節的內容，演示如何在諸如[ViewPager](#)與[GridView](#)等控件中使用後臺線程與緩存加載多張

Bitmaps ◦

# 高效加載大圖

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/load-bitmap.html>

圖片有不同的形狀與大小。在大多數情況下它們的實際大小都比需要呈現的尺寸大很多。例如，系統的圖庫應用會顯示那些我們使用相機拍攝的照片，但是那些圖片的分辨率通常都比設備屏幕的分辨率要高很多。

考慮到應用是在有限的內存下工作的，理想情況是我們只需要在內存中加載一個低分辨率的照片即可。為了更便於顯示，這個低分辨率的照片應該是與其對應的UI控件大小相匹配的。加載一個超過屏幕分辨率的高分辨率照片不僅沒有任何顯而易見的好處，還會佔用寶貴的內存資源，另外在快速滑動圖片時容易產生額外的效率問題。

這一課會介紹如何通過加載一個縮小版本的圖片，從而避免超出程序的內存限制。

## 讀取位圖的尺寸與類型(Read Bitmap Dimensions and Type)

`BitmapFactory`提供了一些解碼（decode）的方法（`decodeByteArray()`, `decodeFile()`, `decodeResource()`等），用來從不同的資源中創建一個Bitmap。我們應該根據圖片的數據源來選擇合適的解碼方法。這些方法在構造位圖的時候會嘗試分配內存，因此會容易導致 `OutOfMemory` 的異常。每一種解碼方法都可以通過`BitmapFactory.Options`設置一些附加的標記，以此來指定解碼選項。設置 `inJustDecodeBounds` 屬性為 `true` 可以在解碼的時候避免內存的分配，它會返回一個 `null` 的Bitmap，但是可以獲取到 `outWidth`, `outHeight` 與 `outMimeType`。該技術可以允許你在構造Bitmap之前優先讀圖片的尺寸與類型。

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeResource(getResources(), R.id.myimage, options);
int imageHeight = options.outHeight;
int imageWidth = options.outWidth;
String imageType = options.outMimeType;
```

為了避免 `java.lang.OutOfMemory` 的異常，我們需要在真正解析圖片之前檢查它的尺寸（除非你能確定這個數據源提供了準確無誤的圖片且不會導致佔用過多的內存）。

## 加載一個按比例縮小的版本到內存中(Load a Scaled Down Version into Memory)

通過上面的步驟我們已經獲取到了圖片的尺寸，這些數據可以用來幫助我們決定應該加載整個圖片到內存中還是加載一個縮小的版本。有下面一些因素需要考慮：

- 評估加載完整圖片所需要耗費的內存。
- 程序在加載這張圖片時可能涉及到的其他內存需求。
- 顯示這張圖片的控件的尺寸大小。
- 屏幕大小與當前設備的屏幕密度。

例如，如果把一個大小為1024x768像素的圖片顯示到大小為128x96像素的ImageView上嗎，就沒有必要把整張原圖都加載到內存中。

為了告訴解碼器去加載一個縮小版本的圖片到內存中，需要在`BitmapFactory.Options` 中設置 `inSampleSize` 的值。例如，一個分辨率為2048x1536的圖片，如果設置 `inSampleSize` 為4，那麼會產出一個大約512x384大小的Bitmap。

加載這張縮小的圖片僅僅使用大概0.75MB的內存，如果是加載完整尺寸的圖片，那麼大概需要花費12MB（前提都是Bitmap的配置是ARGB\_8888）。下面有一段根據目標圖片大小來計算Sample圖片大小的代碼示例：

```
public static int calculateInSampleSize(
    BitmapFactory.Options options, int reqWidth, int reqHeight) {
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;

    if (height > reqHeight || width > reqWidth) {

        final int halfHeight = height / 2;
        final int halfWidth = width / 2;

        // Calculate the largest inSampleSize value that is a power of 2 and keeps both
        // height and width larger than the requested height and width.
        while ((halfHeight / inSampleSize) > reqHeight
            && (halfWidth / inSampleSize) > reqWidth) {
            inSampleSize *= 2;
        }
    }

    return inSampleSize;
}
```

Note: 設置inSampleSize為2的冪是因為解碼器最終還是會對非2的冪的數進行向下處理，獲取到最靠近2的冪的數。詳情參考[inSampleSize](#)的文檔。

為了使用該方法，首先需要設置 `inJustDecodeBounds` 為 `true`，把 `options` 的值傳遞過來，然後設置 `inSampleSize` 的值並設置 `inJustDecodeBounds` 為 `false`，之後重新調用相關的解碼方法。

```
public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
    int reqWidth, int reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(res, resId, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeResource(res, resId, options);
}
```

使用上面這個方法可以簡單地加載一張任意大小的圖片。如下面的代碼樣例顯示了一個接近100x100像素的縮略圖：

```
mImageView.setImageBitmap(
    decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 100));
```

我們可以通過替換合適的[BitmapFactory.decode\\*](#) 方法來實現一個類似的方法，從其他的數據源解析Bitmap。

# 非UI線程處理Bitmap

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/process-bitmap.html>

在上一課中介紹了一系列的 `BitmapFactory.decode*` 方法，當圖片來源是網絡或者是存儲卡時（或者是任何不在內存中的形式），這些方法都不應該在UI線程中執行。因為在上述情況下加載數據時，其執行時間是不可估計的，它依賴於許多因素（從網絡或者存儲卡讀取數據的速度，圖片的大小，CPU的速度等）。如果其中任何一個子操作阻塞了UI線程，系統都會容易出現應用無響應的錯誤。

這一節課會介紹如何使用 `AsyncTask` 在後臺線程中處理 Bitmap 並且演示如何處理併發（concurrency）的問題。

## 使用 `AsyncTask` (Use a `AsyncTask`)

`AsyncTask` 類提供了一個在後臺線程執行一些操作的簡單方法，它還可以把後臺的執行結果呈現到UI線程中。下面是一個加載大圖的示例：

```
class BitmapWorkerTask extends AsyncTask {
    private final WeakReference imageViewReference;
    private int data = 0;

    public BitmapWorkerTask(ImageView imageView) {
        // Use a WeakReference to ensure the ImageView can be garbage collected
        imageViewReference = new WeakReference(imageView);
    }

    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        data = params[0];
        return decodeSampledBitmapFromResource(getResources(), data, 100, 100);
    }

    // Once complete, see if ImageView is still around and set bitmap.
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
```

為 `ImageView` 使用 `WeakReference` 確保了 `AsyncTask` 所引用的資源可以被垃圾回收器回收。由於當任務結束時不能確保 `ImageView` 仍然存在，因此我們必須在 `onPostExecute()` 裏面對引用進行檢查。該 `ImageView` 在有些情況下可能已經不存在了，例如，在任務結束之前用戶使用了回退操作，或者是配置發生了改變（如旋轉屏幕等）。

開始異步加載位圖，只需要創建一個新的任務並執行它即可：

```
public void loadBitmap(int resId, ImageView imageView) {
    BitmapWorkerTask task = new BitmapWorkerTask(imageView);
    task.execute(resId);
}
```

# 處理並發問題(Handle Concurrency)

通常類似ListView與GridView等視圖控件在使用上面演示的AsyncTask方法時，會同時帶來並發的問題。首先為了更高的效率，ListView與GridView的子Item視圖會在用戶滑動屏幕時被循環使用。如果每一個子視圖都觸發一個AsyncTask，那麼就無法確保關聯的視圖在結束任務時，分配的視圖已經進入循環隊列中，給另外一個子視圖進行重用。而且，無法確保所有的異步任務的完成順序和他們本身的啓動順序保持一致。

[Multithreading for Performance](#) 這篇博文更進一步的討論瞭如何處理並發問題，並且提供了一種解決方法：ImageView保存最近使用的AsyncTask的引用，這個引用可以在任務完成的時候再次讀取檢查。使用這種方式，就可以對前面提到的AsyncTask進行擴展。

創建一個專用的Drawable的子類來儲存任務的引用。在這種情況下，我們使用了一個BitmapDrawable，在任務執行的過程中，一個佔位圖片會顯示在ImageView中：

```
static class AsyncDrawable extends BitmapDrawable {
    private final WeakReference<BitmapWorkerTask> bitmapWorkerTaskReference;

    public AsyncDrawable(Resources res, Bitmap bitmap,
        BitmapWorkerTask bitmapWorkerTask) {
        super(res, bitmap);
        bitmapWorkerTaskReference =
            new WeakReference<BitmapWorkerTask>(bitmapWorkerTask);
    }

    public BitmapWorkerTask getBitmapWorkerTask() {
        return bitmapWorkerTaskReference.get();
    }
}
```

在執行BitmapWorkerTask之前，你需要創建一個AsyncDrawable並且將它綁定到目標控件ImageView中：

```
public void loadBitmap(int resId, ImageView imageView) {
    if (cancelPotentialWork(resId, imageView)) {
        final BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        final AsyncDrawable asyncDrawable =
            new AsyncDrawable(getResources(), mPlaceHolderBitmap, task);
        imageView.setImageDrawable(asyncDrawable);
        task.execute(resId);
    }
}
```

在上面的代碼示例中，cancelPotentialWork方法檢查是否有另一個正在執行的任務與該ImageView關聯了起來，如果的確是這樣，它通過執行cancel()方法來取消另一個任務。在少數情況下，新創建的任務數據可能會與已經存在的任務相吻合，這樣的話就不需要進行下一步動作了。下面是cancelPotentialWork方法的實現。

```
public static boolean cancelPotentialWork(int data, ImageView imageView) {
    final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask(imageView);

    if (bitmapWorkerTask != null) {
        final int bitmapData = bitmapWorkerTask.data;
        if (bitmapData == 0 || bitmapData != data) {
            // Cancel previous task
            bitmapWorkerTask.cancel(true);
        } else {
            // The same work is already in progress
            return false;
        }
    }
    // No task associated with the ImageView, or an existing task was cancelled
```

```
        return true;
    }
```

在上面的代碼中有一個輔助方法：`getBitmapWorkerTask()`，它被用作檢索`AsyncTask`是否已經被分配到指定的`ImageView`:

```
private static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
    if (imageView != null) {
        final Drawable drawable = imageView.getDrawable();
        if (drawable instanceof AsyncDrawable) {
            final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
            return asyncDrawable.getBitmapWorkerTask();
        }
    }
    return null;
}
```

最後一步是在`BitmapWorkerTask`的`onPostExecute()`方法裏面做更新操作:

```
class BitmapWorkerTask extends AsyncTask {
    ...

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (isCancelled()) {
            bitmap = null;
        }

        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            final BitmapWorkerTask bitmapWorkerTask =
                getBitmapWorkerTask(imageView);
            if (this == bitmapWorkerTask && imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
```

這個方法不僅僅適用於`ListView`與`GridView`控件，在那些需要循環利用子視圖的控件中同樣適用：只需要在設置圖片到`ImageView`的地方調用`loadBitmap`方法。例如，在`GridView`中實現這個方法可以在`getView()`中調用。

# 緩存Bitmap

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html>

將單個Bitmap加載到UI是簡單直接的，但是如果我們需要一次性加載大量的圖片，事情則會變得複雜起來。在大多數情況下（例如在使用ListView，GridView或ViewPager時），屏幕上的圖片和因滑動將要顯示的圖片的數量通常是沒有限制的。

通過循環利用子視圖可以緩解內存的使用，垃圾回收器也會釋放那些不再需要使用的Bitmap。這些機制都非常好，但是為了保證一個流暢的用戶體驗，我們希望避免在每次屏幕滑動回來時，都要重複處理那些圖片。內存與磁盤緩存通常可以起到輔助作用，允許控件可以快速地重新加載那些處理過的圖片。

這一課會介紹在加載多張Bitmap時使用內存緩存與磁盤緩存來提高響應速度與UI流暢度。

## 使用內存緩存(Use a Memory Cache)

內存緩存以花費寶貴的程序內存為前提來快速訪問位圖。[LruCache](#)類（在API Level 4的Support Library中也可以找到）特別適合用來緩存Bitmaps，它使用一個強引用（strong referenced）的[LinkedHashMap](#)保存最近引用的對象，並且在緩存超出設置大小的時候剔除（evict）最近最少使用到的對象。

Note: 在過去，一種比較流行的內存緩存實現方法是使用軟引用（SoftReference）或弱引用（WeakReference）對Bitmap進行緩存，然而我們並不推薦這樣的做法。從Android 2.3 (API Level 9)開始，垃圾回收機制變得更加頻繁，這使得釋放軟（弱）引用的頻率也隨之增高，導致使用引用的效率降低很多。而且在Android 3.0 (API Level 11)之前，備份的Bitmap會存放在Native Memory中，它不是以可預知的方式被釋放的，這樣可能導致程序超出它的內存限制而崩潰。

為了給LruCache選擇一個合適的大小，需要考慮到下面一些因素：

- 應用剩下了多少可用的內存？
- 多少張圖片會同時呈現到屏幕上？有多少圖片需要準備好以便馬上顯示到屏幕？
- 設備的屏幕大小與密度是多少？一個具有特別高密度屏幕（xhdpi）的設備，像Galaxy Nexus會比Nexus S（hdpi）需要一個更大的緩存空間來緩存同樣數量的圖片。
- Bitmap的尺寸與配置是多少，會花費多少內存？
- 圖片被訪問的頻率如何？是其中一些比另外的訪問更加頻繁嗎？如果是，那麼我們可能希望在內存中保存那些最常訪問的圖片，或者根據訪問頻率給Bitmap分組，為不同的Bitmap組設置多個LruCache對象。
- 是否可以在緩存圖片的質量與數量之間尋找平衡點？某些時候保存大量低質量的Bitmap會非常有用，加載更高質量圖片的任務可以交給另外一個後臺線程。

通常沒有指定的大小或者公式能夠適用於所有的情形，我們需要分析實際的使用情況後，提出一個合適的解決方案。緩存太小會導致額外的花銷卻沒有明顯的好處，緩存太大同樣會導致java.lang.OutOfMemory的異常，並且使得你的程序只留下小部分的內存用來工作（緩存佔用太多內存，導致其他操作會因為內存不夠而拋出異常）。

下面是一個為Bitmap建立LruCache的示例：

```
private LruCache<String, Bitmap> mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Get max available VM memory, exceeding this amount will throw an
    // OutOfMemory exception. Stored in kilobytes as LruCache takes an
```

```

// int in its constructor.
final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);

// Use 1/8th of the available memory for this memory cache.
final int cacheSize = maxMemory / 8;

mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
    @Override
    protected int sizeOf(String key, Bitmap bitmap) {
        // The cache size will be measured in kilobytes rather than
        // number of items.
        return bitmap.getByteCount() / 1024;
    }
};

...
}

public void addBitmapToMemoryCache(String key, Bitmap bitmap) {
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }
}

public Bitmap getBitmapFromMemCache(String key) {
    return mMemoryCache.get(key);
}

```

Note:在上面的例子中，有1/8的內存空間被用作緩存。這意味着在常見的設備上（hdpi），最少大概有4MB的緩存空間（ $32/8$ ）。如果一個填滿圖片的GridView控件放置在800x480像素的手機屏幕上，大概會花費1.5MB的緩存空間（ $800 \times 480 \times 4$  bytes），因此緩存的容量大概可以緩存2.5頁的圖片內容。

當加載Bitmap顯示到ImageView之前，會先從LruCache中檢查是否存在這個Bitmap。如果確實存在，它會立即被用來顯示到ImageView上，如果沒有找到，會觸發一個後臺線程去處理顯示該Bitmap任務。

```

public void loadBitmap(int resId, ImageView imageView) {
    final String imageKey = String.valueOf(resId);

    final Bitmap bitmap = getBitmapFromMemCache(imageKey);
    if (bitmap != null) {
        imageView.setImageBitmap(bitmap);
    } else {
        imageView.setImageResource(R.drawable.image_placeholder);
        BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        task.execute(resId);
    }
}

```

上面的程序中 `BitmapWorkerTask` 需要把解析好的Bitmap添加到內存緩存中：

```

class BitmapWorkerTask extends AsyncTask<Integer, Void, Bitmap> {
    ...
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        final Bitmap bitmap = decodeSampledBitmapFromResource(
            getResources(), params[0], 100, 100);
        addBitmapToMemoryCache(String.valueOf(params[0]), bitmap);
        return bitmap;
    }
    ...
}

```

## 使用磁盤緩存(Use a Disk Cache)

內存緩存能夠提高訪問最近用過的Bitmap的速度，但是我們無法保證最近訪問過的Bitmap都能夠保存在緩存中。像類似GridView等需要大量數據填充的控件很容易就會用盡整個內存緩存。另外，我們的應用可能會被類似打電話等行為暫停並退到後臺，因為後臺應用可能會被殺死，那麼內存緩存就會被銷燬，裏面的Bitmap也就不存在了。一旦用戶恢復應用的狀態，那麼應用就需要重新處理那些圖片。

磁盤緩存可以用來保存那些已經處理過的Bitmap，它還可以減少那些不再內存緩存中的Bitmap的加載次數。當然從磁盤讀取圖片會比從內存要慢，而且由於磁盤讀取操作時間是不可預期的，讀取操作需要在後臺線程中處理。

Note:如果圖片會被更頻繁的訪問，使用ContentProvider或許會更加合適，比如在圖庫應用中。

這一節的範例代碼中使用了一個從Android源碼中剝離出來的DiskLruCache。改進過的範例代碼在已有內存緩存的基礎上增加磁盤緩存的功能。

```
private DiskLruCache mDiskLruCache;
private final Object mDiskCacheLock = new Object();
private boolean mDiskCacheStarting = true;
private static final int DISK_CACHE_SIZE = 1024 * 1024 * 10; // 10MB
private static final String DISK_CACHE_SUBDIR = "thumbnails";

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    // Initialize memory cache
    ...
    // Initialize disk cache on background thread
    File cacheDir = getDiskCacheDir(this, DISK_CACHE_SUBDIR);
    new InitDiskCacheTask().execute(cacheDir);
    ...
}

class InitDiskCacheTask extends AsyncTask<File, Void, Void> {
    @Override
    protected Void doInBackground(File... params) {
        synchronized (mDiskCacheLock) {
            File cacheDir = params[0];
            mDiskLruCache = DiskLruCache.open(cacheDir, DISK_CACHE_SIZE);
            mDiskCacheStarting = false; // Finished initialization
            mDiskCacheLock.notifyAll(); // Wake any waiting threads
        }
        return null;
    }
}

class BitmapWorkerTask extends AsyncTask<Integer, Void, Bitmap> {
    ...
    // Decode image in background.
    @Override
    protected Bitmap doInBackground(Integer... params) {
        final String imageKey = String.valueOf(params[0]);

        // Check disk cache in background thread
        Bitmap bitmap = getBitmapFromDiskCache(imageKey);

        if (bitmap == null) { // Not found in disk cache
            // Process as normal
            final Bitmap bitmap = decodeSampledBitmapFromResource(
                getResources(), params[0], 100, 100));
        }

        // Add final bitmap to caches
        addBitmapToCache(imageKey, bitmap);

        return bitmap;
    }
    ...
}
```

```

public void addBitmapToCache(String key, Bitmap bitmap) {
    // Add to memory cache as before
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }

    // Also add to disk cache
    synchronized (mDiskCacheLock) {
        if (mDiskLruCache != null && mDiskLruCache.get(key) == null) {
            mDiskLruCache.put(key, bitmap);
        }
    }
}

public Bitmap getBitmapFromDiskCache(String key) {
    synchronized (mDiskCacheLock) {
        // Wait while disk cache is started from background thread
        while (mDiskCacheStarting) {
            try {
                mDiskCacheLock.wait();
            } catch (InterruptedException e) {}
        }
        if (mDiskLruCache != null) {
            return mDiskLruCache.get(key);
        }
    }
    return null;
}

// Creates a unique subdirectory of the designated app cache directory. Tries to use external
// but if not mounted, falls back on internal storage.
public static File getDiskCacheDir(Context context, String uniqueName) {
    // Check if media is mounted or storage is built-in, if so, try and use external cache dir
    // otherwise use internal cache dir
    final String cachePath =
        Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState()) ||
        !isExternalStorageRemovable() ? getExternalCacheDir(context).getPath() :
        context.getCacheDir().getPath();

    return new File(cachePath + File.separator + uniqueName);
}

```

Note:因為初始化磁盤緩存涉及到I/O操作，所以它不應該在主線程中進行。但是這也意味着在初始化完成之前緩存可以被訪問。為瞭解決這個問題，在上面的實現中，有一個鎖對象（lock object）來確保在磁盤緩存完成初始化之前，應用無法對它進行讀取。

內存緩存的檢查是可以在UI線程中進行的，磁盤緩存的檢查需要在後臺線程中處理。磁盤操作永遠都不應該在UI線程中發生。當圖片處理完成後，Bitmap需要添加到內存緩存與磁盤緩存中，方便之後的使用。

## 處理配置改變(Handle Configuration Changes)

如果運行時設備配置信息發生改變，例如屏幕方向的改變會導致Android中當前顯示的Activity先被銷燬然後重啓。（關於這一方面的更多信息，請參考[Handling Runtime Changes](#)）。我們需要在配置改變時避免重新處理所有的圖片，這樣才能提供給用戶一個良好的平滑過度的體驗。

幸運的是，在前面介紹使用內存緩存的部分，我們已經知道了如何建立內存緩存。這個緩存可以通過調用[setRetainInstance\(true\)](#)保留一個Fragment實例的方法把緩存傳遞給新的Activity。在這個Activity被重新創建之後，這個保留的Fragment會被重新附着上。這樣你就可以訪問緩存對象了，從緩存中獲取到圖片信息並快速的重新顯示到ImageView上。

下面是配置改變時使用Fragment來保留LruCache的代碼示例：

```

private LruCache<String, Bitmap> mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    RetainFragment retainFragment =
        RetainFragment.findOrCreateRetainFragment(getFragmentManager());
    mMemoryCache = retainFragment.mRetainedCache;
    if (mMemoryCache == null) {
        mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
            ... // Initialize cache here as usual
        };
        retainFragment.mRetainedCache = mMemoryCache;
    }
    ...
}

class RetainFragment extends Fragment {
    private static final String TAG = "RetainFragment";
    public LruCache<String, Bitmap> mRetainedCache;

    public RetainFragment() {}

    public static RetainFragment findOrCreateRetainFragment(FragmentManager fm) {
        RetainFragment fragment = (RetainFragment) fm.findFragmentByTag(TAG);
        if (fragment == null) {
            fragment = new RetainFragment();
            fm.beginTransaction().add(fragment, TAG).commit();
        }
        return fragment;
    }

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

```

為了測試上面的效果，可以嘗試在保留Fragment與沒有這樣做的情況下旋轉屏幕。我們會發現當保留緩存時，從內存緩存中重新繪製幾乎沒有延遲的現象。內存緩存中沒有的圖片可能存儲在磁盤緩存中。如果兩個緩存中都沒有，則圖像會像平時正常流程一樣被處理。

# 管理Bitmap的內存使用

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/manage-memory.html>

這節課將作為緩存Bitmaps課程的進一步延伸。為了優化垃圾回收機制與Bitmap的重用，我們還有一些特定的事情可以做。同時根據Android的不同版本，推薦的策略會有所差異。[Displaying Bitmaps](#)的示例程序會演示如何設計我們的程序，使得它能夠在不同的Android平臺上高效地運行。

為了給這節課奠定基礎，我們首先要知Android管理Bitmap內存使用的演變進程：

- 在Android 2.2 (API level 8)以及之前，當垃圾回收發生時，應用的線程是會被暫停的，這會導致一個延遲滯後，並降低系統效率。從Android 2.3開始，添加了併發垃圾回收的機制，這意味着在一個Bitmap不再被引用之後，它所佔用的內存會被立即回收。
- 在Android 2.3.3 (API level 10)以及之前，一個Bitmap的像素級數據 (pixel data) 是存放在Native內存空間中的。這些數據與Bitmap本身是隔離的，Bitmap本身被存放在Dalvik堆中。我們無法預測在Native內存中的像素級數據何時會被釋放，這意味着程序容易超過它的內存限制並且崩潰。自Android 3.0 (API Level 11)開始，像素級數據則是與Bitmap本身一起存放在Dalvik堆中。

下面會介紹如何在不同的Android版本上優化Bitmap內存使用。

## 管理Android 2.3.3及以下版本的內存使用

在Android 2.3.3 (API level 10) 以及更低版本上，推薦使用[recycle\(\)](#)方法。如果在應用中顯示了大量的Bitmap數據，我們很可能會遇到[OutOfMemoryError](#)的錯誤。[recycle\(\)](#)方法可以使得程序更快的釋放內存。

Caution：只有當我們確定這個Bitmap不再需要用到的時候才應該使用[recycle\(\)](#)。在執行[recycle\(\)](#)方法之後，如果嘗試繪製這個Bitmap，我們將得到 "Canvas: trying to use a recycled bitmap" 的錯誤提示。

下面的代碼片段演示了使用 [recycle\(\)](#) 的例子。它使用了引用計數的方法 ([mDisplayRefCount](#) 與 [mCacheRefCount](#)) 來追蹤一個Bitmap目前是否有被顯示或者是在緩存中。並且在下面列舉的條件滿足時，回收Bitmap：

- [mDisplayRefCount](#) 與 [mCacheRefCount](#) 的引用計數均為 0；
- [bitmap](#)不為 [null](#)，並且它還沒有被回收。

```
private int mCacheRefCount = 0;
private int mDisplayRefCount = 0;
...
// Notify the drawable that the displayed state has changed.
// Keep a count to determine when the drawable is no longer displayed.
public void setIsDisplayed(boolean isDisplayed) {
    synchronized (this) {
        if (isDisplayed) {
            mDisplayRefCount++;
            mHasBeenDisplayed = true;
        } else {
            mDisplayRefCount--;
        }
    }
    // Check to see if recycle() can be called.
    checkState();
}

// Notify the drawable that the cache state has changed.
// Keep a count to determine when the drawable is no longer being cached.
public void setIsCached(boolean isCached) {
```

```

    synchronized (this) {
        if (isCached) {
            mCacheRefCount++;
        } else {
            mCacheRefCount--;
        }
    }
    // Check to see if recycle() can be called.
    checkState();
}

private synchronized void checkState() {
    // If the drawable cache and display ref counts = 0, and this drawable
    // has been displayed, then recycle.
    if (mCacheRefCount <= 0 && mDisplayRefCount <= 0 && mHasBeenDisplayed
        && hasValidBitmap()) {
        getBitmap().recycle();
    }
}

private synchronized boolean hasValidBitmap() {
    Bitmap bitmap = getBitmap();
    return bitmap != null && !bitmap.isRecycled();
}

```

## 管理Android 3.0及其以上版本的內存

從Android 3.0 (API Level 11)開始，引進了`BitmapFactory.Options.inBitmap`字段。如果使用了這個設置字段，`decode`方法會在加載`Bitmap`數據的時候去重用已經存在的`Bitmap`。這意味着`Bitmap`的內存是被重新利用的，這樣可以提升性能，並且減少了內存的分配與回收。然而，使用`inBitmap`有一些限制，特別是在Android 4.4 (API level 19)之前，只有同等大小的位圖纔可以被重用。詳情請查看[inBitmap文檔](#)。

### 保存`Bitmap`供以後使用

下面演示瞭如何將一個已經存在的`Bitmap`存放起來以便後續使用。當一個應用運行在Android 3.0或者更高的平臺上並且`Bitmap`從`LruCache`中移除時，`Bitmap`的一個軟引用會被存放在`HashSet`中，這樣便於之後可能被`inBitmap`重用：

```

Set<SoftReference<Bitmap>> mReusableBitmaps;
private LruCache<String, BitmapDrawable> mMemoryCache;

// If you're running on Honeycomb or newer, create a
// synchronized HashSet of references to reusable bitmaps.
if (Utils.hasHoneycomb()) {
    mReusableBitmaps =
        Collections.synchronizedSet(new HashSet<SoftReference<Bitmap>>());
}

mMemoryCache = new LruCache<String, BitmapDrawable>(mCacheParams.memCacheSize) {

    // Notify the removed entry that is no longer being cached.
    @Override
    protected void entryRemoved(boolean evicted, String key,
        BitmapDrawable oldValue, BitmapDrawable newValue) {
        if (RecyclingBitmapDrawable.class.isInstance(oldValue)) {
            // The removed entry is a recycling drawable, so notify it
            // that it has been removed from the memory cache.
            ((RecyclingBitmapDrawable) oldValue).setIsCached(false);
        } else {
            // The removed entry is a standard BitmapDrawable.
            if (Utils.hasHoneycomb()) {
                // We're running on Honeycomb or later, so add the bitmap
                // to a SoftReference set for possible use with inBitmap later.
                mReusableBitmaps.add
                    (new SoftReference<Bitmap>(oldValue.getBitmap()));
            }
        }
    }
}

```

```
        }
    }
    ...
}
```

## 使用已經存在的Bitmap

在運行的程序中，decode方法會檢查看是否存在可重用的Bitmap。例如：

```
public static Bitmap decodeSampledBitmapFromFile(String filename,
    int reqWidth, int reqHeight, ImageCache cache) {

    final BitmapFactory.Options options = new BitmapFactory.Options();
    ...
    BitmapFactory.decodeFile(filename, options);
    ...

    // If we're running on Honeycomb or newer, try to use inBitmap.
    if (Utils.hasHoneycomb()) {
        addInBitmapOptions(options, cache);
    }
    ...
    return BitmapFactory.decodeFile(filename, options);
}
```

下面的代碼是上述代碼片段中，addInBitmapOptions() 方法的具體實現。它會為inBitmap查找一個已經存在的Bitmap，並將它設置為inBitmap的值。注意這個方法只有在找到合適且可重用的Bitmap時纔會賦值給inBitmap（我們需要在賦值之前進行檢查）：

```
private static void addInBitmapOptions(BitmapFactory.Options options,
    ImageCache cache) {
    // inBitmap only works with mutable bitmaps, so force the decoder to
    // return mutable bitmaps.
    options.inMutable = true;

    if (cache != null) {
        // Try to find a bitmap to use for inBitmap.
        Bitmap inBitmap = cache.getBitmapFromReusableSet(options);

        if (inBitmap != null) {
            // If a suitable bitmap has been found, set it as the value of
            // inBitmap.
            options.inBitmap = inBitmap;
        }
    }
}

// This method iterates through the reusable bitmaps, looking for one
// to use for inBitmap:
protected Bitmap getBitmapFromReusableSet(BitmapFactory.Options options) {
    Bitmap bitmap = null;

    if (mReusableBitmaps != null && !mReusableBitmaps.isEmpty()) {
        synchronized (mReusableBitmaps) {
            final Iterator<SoftReference<Bitmap>> iterator
                = mReusableBitmaps.iterator();
            Bitmap item;

            while (iterator.hasNext()) {
                item = iterator.next().get();

                if (null != item && item.isMutable()) {
                    // Check to see if the item can be used for inBitmap.
                    if (canUseForInBitmap(item, options)) {
                        bitmap = item;
                    }
                }
            }
        }
    }
}
```

```

        // Remove from reusable set so it can't be used again.
        iterator.remove();
        break;
    }
} else {
    // Remove from the set if the reference has been cleared.
    iterator.remove();
}
}
}

return bitmap;
}

```

最後，下面這個方法判斷候選Bitmap是否滿足inBitmap的大小條件：

```

static boolean canUseForInBitmap(
    Bitmap candidate, BitmapFactory.Options targetOptions) {

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        // From Android 4.4 (KitKat) onward we can re-use if the byte size of
        // the new bitmap is smaller than the reusable bitmap candidate
        // allocation byte count.
        int width = targetOptions.outWidth / targetOptions.inSampleSize;
        int height = targetOptions.outHeight / targetOptions.inSampleSize;
        int byteCount = width * height * getBytesPerPixel(candidate.getConfig());
        return byteCount <= candidate.getAllocationByteCount();
    }

    // On earlier versions, the dimensions must match exactly and the inSampleSize must be 1
    return candidate.getWidth() == targetOptions.outWidth
        && candidate.getHeight() == targetOptions.outHeight
        && targetOptions.inSampleSize == 1;
}

/**
 * A helper function to return the byte usage per pixel of a bitmap based on its configuration.
 */
static int getBytesPerPixel(Config config) {
    if (config == Config.ARGB_8888) {
        return 4;
    } else if (config == Config.RGB_565) {
        return 2;
    } else if (config == Config.ARGB_4444) {
        return 2;
    } else if (config == Config.ALPHA_8) {
        return 1;
    }
    return 1;
}

```

# 在UI上顯示Bitmap

編寫:kesenhoo - 原文:<http://developer.android.com/training/displaying-bitmaps/display-bitmap.html>

這一課會演示如何運用前面幾節課的內容，使用後臺線程與緩存機制將圖片加載到ViewPager與GridView控件，並且學習處理併發與配置改變問題。

## 實現加載圖片到ViewPager

Swipe View Pattern是一個使用滑動來切換顯示不同詳情頁面的設計模型。（關於這種效果請先參看Android Design: Swipe Views）。我們可以通過PagerAdapter與ViewPager控件來實現這個效果。不過，一個更加合適的Adapter是PagerAdapter的一個子類，叫做FragmentStatePagerAdapter：它可以在某個ViewPager中的子視圖切換出屏幕時自動銷燬與保存Fragments的狀態。這樣能夠保持更少的內存消耗。

Note: 如果只有為數不多的圖片並且確保不會超出程序內存限制，那麼使用PagerAdapter或FragmentPagerAdapter會更加合適。

下面是一個使用ViewPager與ImageView作為子視圖的示例。主Activity包含有ViewPager和Adapter。

```
public class ImageDetailActivity extends FragmentActivity {
    public static final String EXTRA_IMAGE = "extra_image";

    private ImagePagerAdapter mAdapter;
    private ViewPager mPager;

    // A static dataset to back the ViewPager adapter
    public final static Integer[] imageResIds = new Integer[] {
        R.drawable.sample_image_1, R.drawable.sample_image_2, R.drawable.sample_image_3,
        R.drawable.sample_image_4, R.drawable.sample_image_5, R.drawable.sample_image_6,
        R.drawable.sample_image_7, R.drawable.sample_image_8, R.drawable.sample_image_9};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.image_detail_pager); // Contains just a ViewPager

        mAdapter = new ImagePagerAdapter(getSupportFragmentManager(), imageResIds.length);
        mPager = (ViewPager) findViewById(R.id.pager);
        mPager.setAdapter(mAdapter);
    }

    public static class ImagePagerAdapter extends FragmentStatePagerAdapter {
        private final int mSize;

        public ImagePagerAdapter(FragmentManager fm, int size) {
            super(fm);
            mSize = size;
        }

        @Override
        public int getCount() {
            return mSize;
        }

        @Override
        public Fragment getItem(int position) {
            return ImageDetailFragment.newInstance(position);
        }
    }
}
```

Fragment裏麵包含了ImageView控件：

```
public class ImageDetailFragment extends Fragment {
    private static final String IMAGE_DATA_EXTRA = "resId";
    private int mImageNum;
    private ImageView mImageView;

    static ImageDetailFragment newInstance(int imageNum) {
        final ImageDetailFragment f = new ImageDetailFragment();
        final Bundle args = new Bundle();
        args.putInt(IMAGE_DATA_EXTRA, imageNum);
        f.setArguments(args);
        return f;
    }

    // Empty constructor, required as per Fragment docs
    public ImageDetailFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mImageNum = getArguments() != null ? getArguments().getInt(IMAGE_DATA_EXTRA) : -1;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // image_detail_fragment.xml contains just an ImageView
        final View v = inflater.inflate(R.layout.image_detail_fragment, container, false);
        mImageView = (ImageView) v.findViewById(R.id.imageView);
        return v;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        final int resId = ImageDetailActivity.imageResIds[mImageNum];
        mImageView.setImageResource(resId); // Load image into ImageView
    }
}
```

希望你有發現上面示例存在的問題：在UI線程中讀取圖片可能會導致應用無響應。因此使用在第二課中學習的AsyncTask會更好。

```
public class ImageDetailActivity extends FragmentActivity {
    ...

    public void loadBitmap(int resId, ImageView imageView) {
        mImageView.setImageResource(R.drawable.image_placeholder);
        BitmapWorkerTask task = new BitmapWorkerTask(mImageView);
        task.execute(resId);
    }

    ... // include BitmapWorkerTask class
}

public class ImageDetailFragment extends Fragment {
    ...

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (ImageDetailActivity.class.isInstance(getActivity())) {
            final int resId = ImageDetailActivity.imageResIds[mImageNum];
            // Call out to ImageDetailActivity to load the bitmap in a background thread
            ((ImageDetailActivity) getActivity()).loadBitmap(resId, mImageView);
        }
    }
}
```

在BitmapWorkerTask中做一些例如重設圖片大小，從網絡拉取圖片的任務，可以確保不會阻塞UI線程。如果後臺線程不僅僅是一個簡單的加載操作，增加一個內存緩存或者磁盤緩存會比較好（請參考第三課：緩存Bitmap），下面是一些為了內存緩存而附加的內容：

```
public class ImageDetailActivity extends FragmentActivity {
    ...
    private LruCache mMemoryCache;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // initialize LruCache as per Use a Memory Cache section
    }

    public void loadBitmap(int resId, ImageView imageView) {
        final String imageKey = String.valueOf(resId);

        final Bitmap bitmap = mMemoryCache.get(imageKey);
        if (bitmap != null) {
            imageView.setImageBitmap(bitmap);
        } else {
            imageView.setImageResource(R.drawable.image_placeholder);
            BitmapWorkerTask task = new BitmapWorkerTask(imageView);
            task.execute(resId);
        }
    }

    ... // include updated BitmapWorkerTask from Use a Memory Cache section
}
```

把前面學習到的所有技巧合並起來，我們將得到一個響應性良好的ViewPager實現：它擁有最小的加載延遲，同時可以根據實際需求執行不同的後臺處理任務。

## 實現加載圖片到GridView

**Grid List Building Block**是一種有效顯示大量圖片的方式。它能夠一次顯示許多圖片，同時即將被顯示的圖片會處於準備顯示的狀態。如果我們想要實現這種效果，必須確保UI是流暢的，能夠控制內存使用，並且正確處理並發問題（因為GridView會循環使用子視圖）。

下面是一個典型的使用場景，在Fragment裏面內置GridView，其中GridView的子視圖是ImageView：

```
public class ImageGridFragment extends Fragment implements AdapterView.OnItemClickListener {
    private ImageAdapter mAdapter;

    // A static dataset to back the GridView adapter
    public final static Integer[] imageResIds = new Integer[] {
        R.drawable.sample_image_1, R.drawable.sample_image_2, R.drawable.sample_image_3,
        R.drawable.sample_image_4, R.drawable.sample_image_5, R.drawable.sample_image_6,
        R.drawable.sample_image_7, R.drawable.sample_image_8, R.drawable.sample_image_9};

    // Empty constructor as per Fragment docs
    public ImageGridFragment() {}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mAdapter = new ImageAdapter(getActivity());
    }

    @Override
    public View onCreateView(
```

```

        LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    final View v = inflater.inflate(R.layout.image_grid_fragment, container, false);
    final GridView mGridView = (GridView) v.findViewById(R.id.gridView);
    mGridView.setAdapter(mAdapter);
    mGridView.setOnItemClickListener(this);
    return v;
}

@Override
public void onItemClick(AdapterView parent, View v, int position, long id) {
    final Intent i = new Intent(getActivity(), ImageDetailActivity.class);
    i.putExtra(ImageDetailActivity.EXTRA_IMAGE, position);
    startActivity(i);
}

private class ImageAdapter extends BaseAdapter {
    private final Context mContext;

    public ImageAdapter(Context context) {
        super();
        mContext = context;
    }

    @Override
    public int getCount() {
        return imageResIds.length;
    }

    @Override
    public Object getItem(int position) {
        return imageResIds[position];
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup container) {
        ImageView imageView;
        if (convertView == null) { // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setLayoutParams(new GridView.LayoutParams(
                LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
        } else {
            imageView = (ImageView) convertView;
        }
        //請注意下面的代碼
        imageView.setImageResource(imageResIds[position]); // Load image into ImageView
        return imageView;
    }
}

```

這裏同樣有一個問題，上面的代碼實現中，犯了把圖片加載放在UI線程進行處理的錯誤。如果只是加載一些很小的圖片，或者是經過Android系統縮放並緩存過的圖片，上面的代碼在運行時不會有太大問題，但是如果加載的圖片稍微複雜耗時一點，這都會導致你的UI卡頓甚至應用無響應。

與前面加載圖片到ViewPager一樣，如果 `setImageResource` 的操作會比較耗時，也有可能會阻塞UI線程。不過我們可以使用類似前面異步處理圖片與增加緩存的方法來解決這個問題。然而，我們還需要考慮GridView的循環機制所帶來的並發問題。為了處理這個問題，可以參考前面的課程。下面是一個更新過後的解決方案：

```

public class ImageGridFragment extends Fragment implements AdapterView.OnItemClickListener {
    ...

    private class ImageAdapter extends BaseAdapter {
        ...

```

```

@Override
public View getView(int position, View convertView, ViewGroup container) {
    ...
    loadBitmap(imageResIds[position], imageView)
    return imageView;
}

public void loadBitmap(int resId, ImageView imageView) {
    if (cancelPotentialWork(resId, imageView)) {
        final BitmapWorkerTask task = new BitmapWorkerTask(imageView);
        final AsyncDrawable asyncDrawable =
            new AsyncDrawable(getResources(), mPlaceHolderBitmap, task);
        imageView.setImageDrawable(asyncDrawable);
        task.execute(resId);
    }
}

static class AsyncDrawable extends BitmapDrawable {
    private final WeakReference<BitmapWorkerTask> bitmapWorkerTaskReference;

    public AsyncDrawable(Resources res, Bitmap bitmap,
        BitmapWorkerTask bitmapWorkerTask) {
        super(res, bitmap);
        bitmapWorkerTaskReference =
            new WeakReference<BitmapWorkerTask>(bitmapWorkerTask);
    }

    public BitmapWorkerTask getBitmapWorkerTask() {
        return bitmapWorkerTaskReference.get();
    }
}

public static boolean cancelPotentialWork(int data, ImageView imageView) {
    final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask(imageView);

    if (bitmapWorkerTask != null) {
        final int bitmapData = bitmapWorkerTask.data;
        if (bitmapData != data) {
            // Cancel previous task
            bitmapWorkerTask.cancel(true);
        } else {
            // The same work is already in progress
            return false;
        }
    }
    // No task associated with the ImageView, or an existing task was cancelled
    return true;
}

private static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
    if (imageView != null) {
        final Drawable drawable = imageView.getDrawable();
        if (drawable instanceof AsyncDrawable) {
            final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
            return asyncDrawable.getBitmapWorkerTask();
        }
    }
    return null;
}

... // include updated BitmapWorkerTask class

```

Note:對於ListView同樣可以套用上面的方法。

上面的方法提供了足夠的彈性，使得我們可以做從網絡下載圖片，並對大尺寸大的數碼照片做縮放等操作而不至於阻塞UI線程。

# 使用OpenGL ES顯示圖像

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/index.html>

Android框架提供了大量的標準工具，用來創建吸引人的，功能豐富的圖形界面。然而，如果我們希望應用在屏幕上所繪製的內容進行更多的控制，或者正在嘗試建立三維圖像，那麼我們就需要一個不同的工具了。由Android框架提供的OpenGL ES接口給予我們一組可以顯示高級動畫和圖形的工具集，它的功能僅僅受限於我們自身的想象力。同時，在許多Android設備上搭載的圖形處理單元（GPU）都能為其提供GPU加速等性能優化。

這系列課程將展示如何使用OpenGL構建應用的基礎知識，包括配置，繪製對象，移動圖形元素以及響應點擊事件。

這系列課程所涉及的樣例代碼使用的是OpenGL ES 2.0接口，這是當前Android設備所推薦的接口版本。關於更多OpenGL ES的版本信息，可以閱讀：[OpenGL開發手冊](#)。

Note：注意不要把OpenGL ES 1.x版本的接口和OpenGL ES 2.0的接口混合調用。這兩種版本的接口不是通用的。如果嘗試混用它們可能會讓你感到無奈和沮喪。

## Sample Code

[OpenGLES.zip](#)

## Lessons

### • 配置OpenGL ES的環境

學習如何配置一個可以繪製OpenGL圖形的應用。

### • 定義形狀

學習如何定義形狀，以及為何需要瞭解面（Faces）和卷繞（Winding）這兩個概念的原因。

### • 繪製形狀

學習如何在應用中利用OpenGL繪製形狀。

### • 運用投影與相機視角

學習如何通過投影和相機視角，獲取圖形對象的一個新的透視效果。

### • 添加移動

學習如何對一個OpenGL圖形對象添加基本的運動效果。

### • 嘉應觸摸事件

學習如何與OpenGL圖形進行基本的交互。

# 建立OpenGL ES的環境

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/environment.html>

要在應用中使用OpenGL ES繪製圖像，我們必須為它們創建一個View容器。一種比較直接的方法是實現`GLSurfaceView`類和`GLSurfaceView.Renderer`類。其中，`GLSurfaceView`是一個View容器，它用來存放使用OpenGL繪製的圖形，而`GLSurfaceView.Renderer`則用來控制在該View中繪製的內容。關於這兩個類的更多信息，你可以閱讀：[OpenGL ES開發手冊](#)。

使用`GLSurfaceView`是一種將OpenGL ES集成到應用中的方法之一。對於一個全屏的或者接近全屏的圖形View，使用它是一個理想的選擇。開發者如果希望把OpenGL ES的圖形集成在佈局的一小部分裏面，那麼可以考慮使用`TextureView`。對於喜歡自己動手實現的開發者來說，還可以通過使用`SurfaceView`搭建一個OpenGL ES View，但這將需要編寫更多的代碼。

在這節課中，我們將展示如何在一個的`Activity`中完成`GLSurfaceView`和`GLSurfaceView.Renderer`的最簡單的實現。

## 在Manifest配置文件中聲明使用OpenGL ES

為了讓應用能夠使用OpenGL ES 2.0接口，我們必須將下列聲明添加到Manifest配置文件當中：

```
<uses-feature android:glesVersion="0x00020000" android:required="true" />
```

如果我們的應用使用紋理壓縮（Texture Compression），那麼我們必須對支持的壓縮格式也進行聲明，確保應用僅安裝在可以兼容的設備上：

```
<supports-gl-texture android:name="GL_OES_compressed_ETC1_RGB8_texture" />
<supports-gl-texture android:name="GL_OES_compressed_paletted_texture" />
```

更多關於紋理壓縮的內容，可以閱讀：[OpenGL開發手冊](#)。

## 為OpenGL ES圖形創建一個activity

使用OpenGL ES的安卓應用就像其它類型的應用一樣有自己的用戶接口，即也擁有多個`Activity`。主要的區別體現在Activity佈局內容上的差異。在許多應用中你可能會使用`TextView`，`Button`和`ListView`等，而在使用OpenGL ES的應用中，我們還可以添加一個`GLSurfaceView`。

下面的代碼展示了一個使用`GLSurfaceView`作為其主View的`Activity`：

```
public class OpenGL20Activity extends Activity {

    private GLSurfaceView mGLView;

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

        // Create a GLSurfaceView instance and set it
        // as the Content View for this Activity.
        mGLView = new MyGLSurfaceView(this);
```

```
        setContentView(mGLView);
    }
}
```

Note : OpenGL ES 2.0需要Android 2.2 ( API Level 8 ) 或更高版本的系統，所以確保你的Android項目的API版本滿足該要求。

## 構建一個GLSurfaceView對象

**GLSurfaceView**是一種比較特殊的View，我們可以在該View中繪製OpenGL ES圖形，不過它自己並不做太多和繪製圖形相關的任務。繪製對象的任務是由你在該View中配置的**GLSurfaceView.Renderer**所控制的。事實上，這個對象的代碼非常簡短，你可能會希望不要繼承它，直接創建一個未經修改的GLSurfaceView實例，不過請不要這麼做，因為我們需要繼承該類來捕捉觸控事件，這方面知識會在**響應觸摸事件**（該系列課程的最後一節課）中做進一步的介紹。

**GLSurfaceView**的核心代碼非常簡短，所以對於一個快速的實現而言，我們通常可以在Activity中創建一個內部類並使用它：

```
class MyGLSurfaceView extends GLSurfaceView {
    private final MyGLRenderer mRenderer;
    public MyGLSurfaceView(Context context){
        super(context);
        // Create an OpenGL ES 2.0 context
        setEGLContextClientVersion(2);
        mRenderer = new MyGLRenderer();
        // Set the Renderer for drawing on the GLSurfaceView
        setRenderer(mRenderer);
    }
}
```

另一個對於**GLSurfaceView**實現的可選選項，是將渲染模式設置

為：**GLSurfaceView.RENDERMODE\_WHEN\_DIRTY**，其含義是：僅在你的繪製數據發生變化時纔在視圖中進行繪製操作：

```
// Render the view only when there is a change in the drawing data
setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
```

如果選用這一配置選項，那麼除非調用了**requestRender()**，否則**GLSurfaceView**不會被重新繪製，這樣做可以讓應用的性能及效率得到提高。

## 構建一個渲染類

在一個使用OpenGL ES的應用中，一個**GLSurfaceView.Renderer**類的實現（或者我們將其稱之爲渲染器），正是事情變得有趣的地方。該類會控制和其相關聯的**GLSurfaceView**，具體而言，它會控制在**GLSurfaceView**上繪製的內容。在渲染器中，一共有三個方法會被Android系統調用，以此來明確要在**GLSurfaceView**上繪製的內容以及如何繪製：

- **onSurfaceCreated()**：調用一次，用來配置View的OpenGL ES環境。
- **onDrawFrame()**：每次重新繪製View時被調用。
- **onSurfaceChanged()**：如果View的幾何形態發生變化時會被調用，例如當設備的屏幕方向發生改變時。

下面是一個非常基本的OpenGL ES渲染器的實現，它僅僅在[GLSurfaceView](#)中畫一個黑色的背景：

```
public class MyGLRenderer implements GLSurfaceView.Renderer {  
  
    public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
        // Set the background frame color  
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  
    }  
  
    public void onDrawFrame(GL10 unused) {  
        // Redraw background color  
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);  
    }  
  
    public void onSurfaceChanged(GL10 unused, int width, int height) {  
        GLES20.glViewport(0, 0, width, height);  
    }  
}
```

就是這樣！上面的代碼創建了一個簡單地應用程序，它使用OpenGL讓屏幕呈現為黑色。雖然它的代碼看上去並沒有做什麼非常有意思的事情，但是通過創建這些類，我們已經對使用OpenGL繪製圖形有了基本的認識和鋪墊。

Note：你可能想知道，自己明明使用的是OpenGL ES 2.0接口，為什麼這些方法會有一個**GL10**的參數。這是因為這些方法的簽名（Method Signature）在2.0接口中被簡單地重用了，以此來保持Android框架的代碼儘量簡單。

如果你對OpenGL ES接口很熟悉，那麼你現在就可以在你的應用中構建一個OpenGL ES的環境並繪製圖形了。當然，如果你希望獲取更多的幫助來學會使用OpenGL，那麼請繼續學習下一節課程獲取更多的知識。

# 定義形狀

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/shapes.html>

在一個OpenGL ES View的上下文（Context）中定義形狀，是創建你的傑作所需要的第一步。在瞭解關於OpenGL ES如何定義圖形對象的基本知識之前，通過OpenGL ES 繪圖可能會有些困難。

這節課將講解OpenGL ES相對於Android設備屏幕的座標系，定義形狀和形狀表面的基本知識，如定義一個三角形和一個矩形。

## 定義一個三角形

OpenGL ES允許我們使用三維空間的座標來定義繪畫對象。所以在我們能畫三角形之前，必須先定義它的座標。在OpenGL 中，典型的辦法是為座標定義一個浮點型的頂點數組。為了高效起見，我們可以將座標寫入一個[ByteBuffer](#)，它將會傳入OpenGl ES的圖形處理流程中：

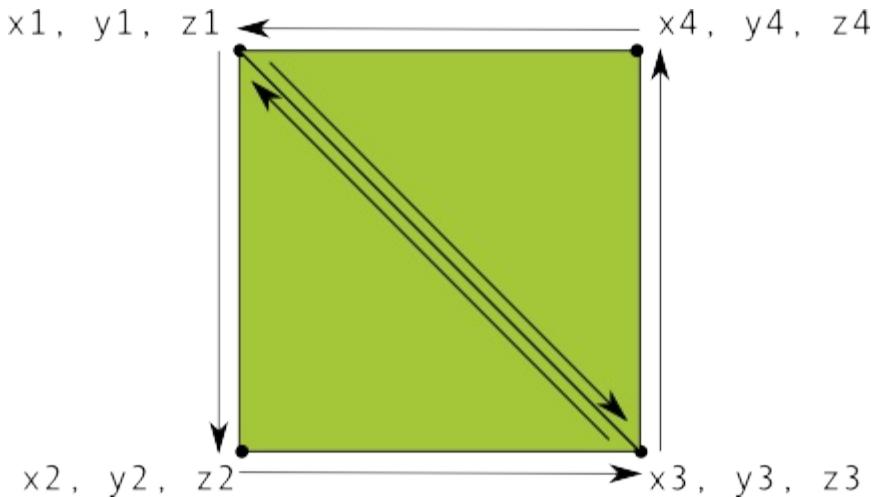
```
public class Triangle {  
  
    private ByteBuffer vertexBuffer;  
  
    // number of coordinates per vertex in this array  
    static final int COORDS_PER_VERTEX = 3;  
    static float triangleCoords[] = { // in counterclockwise order:  
        0.0f, 0.622008459f, 0.0f, // top  
        -0.5f, -0.311004243f, 0.0f, // bottom left  
        0.5f, -0.311004243f, 0.0f // bottom right  
    };  
  
    // Set color with red, green, blue and alpha (opacity) values  
    float color[] = { 0.63671875f, 0.76953125f, 0.22265625f, 1.0f };  
  
    public Triangle() {  
        // initialize vertex byte buffer for shape coordinates  
        ByteBuffer bb = ByteBuffer.allocateDirect(  
            // (number of coordinate values * 4 bytes per float)  
            triangleCoords.length * 4);  
        // use the device hardware's native byte order  
        bb.order(ByteOrder.nativeOrder());  
  
        // create a floating point buffer from the ByteBuffer  
        vertexBuffer = bb.asFloatBuffer();  
        // add the coordinates to the FloatBuffer  
        vertexBuffer.put(triangleCoords);  
        // set the buffer to read the first coordinate  
        vertexBuffer.position(0);  
    }  
}
```

默認情況下，OpenGL ES會假定一個座標系，在這個座標系中，[0, 0, 0]（分別對應X軸座標，Y軸座標，Z軸座標）對應的是GLSurfaceView的中心。[1, 1, 0]對應的是右上角，[-1, -1, 0]對應的則是左下角。如果想要看此座標系的插圖說明，可以閱讀[OpenGL ES開發手冊](#)。

注意到這個形狀的座標是以逆時針順序定義的。繪製的順序非常關鍵，因為它定義了哪一面是形狀的正面（希望繪製的一面），以及背面（使用OpenGL ES的Cull Face功能可以讓背面不要繪製）。更多關於該方面的信息，可以閱讀[OpenGL ES開發手冊](#)。

## 定義一個矩形

在OpenGL中定義三角形非常簡單，那麼你是否想要來點更複雜的呢？比如，定義一個矩形？有很多方法可以用來定義矩形，不過在OpenGL ES中最典型的辦法是使用兩個三角形拼接在一起：



再一次地，我們需要逆時針地為三角形頂點定義座標來表現這個圖形，並將值放入一個[ByteBuffer](#)中。為了避免由兩個三角形重合的那條邊的頂點被重複定義，可以使用一個繪製列表來告訴OpenGL ES圖形處理流程應該如何畫這些頂點。下面是代碼樣例：

```
public class Square {  
  
    private ByteBuffer vertexBuffer;  
    private ShortBuffer drawListBuffer;  
  
    // number of coordinates per vertex in this array  
    static final int COORDS_PER_VERTEX = 3;  
    static float squareCoords[] = {  
        -0.5f, 0.0f, 0.0f, // top left  
        -0.5f, -0.5f, 0.0f, // bottom left  
        0.5f, -0.5f, 0.0f, // bottom right  
        0.5f, 0.5f, 0.0f }; // top right  
  
    private short drawOrder[] = { 0, 1, 2, 0, 2, 3 }; // order to draw vertices  
  
    public Square() {  
        // initialize vertex byte buffer for shape coordinates  
        ByteBuffer bb = ByteBuffer.allocateDirect(  
            // (# of coordinate values * 4 bytes per float)  
            squareCoords.length * 4);  
        bb.order(ByteOrder.nativeOrder());  
        vertexBuffer = bb.asFloatBuffer();  
        vertexBuffer.put(squareCoords);  
        vertexBuffer.position(0);  
  
        // initialize byte buffer for the draw list  
        ByteBuffer dblb = ByteBuffer.allocateDirect(  
            // (# of coordinate values * 2 bytes per short)  
            drawOrder.length * 2);  
        dblb.order(ByteOrder.nativeOrder());  
        drawListBuffer = dblb.asShortBuffer();  
        drawListBuffer.put(drawOrder);  
        drawListBuffer.position(0);  
    }  
}
```

該樣例可以看作是一個如何使用OpenGL創建複雜圖形的啓發，通常來說，我們需要使用三角形的集合來繪製對象。在

下一節課中，我們將學習如何在屏幕上畫這些形狀。

# 繪製形狀

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/draw.html>

在定義了使用OpenGL繪製的形狀之後，你可能希望繪製出它們。使用OpenGL ES 2.0繪製圖形可能會比你想象當中更複雜一些，因為API中提供了大量對於圖形渲染流程的控制。

這節課將解釋如何使用OpenGL ES 2.0接口畫出在上一節課中定義的形狀。

## 初始化形狀

在你開始繪畫之前，你需要初始化並加載你期望繪製的圖形。除非你所使用的形狀結構（原始座標）在執行過程中發生了變化，不然的話你應該在渲染器的`onSurfaceCreated()`方法中初始化它們，這樣做是出於內存和執行效率的考量。

```
public class MyGLRenderer implements GLSurfaceView.Renderer {  
  
    ...  
    private Triangle mTriangle;  
    private Square mSquare;  
  
    public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
        ...  
  
        // initialize a triangle  
        mTriangle = new Triangle();  
        // initialize a square  
        mSquare = new Square();  
    }  
    ...  
}
```

## 畫一個形狀

使用OpenGL ES 2.0畫一個定義好的形狀需要較多代碼，因為你需要提供很多圖形渲染流程的細節。具體而言，你必須定義如下幾項：

- 頂點着色器（Vertex Shader）：用來渲染形狀頂點的OpenGL ES代碼。
- 片段着色器（Fragment Shader）：使用顏色或紋理渲染形狀表面的OpenGL ES代碼。
- 程式（Program）：一個OpenGL ES對象，包含了你希望用來繪製一個或更多圖形所要用到的着色器。

你需要至少一個頂點着色器來繪製一個形狀，以及一個片段着色器為該形狀上色。這些着色器必須被編譯然後添加到一個OpenGL ES Program當中，並利用它來繪製形狀。下面的代碼在Triangle類中定義了基本的着色器，我們可以利用它們繪製出一個圖形：

```
public class Triangle {  
  
    private final String vertexShaderCode =  
        "attribute vec4 vPosition;" +  
        "void main() {" +  
        "    gl_Position = vPosition;" +  
        "}";  
  
    private final String fragmentShaderCode =  
        "precision mediump float;" +  
        "uniform vec4 vColor;" +
```

```

    "void main() {" +
    "    gl_FragColor = vColor;" +
    "}";
}

...

```

着色器包含了OpenGL Shading Language ( GLSL ) 代碼，它必須先被編譯然後才能在OpenGL環境中使用。要編譯這些代碼，需要在你的渲染器類中創建一個輔助方法：

```

public static int loadShader(int type, String shaderCode){

    // create a vertex shader type (GLES20.GL_VERTEX_SHADER)
    // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)
    int shader = GLES20.glCreateShader(type);

    // add the source code to the shader and compile it
    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}

```

為了繪製你的圖形，你必須編譯着色器代碼，將它們添加至一個OpenGL ES Program對象中，然後執行鏈接。在你的繪製對象的構造函數裏做這些事情，這樣上述步驟就只用執行一次。

**Note :** 編譯OpenGL ES着色器及鏈接操作對於CPU週期和處理時間而言，消耗是巨大的，所以你應該避免重複執行這些事情。如果在執行期間不知道着色器的內容，那麼你應該在構建你的應用時，確保它們只被創建了一次，並且緩存以備後續使用。

```

public class Triangle() {
    ...

    private final int mProgram;

    public Triangle() {
        ...

        int vertexShader = MyGLRenderer.loadShader(GLES20.GL_VERTEX_SHADER,
                                                    vertexShaderCode);
        int fragmentShader = MyGLRenderer.loadShader(GLES20.GL_FRAGMENT_SHADER,
                                                    fragmentShaderCode);

        // create empty OpenGL ES Program
        mProgram = GLES20.glCreateProgram();

        // add the vertex shader to program
        GLES20.glAttachShader(mProgram, vertexShader);

        // add the fragment shader to program
        GLES20.glAttachShader(mProgram, fragmentShader);

        // creates OpenGL ES program executables
        GLES20.glLinkProgram(mProgram);
    }
}

```

至此，你已經完全準備好添加實際的調用語句來繪製你的圖形了。使用OpenGL ES繪製圖形需要你定義一些變量來告訴渲染流程你需要繪製的內容以及如何繪製。既然繪製屬性會根據形狀的不同而發生變化，把繪製邏輯包含在形狀類裏面將是一個不錯的主意。

創建一個 `draw()` 方法來繪製圖形。下面的代碼為形狀的頂點着色器和形狀着色器設置了位置和顏色值，然後執行繪製函

數：

```
private int mPositionHandle;
private int mColorHandle;

private final int vertexCount = triangleCoords.length / COORDS_PER_VERTEX;
private final int vertexStride = COORDS_PER_VERTEX * 4; // 4 bytes per vertex

public void draw() {
    // Add program to OpenGL ES environment
    GLES20.glUseProgram(mProgram);

    // get handle to vertex shader's vPosition member
    mPositionHandle = GLES20.glGetAttribLocation(mProgram, "vPosition");

    // Enable a handle to the triangle vertices
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // Prepare the triangle coordinate data
    GLES20.glVertexAttribPointer(mPositionHandle, COORDS_PER_VERTEX,
                                 GLES20.GL_FLOAT, false,
                                 vertexStride, vertexBuffer);

    // get handle to fragment shader's vColor member
    mColorHandle = GLES20 glGetUniformLocation(mProgram, "vColor");

    // Set color for drawing the triangle
    GLES20.glUniform4fv(mColorHandle, 1, color, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

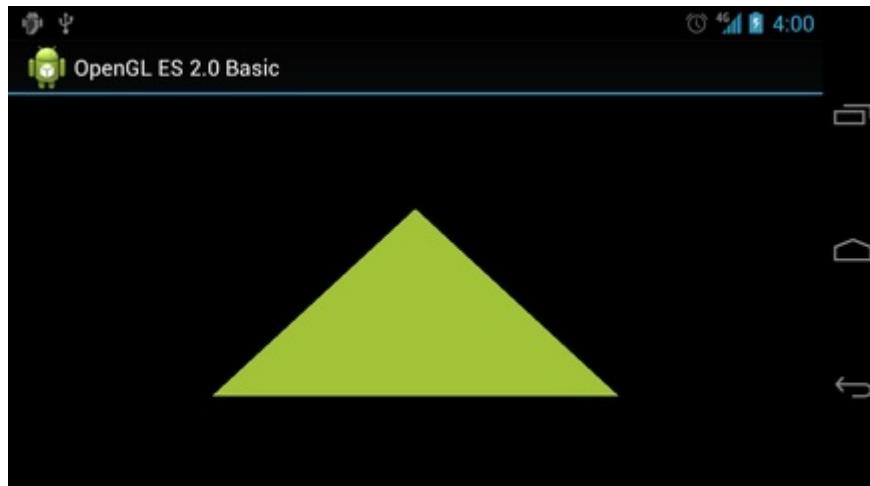
    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
```

一旦你完成了上述所有代碼，僅需要在你渲染器的[onDrawFrame\(\)](#)方法中調用 `draw()` 方法就可以畫出我們想要畫的對象了：

```
public void onDrawFrame(GL10 unused) {
    ...

    mTriangle.draw();
}
```

當你運行這個應用時，它看上去會像是這樣：



在這個代碼樣例中，還存在一些問題。首先，它無法給用戶帶來什麼深刻的印象。其次，這個三角形看上去有一些扁，另外當你改變屏幕方向時，它的形狀也會隨之改變。發生形變的原因是因為對象的頂點沒有根據顯示`GLSurfaceView`的屏幕區域的長寬比進行修正。你可以在下一節課中使用投影（Projection）或者相機視角（Camera View）來解決這個問題。

最後，這個三角形是靜止的，這看上去有些無聊。在[添加移動](#)課程當中（後續課程），你會讓這個形狀發生旋轉，並使用一些OpenGL ES圖形處理流程中更加新奇的用法。

# 運用投影與相機視角

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/projection.html>

在OpenGL ES環境中，利用投影和相機視角可以讓顯示的繪圖對象更加酷似於我們用肉眼看到的真實物體。該物理視角的仿真是對繪製對象座標的進行數學變換實現的：

- 投影（Projection）：這個變換會基於顯示它們的`GLSurfaceView`的長和寬，來調整繪圖對象的座標。如果沒有該計算，那麼用OpenGL ES繪製的對象會由於其長寬比例和View窗口比例的不一致而發生形變。一個投影變換一般僅當OpenGL View的比例在渲染器的`onSurfaceChanged()`方法中建立或發生變化時才被計算。關於更多OpenGL ES投影和座標映射的知識，可以閱讀[Mapping Coordinates for Drawn Objects](#)。
- 相機視角（Camera View）：這個變換會基於一個虛擬相機位置改變繪圖對象的座標。注意到OpenGL ES並沒有定義一個實際的相機對象，取而代之的，它提供了一些輔助方法，通過對繪圖對象的變換來模擬相機視角。一個相機視角變換可能僅在建立你的`GLSurfaceView`時計算一次，也可能根據用戶的行為或者你的應用的功能進行動態調整。

這節課將解釋如何創建一個投影和一個相機視角，並應用它們到`GLSurfaceView`中的繪製圖像上。

## 定義一個投影

投影變換的數據會在`GLSurfaceView.Renderer`類的`onSurfaceChanged()`方法中被計算。下面的代碼首先接收`GLSurfaceView`的高和寬，然後利用它並使用`Matrix.frustumM()`方法來填充一個投影變換矩陣（Projection Transformation Matrix）：

```
// mMVPMatrix is an abbreviation for "Model View Projection Matrix"
private final float[] mMVPMatrix = new float[16];
private final float[] mProjectionMatrix = new float[16];
private final float[] mViewMatrix = new float[16];

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

    float ratio = (float) width / height;

    // this projection matrix is applied to object coordinates
    // in the onDrawFrame() method
    Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1, 1, 3, 7);
}
```

該代碼填充了一個投影矩陣：`mProjectionMatrix`，在下一節中，我們可以在`onDrawFrame()`方法中將它和一個相機視角變換結合起來。

Note：在繪圖對象上只應用一個投影變換會導致顯示效果看上去很空曠。一般而言，我們還要實現一個相機視角，使得所有對象出現在屏幕上。

## 定義一個相機視角

在渲染器中添加一個相機視角變換作為繪圖過程的一部分，以此完成我們的繪圖對象所需變換的所有步驟。在下面的代碼中，使用`Matrix.setLookAtM()`方法來計算相機視角變換，然後與之前計算的投影矩陣結合起來，結合後的變換矩陣傳遞給繪製圖像：

```

@Override
public void onDrawFrame(GL10 unused) {
    ...
    // Set the camera position (View matrix)
    Matrix.setLookAtM(mViewMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);

    // Calculate the projection and view transformation
    Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix, 0, mViewMatrix, 0);

    // Draw shape
    mTriangle.draw(mMVPMatrix);
}

```

## 應用投影和相機變換

為了使用在之前章節中結合了的相機視角變換和投影變換，我們首先為之前在Triangle類中定義的頂點着色器添加一個Matrix變量：

```

public class Triangle {

    private final String vertexShaderCode =
        // This matrix member variable provides a hook to manipulate
        // the coordinates of the objects that use this vertex shader
        "uniform mat4 uMVPMatrix;" +
        "attribute vec4 vPosition;" +
        "void main() {" +
        // the matrix must be included as a modifier of gl_Position
        // Note that the uMVPMatrix factor *must* be first* in order
        // for the matrix multiplication product to be correct.
        "    gl_Position = uMVPMatrix * vPosition;" +
        "}";

        // Use to access and set the view transformation
        private int mMVPMatrixHandle;

    ...
}

```

之後，修改圖形對象的 `draw()` 方法，使得它接收組合後的變換矩陣，並將它應用到圖形上：

```

public void draw(float[] mvpMatrix) { // pass in the calculated transformation matrix
    ...

    // get handle to shape's transformation matrix
    mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVPMatrix");

    // Pass the projection and view transformation to the shader
    GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1, false, mvpMatrix, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}

```

一旦我們正確地計算並應用了投影變換和相機視角變換，我們的圖形就會以正確的比例繪製出來，它看上去會像是這樣：



現在，應用已經可以通過正確的比例顯示圖形了，下面就為圖形添加一些動畫效果吧！

# 添加移動

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/motion.html>

在屏幕上繪製圖形是OpenGL的一個基本特性，當然我們也可以通過其它的Android圖形框架類做這些事情，包括Canvas和Drawable對象。OpenGL ES的特別之處在於，它還提供了其它的一些功能，比如在三維空間中對繪製圖形進行移動和變換操作，或者通過其它獨有的方法創建出引人入勝的用戶體驗。

在這節課中，我們會更深入地學習OpenGL ES的知識：對一個圖形添加旋轉動畫。

## 旋轉一個形狀

使用OpenGL ES 2.0 旋轉一個繪製圖形是比較簡單的。在渲染器中，創建另一個變換矩陣（一個旋轉矩陣），並且將它和我們的投影變換矩陣以及相機視角變換矩陣結合在一起：

```
private float[] mRotationMatrix = new float[16];
public void onDrawFrame(GL10 gl) {
    float[] scratch = new float[16];

    ...

    // Create a rotation transformation for the triangle
    long time = SystemClock.uptimeMillis() % 4000L;
    float angle = 0.090f * ((int) time);
    Matrix.setRotateM(mRotationMatrix, 0, angle, 0, 0, -1.0f);

    // Combine the rotation matrix with the projection and camera view
    // Note that the mMVPMatrix factor *must* be first* in order
    // for the matrix multiplication product to be correct.
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix, 0);

    // Draw triangle
    mTriangle.draw(scratch);
}
```

如果完成了這些變更以後，你的三角形還是沒有旋轉的話，確認一下你是否將啓用`GLSurfaceView.RENDERMODE_WHEN_DIRTY`的這一配置所對應的代碼註釋掉了，有關該方面的知識會在下一節中展開。

## 啓用連續渲染

如果嚴格按照這節課的樣例代碼走到了現在這一步，那麼請確認一下是否將設置渲染模式為`RENDERMODE_WHEN_DIRTY`的那行代碼註釋了，不然的話OpenGL只會對這個形狀執行一次旋轉，然後就等待`GLSurfaceView`容器的`requestRender()`方法被調用後纔會繼續執行渲染操作。

```
public MyGLSurfaceView(Context context) {
    ...
    // Render the view only when there is a change in the drawing data.
    // To allow the triangle to rotate automatically, this line is commented out:
    //setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
}
```

除非某個對象，它的變化和用戶的交互無關，不然的話一般還是建議將這個配置打開。在下一節課中的內容將會把這個

註釋放開，再次設定這一配置選項。

# 響應觸摸事件

編寫:jdneo - 原文:<http://developer.android.com/training/graphics/opengl/touch.html>

讓對象根據預設的程序運動（如讓一個三角形旋轉），可以有效地引起用戶的注意，但是如果希望讓OpenGL ES的圖形對象與用戶交互呢？讓我們的OpenGL ES應用可以支持觸控交互的關鍵點在於，拓展`GLSurfaceView`的實現，重寫`onTouchEvent()`方法來監聽觸摸事件。

這節課將會向你展示如何監聽觸控事件，讓用戶旋轉一個OpenGL ES對象。

## 配置觸摸監聽器

為了讓我們的OpenGL ES應用響應觸控事件，我們必須實現`GLSurfaceView`類中的`onTouchEvent()`方法。下面的例子展示了如何監聽`MotionEvent.ACTION_MOVE`事件，並將事件轉換為形狀旋轉的角度：

```
private final float TOUCH_SCALE_FACTOR = 180.0f / 320;
private float mPreviousX;
private float mPreviousY;

@Override
public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. In this case, you are only
    // interested in events where the touch position changed.

    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
        case MotionEvent.ACTION_MOVE:

            float dx = x - mPreviousX;
            float dy = y - mPreviousY;

            // reverse direction of rotation above the mid-line
            if (y > getHeight() / 2) {
                dx = dx * -1;
            }

            // reverse direction of rotation to left of the mid-line
            if (x < getWidth() / 2) {
                dy = dy * -1;
            }

            mRenderer.setAngle(
                mRenderer.getAngle() +
                ((dx + dy) * TOUCH_SCALE_FACTOR));
            requestRender();
    }

    mPreviousX = x;
    mPreviousY = y;
    return true;
}
```

注意在計算旋轉角度後，該方法會調用`requestRender()`來告訴渲染器現在可以進行渲染了。這種辦法對於這個例子來說是最有效的，因為圖形並不需要重新繪製，除非有一個旋轉角度的變化。當然，為了能夠真正實現執行效率的提高，記得使用`setRenderMode()`方法以保證渲染器僅在數據發生變化時纔會重新繪製圖形，所以請確保這一行代碼沒有被註釋掉：

```
public MyGLSurfaceView(Context context) {  
    ...  
    // Render the view only when there is a change in the drawing data  
    setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);  
}
```

## 公開旋轉角度

上述樣例代碼需要我們公開旋轉的角度，具體來說，是在渲染器中添加一個 `public` 成員變量。由於渲染器代碼運行在一個獨立的線程中（非主UI線程），我們必須同時將該變量聲明為 `volatile`。注意下面聲明該變量的代碼，另外對應的 `get` 和 `set` 方法也被聲明為 `public` 成員函數：

```
public class MyGLRenderer implements GLSurfaceView.Renderer {  
    ...  
  
    public volatile float mAngle;  
  
    public float getAngle() {  
        return mAngle;  
    }  
  
    public void setAngle(float angle) {  
        mAngle = angle;  
    }  
}
```

## 應用旋轉

為了應用觸控輸入所生成的旋轉，註釋掉創建旋轉角度的代碼，然後添加 `mAngle`，該變量包含了觸控輸入所生成的角度：

```
public void onDrawFrame(GL10 gl) {  
    ...  
    float[] scratch = new float[16];  
  
    // Create a rotation for the triangle  
    // long time = SystemClock.uptimeMillis() % 4000L;  
    // float angle = 0.090f * ((int) time);  
    Matrix.setRotateM(mRotationMatrix, 0, mAngle, 0, 0, -1.0f);  
  
    // Combine the rotation matrix with the projection and camera view  
    // Note that the mMVPMatrix factor *must be first* in order  
    // for the matrix multiplication product to be correct.  
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix, 0);  
  
    // Draw triangle  
    mTriangle.draw(scratch);  
}
```

當完成了上述步驟，我們就可以運行這個程序，並通過手指在屏幕上的滑動旋轉三角形了：



# 添加動畫

---

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/index.html>

動畫為你 App 上將要發生的事件增加精細的視覺提示，並且能改進你 App 界面的思維模型。當界面改變其狀態時，例如加載內容或新操作可用時，動畫特別有幫助。動畫也能為你的 App 增添優雅的外觀，提供給你的 App 一種優質體驗。

但是記住：濫用動畫或者在錯誤時機使用動畫也是有害的，例如：他們造成了延遲。這節課程告訴你如何應用常用動畫類型來提升易用性，在不給用戶增加煩惱的前提下提升性能。

## Sample Code

---

[Animations.zip](#)

## Lessons

---

- [View間漸變](#)

學習在重疊 view 間的淡入淡出。這節課將展示給你如何使用一個進度提示來淡入淡出包含文本信息的 view。

- [使用ViewPager實現屏幕滑動](#)

學習怎樣為水平相鄰的界面提供漸變動畫。

- [展示卡片（Card）翻轉動畫](#)

學習怎樣實現兩個視圖的翻轉動畫。

- [縮放View](#)

學習怎樣通過觸控放大視圖。

- [佈局變更動畫](#)

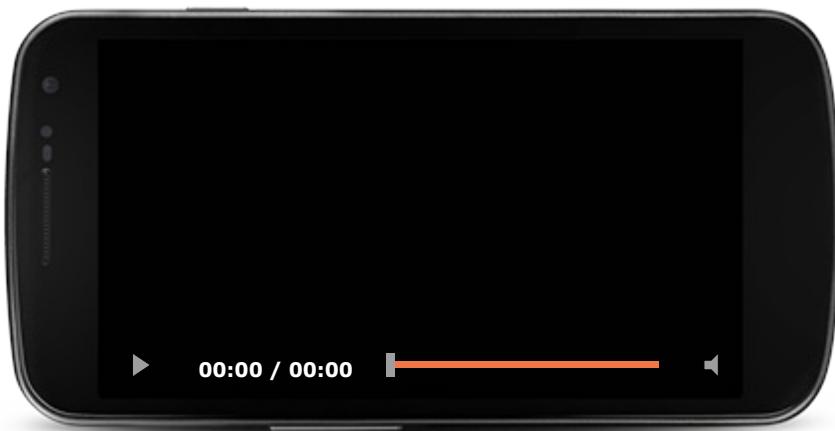
學習當你增加、移除或更新子 View 時怎樣使用內置動畫。

# View間漸變

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/crossfade.html>

漸變動畫（也叫消失）通常指漸漸的淡出某個 UI 組件，同時同步地淡入另一個。在你 App 想切換內容或 view的情況下，這種動畫很有用。漸變簡短不易察覺，它也能提供從一個界面到下一個之間流暢的轉換。當你不使用它們，不管怎麼樣轉換經常感到生硬而倉促。

下面是一個利用進度指示漸變到一些文本內容的例子。



如果你想跳過看整個例子，[下載](#) App 樣例然後運行漸變例子。查看下列文件中的代碼實現：

- src/CrossfadeActivity.java
- layout/activity\_crossfade.xml
- menu/activity\_crossfade.xml

## 創建View

創建兩個你想相互漸變的 view。下面的例子創建了一個進度提示圈和可滑動文本 view。

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView style="?android:textAppearanceMedium"
            android:lineSpacingMultiplier="1.2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/lorem_ipsum"
            android:padding="16dp" />

    </ScrollView>

    <ProgressBar android:id="@+id/loading_spinner"
        style="?android:progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_gravity="center" />
```

```
</FrameLayout>
```

## 設置動畫

為設置動畫，你需要：

1. 為你想漸變的 view 創建成員變量。之後動畫應用途中修改 View 的時候你會需要這些引用的。
2. 對於被淡出的 view，設置它的 visibility 為 `GONE`。這樣防止 view 再佔據佈局的空間，而且也能在佈局計算中將其忽略，加速處理過程。
3. 將 `config_shortAnimTime` 系統屬性暫存到一個成員變量裏。這個屬性為動畫定義了一個標準的“短”持續時間。對於微妙或者快速發生的動畫，這是個很理想的時間段。`config_longAnimTime` 和 `config_mediumAnimTime` 也行，如果你想用的話。

下面是個內容 view 的 `activity` 例子，它使用了前面所述代碼片段中的佈局。

```
public class CrossfadeActivity extends Activity {  
  
    private View mContentView;  
    private View mLoadingView;  
    private int mShortAnimationDuration;  
  
    ...  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_crossfade);  
  
        mContentView = findViewById(R.id.content);  
        mLoadingView = findViewById(R.id.loading_spinner);  
  
        // 初始化隱藏這個View.  
        mContentView.setVisibility(View.GONE);  
  
        // 獲取並緩存系統默認的“短”時長  
        mShortAnimationDuration = getResources().getInteger(  
            android.R.integer.config_shortAnimTime);  
    }  
}
```

## 漸變View

既然正確地設置了那些 view，做下面這些事情來漸變他們吧：

1. 對於淡入的 view，設置 alpha 值為 0 並且設置 visibility 為 `VISIBLE`（要記得他起初被設置成了 `GONE`）。這讓 view 可見了但是它是透明的。
2. 對於淡入的 view，把 alpha 值從 0 動態改變到 1。同時，對於淡出的 view，把 alpha 值從 1 動態變到 0。
3. 使用 `Animator.AnimatorListener` 中的 `onAnimationEnd()`，設置淡出 view 的 visibility 為 `GONE`。即使 alpha 值為 0，也要把 view 的 visibility 設置成 `GONE` 來防止 view 佔據佈局空間，還能把它從佈局計算中忽略，加速處理過程。

下面方法展示如何去做：

```
private View mContentView;
private View mLoadingView;
private int mShortAnimationDuration;

...
private void crossfade() {

    // 設置內容View為0%的不透明度，但是狀態為“可見”，
    // 因此在動畫過程中是一直可見的（但是為全透明）。
    mContentView.setAlpha(0f);
    mContentView.setVisibility(View.VISIBLE);

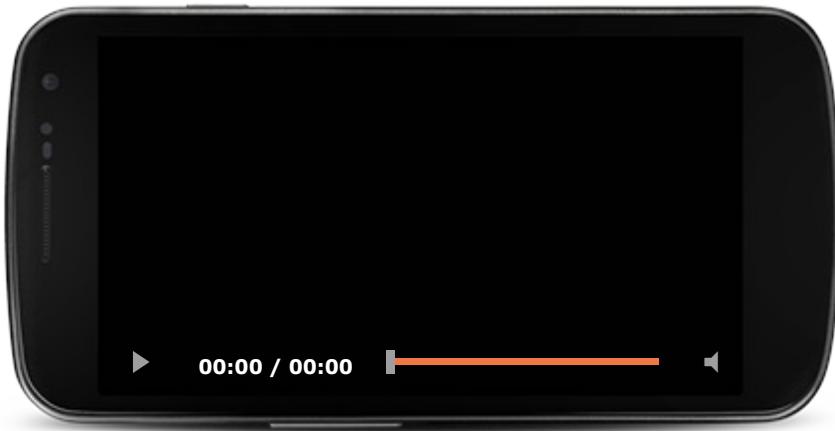
    // 開始動畫內容View到100%的不透明度，然後清除所有設置在View上的動畫監聽器。
    mContentView.animate()
        .alpha(1f)
        .setDuration(mShortAnimationDuration)
        .setListener(null);

    // 加載View開始動畫逐漸變為0%的不透明度，
    // 動畫結束後，設置可見性為GONE（消失）作為一個優化步驟
    //（它將不再參與佈局的傳遞等過程）
    mLoadingView.animate()
        .alpha(0f)
        .setDuration(mShortAnimationDuration)
        .setListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                mLoadingView.setVisibility(View.GONE);
            }
        });
}
```

# 使用ViewPager實現屏幕側滑

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/screen-slide.html>

滑屏是在兩個完整界面間的轉換，它在一些 UI 中很常見，比如設置導向和幻燈放映。這節課將告訴你怎樣通過支持庫（support library）提供的ViewPager實現滑屏。ViewPager能自動實現滑屏動畫。下面展示了從一個內容界面到一下界面的滑屏轉換是什麼樣子。



如果你想跳過看整個例子，[下載](#) App 樣例然後運行屏幕滑動例子。查看下列文件中的代碼實現：

- src/ScreenSlidePageFragment.java
- src/ScreenSlideActivity.java
- layout/activity\_screen\_slide.xml
- layout/fragment\_screen\_slide\_page.xml

## 創建View

創建你之後會為 Fragment 內容使用的佈局文件。下面例子包含一個展示文本的 text view：

```
<!-- fragment_screen_slide_page.xml -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/content"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView style="?android:textAppearanceMedium"
        android:padding="16dp"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/lorem_ipsum" />
</ScrollView>
```

同時也為 fragment 內容定義了一個字符串。

## 創建Fragment

創建一個 Fragment 子類，它從 `onCreateView()` 方法中返回你剛創建的佈局。無論什麼時候，如果你需要為用戶展示

一個頁面，你可以在它的父 [activity](#) 中為這個 fragment 新建實例：

```
import android.support.v4.app.Fragment;
...
public class ScreenSlidePageFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ViewGroup rootView = (ViewGroup) inflater.inflate(
            R.layout.fragment_screen_slide_page, container, false);

        return rootView;
    }
}
```

## 添加ViewPager

[ViewPager](#) 有內建的滑動手勢用來在頁面間轉換，並且他默認使用滑屏動畫，所以你不用自己創建。[ViewPager](#) 使用 [PagerAdapter](#) 來補充新頁面，所以 [PagerAdapter](#) 會用到你之前新建 fragment 類。

開始之前，創建一個包含 [ViewPager](#) 的佈局：

```
<!-- activity_screen_slide.xml -->
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

創建一個 [activity](#) 做下面這些事情：

- 把 content view 設置成這個包含 [ViewPager](#) 的佈局。
- 創建一個繼承自 [FragmentStatePagerAdapter](#) 抽象類的類，然後實現 [getItem\(\)](#) 方法來把 [ScreenSlidePageFragment](#) 實例作為新頁面補充進來。pager adapter還需要實現 [getCount\(\)](#) 方法，它返回 adapter 將要創建頁面的總數（例如5個）。
- 把 [PagerAdapter](#) 和 [ViewPager](#) 關聯起來。
- 處理 Back 按鈕，按下變為在虛擬的 fragment 樣中回退。如果用戶已經在第一個頁面了，則在 [activity](#) 的 back stack 中回退。

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
...
public class ScreenSlidePagerAdapterActivity extends FragmentActivity {
    /**
     * 本演示代碼的顯示頁數（看作嚮導步驟）。
     */
    private static final int NUM_PAGES = 5;

    /**
     * Pager控件，用來處理動畫並允許整頁橫向滑動來展示上一個和下一個“嚮導步驟”
     */
    private ViewPager mPager;
}
```

```

    * Pager適配器，用於提供ViewPager控件的具體頁面。
    */
    private PagerAdapter mPagerAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_screen_slide);

        // 初始化 ViewPager 和 PagerAdapter.
        mPager = (ViewPager) findViewById(R.id.pager);
        mPagerAdapter = new ScreenSlidePagerAdapter(getSupportFragmentManager());
        mPager.setAdapter(mPagerAdapter);
    }

    @Override
    public void onBackPressed() {
        if (mPager.getCurrentItem() == 0) {
            // 如果用戶當前正在看第一步（也就是第一頁），那就要讓系統來處理返回按鈕。
            // 這個是結束（finish()）當前活動並彈出回退棧。
            super.onBackPressed();
        } else {
            // 否則，返回前一頁
            mPager.setCurrentItem(mPager.getCurrentItem() - 1);
        }
    }

    /**
     * 一個簡單的Pager控件適配器，它順序地展示了5個ScreenSlidePageFragment對象
     */
    private class ScreenSlidePagerAdapter extends FragmentStatePagerAdapter {
        public ScreenSlidePagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public Fragment getItem(int position) {
            return new ScreenSlidePageFragment();
        }

        @Override
        public int getCount() {
            return NUM_PAGES;
        }
    }
}

```

## 用PageTransformer自定義動畫

為展示不同於默認滑屏效果的動畫，實現 `ViewPager.PageTransformer` 接口，然後把它補充到 view pager 裏。這接口只暴露了一個方法，`transformPage()`。每次界面切換，這個方法都會為每個可見頁面和界面中消失的相鄰界面調用一次（通常只有一個頁面可見）。例如，第三頁可見而且用戶向第四頁拖動，`transformPage()` 在手勢的各個階段為第二，三，四頁分別調用。

在你 `transformPage()` 的實現中，基於當前界面上頁面的 `position`（`position` 由 `transformPage()` 方法的參數給出）決定哪些頁面需要被動畫轉換，通過這樣你就能新建自己的動畫。

`position` 參數表示給定頁面相對於屏幕中的頁面的位置。它的值在用戶滑動頁面過程中動態變化。當頁面填充屏幕，它的值為 0。當頁面剛從屏幕右邊拖走，它的值為 1。如果用戶在頁面一和頁面二間滑動到一半，那麼頁面一的 `position` 為 -0.5 並且頁面二的 `position` 為 0.5。根據屏幕上頁面的 `position`，你可以通過 `setAlpha()`，`setTranslationX()` 或 `setScaleY()` 這些方法設定頁面屬性來自定義滑動動畫。

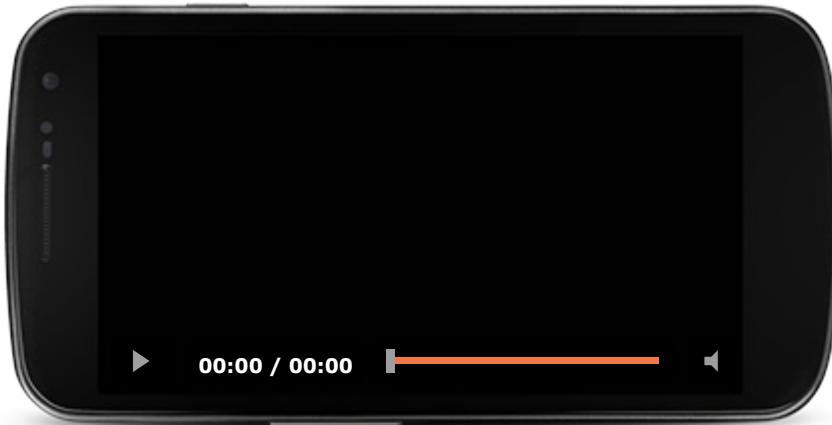
當你實現了 `PageTransformer` 後，用你的實現調用 `setPageTransformer()` 來應用這些自定義動畫。例如，如果你有一個叫做 `ZoomOutPageTransformer` 的 `PageTransformer`，你可以這樣設置自定義動畫：

```
ViewPager mPager = (ViewPager) findViewById(R.id.pager);
...
mPager.setPageTransformer(true, new ZoomOutPageTransformer());
```

詳情查看[縮放型 Page Transformer](#)和[潛藏型 Page Transformer](#)部分和 [PageTransformer](#) 視頻。

## 縮放PageTransformer

當在相鄰界面滑動時，這個page transformer使頁面收縮並褪色。當頁面越靠近中心，它將漸漸還原到正常大小並且圖像漸明。



```
public class ZoomOutPageTransformer implements ViewPager.PageTransformer {
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.5f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();
        int pageHeight = view.getHeight();

        if (position < -1) { // [-∞, -1]
            // 這一頁已經是最左邊的屏幕頁
            view.setAlpha(0);

        } else if (position <= 1) { // [-1, 1]
            // 修改默認的滑動過渡效果為縮放效果
            float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(position));
            float vertMargin = pageHeight * (1 - scaleFactor) / 2;
            float horzMargin = pageWidth * (1 - scaleFactor) / 2;
            if (position < 0) {
                view.setTranslationX(horzMargin - vertMargin / 2);
            } else {
                view.setTranslationX(-horzMargin + vertMargin / 2);
            }

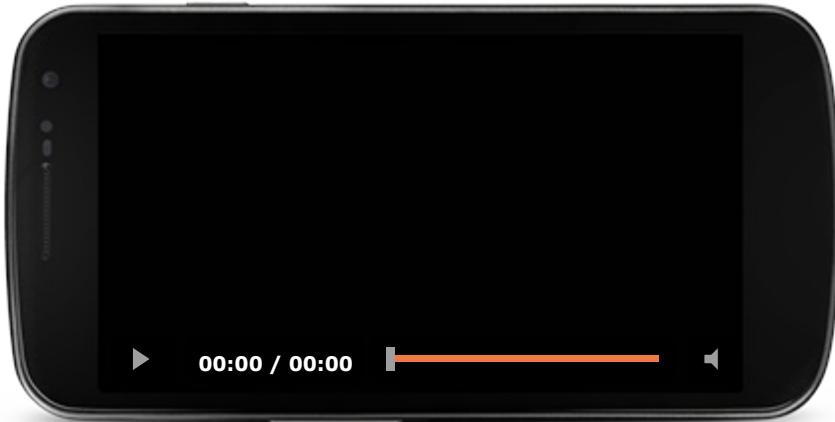
            // 開始根據縮放係數進行變化 (在 MIN_SCALE 和 1 之間變化)
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

            // 根據大小 (縮放係數) 變化化透明度實現淡化頁面效果
            view.setAlpha(MIN_ALPHA +
                (scaleFactor - MIN_SCALE) /
                (1 - MIN_SCALE) * (1 - MIN_ALPHA));

        } else { // (1, +∞ ]
            // 這一頁已經是最右邊的屏幕頁
            view.setAlpha(0);
        }
    }
}
```

## 潛藏型PageTransformer（頁面轉換動畫）

這個page transformer使用默認動畫的屏幕左滑動畫。但是為右滑使用一種“潛藏”效果的動畫。潛藏動畫淡出頁面，並且線性縮小它。



注意：在潛藏過程中，默認動畫（屏幕滑動）是仍舊發生的，所以你必須用負的 X平移來抵消它。例如：

```
view.setTranslationX(-1 * view.getWidth() * position);
```

下面的例子展示瞭如何抵消默認滑屏動畫：

```
public class DepthPageTransformer implements ViewPager.PageTransformer {
    private static final float MIN_SCALE = 0.75f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();

        if (position < -1) { // [-∞, -1]
            // 這一頁已經是最左邊的屏幕頁
            view.setAlpha(0);

        } else if (position <= 0) { // [-1, 0]
            // 向左面滑屏使用默認的過渡動畫
            view.setAlpha(1);
            view.setTranslationX(0);
            view.setScaleX(1);
            view.setScaleY(1);

        } else if (position <= 1) { // (0, 1]
            // 淡出頁面
            view.setAlpha(1 - position);

            // 抵消默認的整頁過渡
            view.setTranslationX(pageWidth * -position);

            // 根據縮放係數變化 (在 MIN_SCALE 和 1 之間變化)
            float scaleFactor = MIN_SCALE
                + (1 - MIN_SCALE) * (1 - Math.abs(position));
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);

        } else { // (1, +∞]
            // 這一頁已經是最右邊的屏幕頁
            view.setAlpha(0);
        }
    }
}
```



# 展示卡片翻轉動畫

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/cardflip.html>

這節課展示如何使用自定義 fragment 動畫實現卡片翻轉動畫。通過展示一個模擬卡片翻轉的動畫實現 view 內容的卡片翻轉效果。

下面是卡片翻轉動畫的樣子：



如果你想跳過看整個例子，[下載 App 樣例](#)然後運行卡片翻轉例子。查看下列文件中的代碼實現：

- src/CardFlipActivity.java
- animator/card\_flip\_right\_in.xml
- animator/card\_flip\_right\_out.xml
- animator/card\_flip\_left\_in.xml
- animator/card\_flip\_left\_out.xml
- layout/fragment\_card\_back.xml
- layout/fragment\_card\_front.xml

## 創建Animator

創建卡片翻轉動畫，你需要兩個 animator 讓前面的卡片向右翻轉消失，向左翻轉出現。你還需要兩個 animator 讓背面的卡片向左翻轉出現，向右翻轉消失。

card\_flip\_left\_in.xml

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

<!--旋轉之前，立刻設置透明度alpha為0-->
<objectAnimator
    android:valueFrom="1.0"
    android:valueTo="0.0"
    android:propertyName="alpha"
    android:duration="0" />

<!--旋轉-->
<objectAnimator
    android:valueFrom="-180"
    android:valueTo="0"
    android:propertyName="rotationY"
```

```
    android:interpolator="@android:interpolator/accelerate_decelerate"
    android:duration="@integer/card_flip_time_full" />

    <!--旋轉中途(時間偏移量取決於startOffset屬性)設置透明度為1-->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

### card\_flip\_left\_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 旋轉. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!--旋轉中途(時間偏移量取決於startOffset屬性)設置透明度為0-->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

### card\_flip\_right\_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!--旋轉之前，立刻設置透明度alpha為0-->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- 旋轉. -->
    <objectAnimator
        android:valueFrom="180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!--旋轉中途(時間偏移量取決於startOffset屬性)設置透明度為1-->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
```

### card\_flip\_right\_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 旋轉. -->
    <objectAnimator
        android:valueFrom="0"
```

```
    android:valueTo="-180"
    android:propertyName="rotationY"
    android:interpolator="@android:interpolator/accelerate_decelerate"
    android:duration="@integer/card_flip_time_full" />

    <!--旋轉中途(時間偏移量取決於startOffset屬性)設置透明度為0-->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

## 創建View

卡片的每一面是一個獨立包含你想要內容的佈局，比如兩屏文字，兩張圖片，或者任何view的組合。然後你將在應用動畫的fragment裏面用到這倆佈局。下面的佈局創建了卡片展示文本一面的佈局：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#a6c"
    android:padding="16dp"
    android:gravity="bottom">

    <TextView android:id="@+id/text1"
        style="?android:textAppearanceLarge"
        android:textStyle="bold"
        android:textColor="#fff"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_title" />

    <TextView style="?android:textAppearanceSmall"
        android:textAllCaps="true"
        android:textColor="#80ffffffff"
        android:textStyle="bold"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/card_back_description" />

</LinearLayout>
```

卡片另一面顯示一個 ImageView：

```
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/image1"
    android:scaleType="centerCrop"
    android:contentDescription="@string/description_image_1" />
```

## 創建Fragment

為卡片正反面創建fragment，這些類從 [onCreateView\(\)](#) 方法中分別為每個fragment返回你之前創建的佈局。在父[activity](#)中，你可以在你想要顯示卡片時創建對應的 fragment 實例。下面的例子展示父[activity](#)內嵌套的fragment：

```

public class CardFlipActivity extends Activity {
    ...
    /**
     * 一個呈現在卡片前方的fragment
     */
    public class CardFrontFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                               Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_card_front, container, false);
        }
    }

    /**
     * 一個呈現在卡片後方的fragment
     */
    public class CardBackFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                               Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_card_back, container, false);
        }
    }
}

```

## 應用卡片翻轉動畫

現在，你需要在父activity中展示fragment。為做這件事，首先創建你activity的佈局。下面例子創建了一個你可以在運行時添加fragment的 FrameLayout。

```

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

在activity代碼中，把剛創建的佈局設置成content view。當activity創建時展示一個默認的fragment是個不錯的注意。所以下面的activity樣例告訴你如何默認顯示卡片正面：

```

public class CardFlipActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_card_flip);

        if (savedInstanceState == null) {
            getFragmentManager()
                .beginTransaction()
                .add(R.id.container, new CardFrontFragment())
                .commit();
        }
    }
    ...
}

```

既然現在顯示了卡片的正面，你可以在合適時機用翻轉動畫顯示卡片背面了。創建一個方法來顯示背面需要做下面這些事情：

- 為fragment轉換設置你剛做的自定義動畫

- 用新fragment替換當前顯示的fragment，並且應用你剛創建的動畫到這個事件中。
- 添加之前顯示的fragment到fragment的back stack中，所以當用戶摁 Back 鍵時，卡片會翻轉回來。

```

private void flipCard() {
    if (mShowingBack) {
        getSupportFragmentManager().popBackStack();
        return;
    }

    // 向後翻轉.

    mShowingBack = true;

    // 創建並提交一個新的Fragment事務用於在卡片後面添加Fragment，使用自定義動畫，並且加入
    // Fragment管理器回退棧
    getSupportFragmentManager()
        .beginTransaction()

    // 用動畫器資源呈現卡片自前向後的旋轉效果替換默認的Fragment動畫，
    // 當翻轉到前面的時候動畫器資源也可以呈現自後向前的旋轉效果（例如按下系統返回鍵時）
    .setCustomAnimations(
        R.animator.card_flip_right_in, R.animator.card_flip_right_out,
        R.animator.card_flip_left_in, R.animator.card_flip_left_out)

    // 用一個Fragment替換任何當前在容器佈局內的Fragment來呈現下一本頁
    // (通過僅自增的變量currentPage來表示)
    .replace(R.id.container, new CardBackFragment())

    // 添加這個事務到回退棧，允許用戶來按下返回按鈕來回退到卡片正面.
    .addToBackStack(null)

    // 提交完成事務.
    .commit();
}

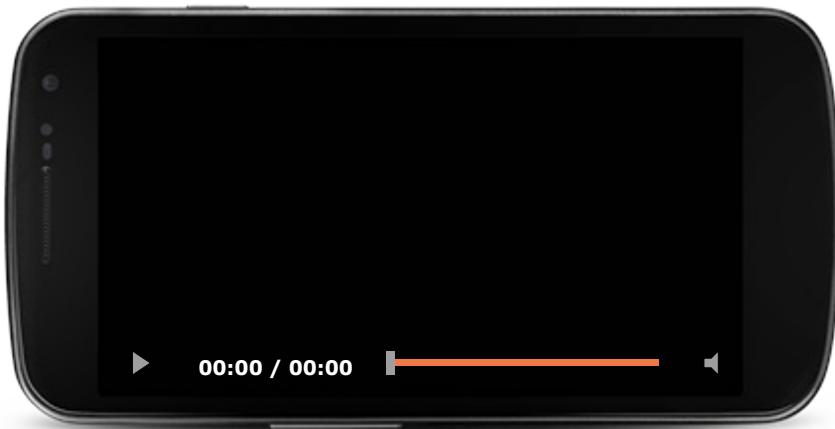
```

# 縮放View

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/zoom.html>

這節課示範怎樣實現點擊縮放動畫，這對相冊很有用，他能允許相片從縮略圖轉換成原圖並填充屏幕提供動畫。

下面展示了觸摸縮放動畫效果是什麼樣子，它將縮略圖擴大並填充屏幕。



如果你想跳過看整個例子，[下載 App 樣例](#)然後運行縮放的例子。查看下列文件中的代碼實現：

- src/TouchHighlightImageButton.java (簡單的helper類，當image button被按下它顯示藍色高亮)
- src/ZoomActivity.java
- layout/activity\_zoom.xml

# 創建View

爲你想縮放的內容創建一大一小兩個版本佈局文件。下面的例子爲可點擊的縮略圖新建了一個 [ImageButton](#) 和一個 [ImageView](#) 來展示原圖：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <ImageButton
            android:id="@+id/thumb_button_1"
            android:layout_width="100dp"
            android:layout_height="75dp"
            android:layout_marginRight="1dp"
            android:src="@drawable/thumb1"
            android:scaleType="centerCrop"
            android:contentDescription="@string/description_image_1" />

    </LinearLayout>

    <!-- 這個初始化狀態爲隱藏的ImageView將會持有一個擴大/縮放版本的圖片，並且浮於佈局上層，-->
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/placeholder" />

```

```
-->

<ImageView
    android:id="@+id/expanded_image"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="invisible"
    android:contentDescription="@string/description_zoom_touch_close" />

</FrameLayout>
```

## 設置縮放動畫

一旦實現了佈局，你需要設置觸發縮放事件handler。下面的例子為`ImageButton`添加了一個`View.OnClickListener`，當用戶點擊按鈕時它執行放大動畫。

```
public class ZoomActivity extends FragmentActivity {
    // 持有一個當前 animator 的引用,
    // 以後以便於中途取消動畫。
    private Animator mCurrentAnimator;

    // 這個系統內的“短”動畫時長是以毫秒為單位的。
    // 這個時長對於精確控制的動畫或頻繁激發的動畫是非常理想的。
    private int mShortAnimationDuration;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_zoom);

        // 為縮略圖連結點擊事件
        final View thumb1View = findViewById(R.id.thumb_button_1);
        thumb1View.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                zoomImageFromThumb(thumb1View, R.drawable.image1);
            }
        });

        // 獲取並緩存系統默認定義的“短”動畫時長
        mShortAnimationDuration = getResources().getInteger(
            android.R.integer.config_shortAnimTime);
    }
    ...
}
```

## 縮放View

你現在需要適時應用放大動畫了。通常來說，你需要按邊界來從小號View放大到大號View。下面的方法告訴你如何實現縮放動畫：

1. 把高清圖像設置到“放大版”隱藏的`ImageView`中。為表簡單，下面的例子在UI線程中加載了一張大圖。但是你需要在一個單獨的線程中來加載以免阻塞UI線程，然後再回到UI線程中設置。理想狀況下，圖片不要大過屏幕。
2. 計算`ImageView`開始和結束時的邊界。
3. 同步地動態改變四個位置和大小屬性`X`, `Y` (`SCALE_X` 和 `SCALE_Y`)，從起始點到結束點。這四個動畫被加入到了`AnimatorSet`，所以你可以一起開始。
4. 縮回則運行相同的動畫，但是是用戶點擊屏幕放大時的逆向效果。你可以在`ImageView`中添加一

個View.OnClickListener來實現它。當點擊時，ImageView縮回到原來縮略圖的大小，然後設置它的visibility為GONE來隱藏。

```
private void zoomImageFromThumb(final View thumbView, int imageResId) {
    //如果一個動畫正在進行過程中，那麼就要立即取消之前的動畫並進行這一個。
    if (mCurrentAnimator != null) {
        mCurrentAnimator.cancel();
    }

    // 載入一個高分辨率的所謂 "已放大" 的圖片。
    final ImageView expandedImageView = (ImageView) findViewById(
        R.id.expanded_image);
    expandedImageView.setImageResource(imageResId);

    // 為放大的圖片計算開始動畫和結束動畫的矩形邊界
    // 這個步驟牽扯到了大量的數學計算，YEAH!!坑爹的數學!!
    final Rect startBounds = new Rect();
    final Rect finalBounds = new Rect();
    final Point globalOffset = new Point();

    // 動畫開始的邊界是縮略圖對全局可見的矩形，最終的邊界是外部包裹的佈局可見矩形。
    // 這裏還設置了包裹視圖的偏移量為原點的邊界，因為這是原點為定位的動畫屬性(X, Y)。
    thumbView.getGlobalVisibleRect(startBounds);
    findViewById(R.id.container).getGlobalVisibleRect(finalBounds, globalOffset);
    startBounds.offset(-globalOffset.x, -globalOffset.y);
    finalBounds.offset(-globalOffset.x, -globalOffset.y);

    // 調整開始邊界要和使用了"centerCrop"技術的最終邊界保持相同的縱橫比。
    // 這可以在動畫過程中防止不希望出現的拉伸現象。還計算了開始大小的縮放係數
    // (結束大小的係數則一直為1.0)
    float startScale;
    if ((float) finalBounds.width() / finalBounds.height()
        > (float) startBounds.width() / startBounds.height()) {
        // 水平擴展開始邊界
        startScale = (float) startBounds.height() / finalBounds.height();
        float startWidth = startScale * finalBounds.width();
        float deltaWidth = (startWidth - startBounds.width()) / 2;
        startBounds.left -= deltaWidth;
        startBounds.right += deltaWidth;
    } else {
        // 豐直擴展開始邊界
        startScale = (float) startBounds.width() / finalBounds.width();
        float startHeight = startScale * finalBounds.height();
        float deltaHeight = (startHeight - startBounds.height()) / 2;
        startBounds.top -= deltaHeight;
        startBounds.bottom += deltaHeight;
    }

    // 隱藏縮略圖並顯示放大後的View。當動畫開始，將在縮略圖的位置定位放大的視圖
    thumbView.setAlpha(0f);
    expandedImageView.setVisibility(View.VISIBLE);

    // 設置錨點，以放大後的view左上角座標為準來準備 SCALE_X 和 SCALE_Y 變換
    // (默認為View的中心)
    expandedImageView.setPivotX(0f);
    expandedImageView.setPivotY(0f);

    // 構建並並行化運行4個平移動畫和縮放屬性(X, Y, SCALE_X, and SCALE_Y)
    AnimatorSet set = new AnimatorSet();
    set
        .play(ObjectAnimator.ofFloat(expandedImageView, View.X,
            startBounds.left, finalBounds.left))
        .with(ObjectAnimator.ofFloat(expandedImageView, View.Y,
            startBounds.top, finalBounds.top))
        .with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE_X,
            startScale, 1f)).with(ObjectAnimator.ofFloat(expandedImageView,
            View.SCALE_Y, startScale, 1f));
    set.setDuration(mShortAnimationDuration);
    set.setInterpolator(new DecelerateInterpolator());
    set.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
```

```
        mCurrentAnimator = null;
    }

    @Override
    public void onAnimationCancel(Animator animation) {
        mCurrentAnimator = null;
    }
});

set.start();
mCurrentAnimator = set;

// 點擊放大後的圖片，應該是縮收回原來的邊界並顯示縮略圖
// 而不是顯示擴大的圖
final float startScaleFinal = startScale;
expandedImageView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mCurrentAnimator != null) {
            mCurrentAnimator.cancel();
        }

        // 開始並行動畫這四個位置/大小屬性，直到歸至原始值。
        AnimatorSet set = new AnimatorSet();
        set.play(ObjectAnimator
            .ofFloat(expandedImageView, View.X, startBounds.left))
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.Y,startBounds.top))
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.SCALE_X, startScaleFinal))
            .with(ObjectAnimator
                .ofFloat(expandedImageView,
                    View.SCALE_Y, startScaleFinal));
        set.setDuration(mShortAnimationDuration);
        set.setInterpolator(new DecelerateInterpolator());
        set.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                thumbView.setAlpha(1f);
                expandedImageView.setVisibility(View.GONE);
                mCurrentAnimator = null;
            }

            @Override
            public void onAnimationCancel(Animator animation) {
                thumbView.setAlpha(1f);
                expandedImageView.setVisibility(View.GONE);
                mCurrentAnimator = null;
            }
        });
        set.start();
        mCurrentAnimator = set;
    }
});
}
```

# 佈局變更動畫

編寫:XizhiXu - 原文:<http://developer.android.com/training/animation/layout.html>

佈局動畫是一種預加載動畫，系統在你每次改變的佈局配置時運行它。你需要做的僅是在佈局文件裏設置屬性告訴Android系統為你這些佈局的變更應用動畫。系統動畫時為你默認執行的。

小貼士: 如果你想補充自定義佈局動畫，創建 `LayoutTransition` 對象和然後用 `setLayoutTransition()` 方法把它加到佈局中。

下面是在一個list中添加一項的默認佈局動畫：



如果你想跳過看整個例子，[下載 App 樣例](#)然後運行佈局漸變的例子。查看下列文件中的代碼實現：

1. `src/LayoutChangesActivity.java`
2. `layout/activity_layout_changes.xml`
3. `menu/activity_layout_changes.xml`

## 創建佈局

在你[activity](#)的XML佈局文件中，為你想開啓動畫的佈局設置 `android:animateLayoutChanges` 屬性為真。例如：

```
<LinearLayout android:id="@+id/container"
    android:animateLayoutChanges="true"
    ...
/>
```

## 從佈局中添加，更新或刪除項目

現在，你需要做的就是添加，刪除或更新布局裏的項目，然後這些項目就會自動顯示動畫啦：

```
private ViewGroup mContainerView;
...
private void addItem() {
    View newView;
    ...
}
```

```
    mContainerView.addView(newView, 0);  
}
```

# Android網絡連接與雲服務

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/building-connectivity.html>

這些課程教你如何讓你的app連接到用戶設備之外的世界.你將會學習如何連接到這個區域的其他設備,連接到互聯網,以及備份和同步應用程序數據等等.

## 無線連接設備 - Connecting Devices Wirelessly

如何使用網絡服務發現(Network Service Discovery)找到並連接當地設備,以及如何用Wi-Fi創建點對點連接.

## 執行網絡操作 - Performing Network Operations

如何創建一個網絡連接,監視連接的變化,以及使用XML數據執行事務.

## 高效下載 - Transferring Data Without Draining the Battery

如何在你的app執行下載和其他網絡事務時最小化對電池的影響.

## 雲同步 - Syncing to the Cloud

如何同步和備份應用程序和用戶數據到雲中的遠程web服務以及如何恢復數據到多個設備.

## 解決雲同步的保存衝突 : Resolving Cloud Save Conflicts

如何為app設計一個健壯的存儲數據到雲的衝突解決策略.

## 使用Sync Adapter傳輸數據 - Transferring Data Using Sync Adapters

如何使用Android sync adapter框架在雲和設備間傳輸數據.

## 使用Volley傳輸網絡數據 - Transmitting Network Data Using Volley

如何使用Volley通過網絡執行快速可擴展的UI操作.

# 無線連接設備

---

編寫:acenodie - 原文:<http://developer.android.com/training/connect-devices-wirelessly/index.html>

除了能夠在雲端通信，Android的無線接口（wireless APIs）也允許同一局域網中的設備進行通信，甚至沒有連接到網絡上，而是物理上隔得很近，也可以相互通信。此外，網絡服務發現（Network Service Discovery，簡稱NSD）可以進一步通過允許應用程序運行能相互通信的服務去尋找附近運行相同服務的設備。把這個功能整合到你的應用中可以提供許多功能，如在同一個房間，用戶玩遊戲，可以利用NSD實現從一個網絡攝像頭獲取圖像，或遠程登錄到在同一網絡中的其他機器。

本節課介紹了一些使你的應用程序能夠尋找和連接其他設備的主要的接口（APIs）。具體地說，它介紹了用於發現可用服務的NSD API和能實現點對點無線連接的無線點對點（the Wi-Fi Peer-to-Peer，簡稱Wi-Fi P2P）API。本節課也將告訴我們怎樣將NSD和Wi-Fi P2P結合起來去檢測其他設備所提供的服務，當檢測到時，連接到相應的設備上。即使設備都沒有連接到一個網絡中。

## Lessons

---

- [使用網絡服務發現](#)

學習如何廣播由你自己的應用程序提供的服務，如何發現在本地網絡上提供的服務並用NSD獲取你將要連接的服務的詳細信息。

- [使用WiFi建立P2P連接](#)

學習如何獲取附近的對等設備，如何創建一個設備接入點，如何連接到其他具有Wi-Fi P2P連接功能的設備。

- [使用WiFi P2P發現服務](#)

學習如何使用WiFi P2P服務去發現附近的不在同一個網絡的服務。

# 使得網絡服務可發現

編寫:[naizhengtan](#) - 原文:<http://developer.android.com/training/connect-devices-wirelessly/nsd.html>

添加網絡服務發現（Network Service Discovery）到你的app中可以使你的用戶辨識在局域網且支持你的app所請求的服務的設備。這種技術在端對端應用中能夠提供大量幫助，例如文件共享、聯機遊戲等。Android提供了網絡服務發現（NSD）相應的API，大大降低了其實現難度。

本講將簡要介紹如何創建NSD應用，使其能夠在本地網絡內廣播自己的名稱和鏈接信息，並且掃描網絡發現其他NSD設備。最後，將介紹如何連接到運行着同樣應用的另一臺設備上。

## 註冊NSD服務

Note：這一步驟是選做的。如果你並不關心自己的服務是否被廣播，你可以跳過這一步，直接嘗試[發現網絡中的服務](#)。

在局域網內註冊自己服務的第一步是創建[NsdServiceInfo](#)對象。此對象包含的信息能夠幫助其他設備決定是否要連接到你所提供的服務。

```
public void registerService(int port) {
    // Create the NsdServiceInfo object, and populate it.
    NsdServiceInfo serviceInfo = new NsdServiceInfo();

    // The name is subject to change based on conflicts
    // with other services advertised on the same network.
    serviceInfo.setServiceName("NsdChat");
    serviceInfo.setServiceType("_http._tcp.");
    serviceInfo.setPort(port);

    ...
}
```

這段代碼將所提供的服務命名為“NsdChat”。該名稱將對所有局域網絡中的設備可見。需要注意的是，在網絡內該名稱必須是獨一無二的。Android系統會自動處理衝突的服務名稱。如果同時有兩個名為“NsdChat”的應用，其中一個會被自動轉換為“NsdChat(1)”。

第二個參數設置了服務類型，即，使用的通信協議和傳輸層協議，語法是“\_< protocol >.\_< transportlayer >”。在上面的代碼中，服務使用了TCP協議上的HTTP協議。如果應用想要提供打印服務（例如，一臺網絡打印機）應該將服務的類型設置為“\_ipp.\_tcp”。

Note: 互聯網編號分配機構（International Assigned Numbers Authority，簡稱IANA）提供用於服務發現協議（例如NSD和Bonjour）的官方服務種類列表。你可以下載該列表瞭解相應的服務名稱和端口號碼。如果你想使用新的服務種類，應該向IANA官方提交申請。

當為你的服務設置端口號時，應該儘量避免將其硬編碼在代碼中，防止與其他應用產生衝突。例如，如果你的應用僅僅使用端口1337，就可能與其他使用1337端口的應用發生衝突。解決方法是，不要硬編碼，使用下一個可用的端口。不必擔心其他應用無法知曉服務的端口號，因為該信息將包含在服務的廣播包中。接收到廣播後，其他應用將從廣播包中得知服務端口號，並通過端口連接到你的服務上。

如果你使用的是socket，你可以將端口號初始值設置為0來使用下一個可用端口。

```
public void initializeServerSocket() {
```

```
// Initialize a server socket on the next available port.  
mServerSocket = new ServerSocket(0);  
  
// Store the chosen port.  
mLocalPort = mServerSocket.getLocalPort();  
...  
}
```

現在，你已經成功的創建了[NsdServiceInfo](#)對象，接下來要做的是實現[RegistrationListener](#)接口。該接口包含了註冊在Android系統中的一系列回調函數，作用是通知應用程序服務註冊/註銷的成功和失敗。

```
public void initializeRegistrationListener() {  
    mRegistrationListener = new NsdManager.RegistrationListener() {  
  
        @Override  
        public void onServiceRegistered(NsdServiceInfo NsdServiceInfo) {  
            // Save the service name. Android may have changed it in order to  
            // resolve a conflict, so update the name you initially requested  
            // with the name Android actually used.  
            mServiceName = NsdServiceInfo.getServiceName();  
        }  
  
        @Override  
        public void onRegistrationFailed(NsdServiceInfo serviceInfo, int errorCode) {  
            // Registration failed! Put debugging code here to determine why.  
        }  
  
        @Override  
        public void onServiceUnregistered(NsdServiceInfo arg0) {  
            // Service has been unregistered. This only happens when you call  
            // NsdManager.unregisterService() and pass in this listener.  
        }  
  
        @Override  
        public void onUnregistrationFailed(NsdServiceInfo serviceInfo, int errorCode) {  
            // Unregistration failed. Put debugging code here to determine why.  
        }  
    };  
}
```

萬事俱備只欠東風，調用[registerService\(\)](#)方法，真正註冊服務。

因為該方法是異步的，所以在服務註冊之後的操作都需要在[onServiceRegistered\(\)](#)方法中進行。

```
public void registerService(int port) {  
    NsdServiceInfo serviceInfo = new NsdServiceInfo();  
    serviceInfo.setServiceName("NsdChat");  
    serviceInfo.setServiceType("_http._tcp.");  
    serviceInfo.setPort(port);  
  
    mNsdManager = Context.getSystemService(Context.NSD_SERVICE);  
  
    mNsdManager.registerService(  
        serviceInfo, NsdManager.PROTOCOL_DNS_SD, mRegistrationListener);  
}
```

## 發現網絡中的服務

網絡充斥着我們的生活，從網絡打印機到網絡攝像頭，再到聯網井字棋。網絡服務發現是能讓你的應用融入這一切功能的關鍵。你的應用需要偵聽網絡內服務的廣播，發現可用的服務，過濾無效的信息。

與註冊網絡服務類似，服務發現需要兩步驟：正確配置發現監聽者(Discover Listener)，以及調用discoverServices()這個異步API。

首先，實例化一個實現NsdManager.DiscoveryListener接口的匿名類。下列代碼是一個簡單的範例：

```
public void initializeDiscoveryListener() {  
  
    // Instantiate a new DiscoveryListener  
    mDiscoveryListener = new NsdManager.DiscoveryListener() {  
  
        // Called as soon as service discovery begins.  
        @Override  
        public void onDiscoveryStarted(String regType) {  
            Log.d(TAG, "Service discovery started");  
        }  
  
        @Override  
        public void onServiceFound(NsdServiceInfo service) {  
            // A service was found! Do something with it.  
            Log.d(TAG, "Service discovery success" + service);  
            if (!service.getServiceType().equals(SERVICE_TYPE)) {  
                // Service type is the string containing the protocol and  
                // transport layer for this service.  
                Log.d(TAG, "Unknown Service Type: " + service.getServiceType());  
            } else if (service.get serviceName().equals(mServiceName)) {  
                // The name of the service tells the user what they'd be  
                // connecting to. It could be "Bob's Chat App".  
                Log.d(TAG, "Same machine: " + mServiceName);  
            } else if (service.get serviceName().contains("NsdChat")){  
                mNsdManager.resolveService(service, mResolveListener);  
            }  
        }  
  
        @Override  
        public void onServiceLost(NsdServiceInfo service) {  
            // When the network service is no longer available.  
            // Internal bookkeeping code goes here.  
            Log.e(TAG, "Service lost" + service);  
        }  
  
        @Override  
        public void onDiscoveryStopped(String serviceType) {  
            Log.i(TAG, "Discovery stopped: " + serviceType);  
        }  
  
        @Override  
        public void onStartDiscoveryFailed(String serviceType, int errorCode) {  
            Log.e(TAG, "Discovery failed: Error code:" + errorCode);  
            mNsdManager.stopServiceDiscovery(this);  
        }  
  
        @Override  
        public void onStopDiscoveryFailed(String serviceType, int errorCode) {  
            Log.e(TAG, "Discovery failed: Error code:" + errorCode);  
            mNsdManager.stopServiceDiscovery(this);  
        }  
    };  
}
```

NSD API通過使用該接口中的方法通知用戶程序何時發現開始、何時發現失敗、何時找到可用服務、何時服務丟失（丟失意味着“不再可用”）。在上述代碼中，當發現了可用的服務時，程序做了幾次檢查。

1. 比較發現的服務名稱與本地的服務，判斷是否發現的是本機的服務（這是合法的）。
2. 檢查服務的類型，確認是否可以接入。
3. 檢查服務的名稱，確認接入了正確的應用。

我們並不需要每次都檢查服務名稱，僅當你想要接入特定的應用時，再去判斷。例如，應用只想與運行在其他設備上的

相同應用通信。然而，如果應用僅僅想接入到一臺網絡打印機，那麼看到服務類型是“\_ipp.\_tcp”的服務就足夠了。

當配置好發現回調函數後，調用`discoverService()`函數，其參數包括試圖發現的服務種類、發現使用的協議、以及上一步創建的監聽者。

```
mNsdManager.discoverServices(  
    SERVICE_TYPE, NsdManager.PROTOCOL_DNS_SD, mDiscoveryListener);
```

## 連接到網絡上的服務

當你的代碼發現了網上可接入的服務，第一件必須做的事情是確認服務的連接信息。調用`resolveService()`方法，並將實現了`NsdManager.ResolveListener`的對象和在服務發現過程中得到的`NsdServiceInfo`對象傳入。通過該方法，可以將連接信息從`NsdServiceInfo`對象中解析出來。

```
public void initializeResolveListener() {  
    mResolveListener = new NsdManager.ResolveListener() {  
  
        @Override  
        public void onResolveFailed(NsdServiceInfo serviceInfo, int errorCode) {  
            // Called when the resolve fails. Use the error code to debug.  
            Log.e(TAG, "Resolve failed" + errorCode);  
        }  
  
        @Override  
        public void onServiceResolved(NsdServiceInfo serviceInfo) {  
            Log.e(TAG, "Resolve Succeeded. " + serviceInfo);  
  
            if (serviceInfo.getServiceName().equals(mServiceName)) {  
                Log.d(TAG, "Same IP.");  
                return;  
            }  
            mService = serviceInfo;  
            int port = mService.getPort();  
            InetAddress host = mService.getHost();  
        }  
    };  
}
```

當服務解析完成後，你將獲得服務的詳細資料，包括其IP地址和端口號。此時，你有了服務的地址和端口，可以通過創建自己網絡連接與服務進行通訊。

## 當程序退出時註銷服務

在應用的生命週期中正確的開啓和關閉NSD服務是十分關鍵的。在程序退出時註銷服務可以防止其他程序因為不知道服務退出而反覆嘗試連接的行為。另外，服務發現是一種開銷很大的操作，應該隨着父`Activity`的暫停而停止，當用戶返回該界面是再開啓。因此，開發者應該重載`Activity`的生命周期函數，並正確操作服務的廣播和發現。

```
//In your application's Activity  
  
@Override  
protected void onPause() {  
    if (mNsdHelper != null) {  
        mNsdHelper.tearDown();  
    }  
    super.onPause();  
}
```

```
@Override
protected void onResume() {
    super.onResume();
    if (mNsdHelper != null) {
        mNsdHelper.registerService(mConnection.getLocalPort());
        mNsdHelper.discoverServices();
    }
}

@Override
protected void onDestroy() {
    mNsdHelper.tearDown();
    mConnection.tearDown();
    super.onDestroy();
}

// NsdHelper's tearDown method
public void tearDown() {
    mNsdManager.unregisterService(mRegistrationListener);
    mNsdManager.stopServiceDiscovery(mDiscoveryListener);
}
```

# 使用WiFi建立P2P連接

編寫:[naizhengtan](#) - 原文:<http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html>

Android提供的Wi-Fi點對點（P2P）APIs允許應用程序無需連接到網絡和熱點的情況下連接到附近的設備。（Android Wi-Fi P2P框架遵循Wi-Fi Direct™驗證程序）Wi-Fi P2P技術使得應用程序可以快速發現附近的設備並與之交互。相比於藍牙技術，Wi-Fi P2P的優勢是具有較大的連接範圍。

本節主要內容是使用Wi-Fi P2P技術發現並連接到附近的設備。

## 配置應用權限

使用Wi-Fi P2P技術，需要添加`CHANGE_WIFI_STATE`,`ACCESS_WIFI_STATE`以及`INTERNET`三種權限到應用的manifest文件。Wi-Fi P2P技術雖然不需要訪問互聯網，但是它會使用Java中的標準socket。而使用socket需要具有`INTERNET`權限，這也是Wi-Fi P2P技術需要申請該權限的原因。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...

```

## 廣播接收器(BroadCast Receiver)和點對點管理器(Peer-to-peer Manager)

使用Wi-Fi P2P的時候需要偵聽相關的廣播事件（broadcast intent）。所以在應用中需要實例化一個`IntentFilter`，並將其設置為偵聽下列事件：

- `WIFI_P2P_STATE_CHANGED_ACTION`  
指示Wi-Fi P2P是否開啟
- `WIFI_P2P_PEERS_CHANGED_ACTION`  
代表對等節點（peer）列表發生了變化
- `WIFI_P2P_CONNECTION_CHANGED_ACTION`  
表明Wi-Fi P2P的連接狀態發生了改變
- `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`  
指示設備的詳細配置發生了變化

```
private final IntentFilter intentFilter = new IntentFilter();
...

```

```

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

    // Indicates a change in the Wi-Fi P2P status.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);

    // Indicates a change in the list of available peers.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);

    // Indicates the state of Wi-Fi P2P connectivity has changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);

    // Indicates this device's details have changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    ...
}

```

在`onCreate()`方法的最後，需要獲得`WifiPpManager`的實例，並調用它的`initialize()`方法。該方法將返回`WifiP2pManager.Channel`對象。你的應用將使用該對象與Wi-Fi P2P框架進行交互。

```

@Override
Channel mChannel;

public void onCreate(Bundle savedInstanceState) {
    ...
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
}

```

接下來，創建一個新的`BroadcastReceiver`類偵聽系統中Wi-Fi P2P的狀態變化。在`onReceive()`方法中，加入對上述四種不同P2P狀態變化的處理。

```

@Override
public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        // Determine if Wifi P2P mode is enabled or not, alert
        // the Activity.
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            activity.setIsWifiP2pEnabled(true);
        } else {
            activity.setIsWifiP2pEnabled(false);
        }
    } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

        // The peer list has changed! We should probably do something about
        // that.

    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {

        // Connection state changed! We should probably do something about
        // that.

    } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
        DeviceListFragment fragment = (DeviceListFragment) activity.getFragmentManager()
            .findFragmentById(R.id.frag_list);
        fragment.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(
            WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
    }
}

```

最後，在主界面開啓時，加入註冊intent filter和broadcast receiver的代碼，並在暫停或關閉時，註銷它們。最好的位置是在onResume()和onPause()方法中。

```
/** register the BroadcastReceiver with the intent values to be matched */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
    registerReceiver(receiver, intentFilter);
}

@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

## 初始化對等節點發現（Peer Discovery）過程

在Wi-Fi P2P中，應用通過調用discoverPeers()搜尋附近的設備。該方法需要以下參數：

- 上節中調用WifiP2pManager的initialize()函數獲得的WifiP2pManager.Channel對象
- 一個對WifiP2pManager.ActionListener接口的實現，包括了當系統成功/失敗發現所調用的方法

```
mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {

    @Override
    public void onSuccess() {
        // Code for when the discovery initiation is successful goes here.
        // No services have actually been discovered yet, so this method
        // can often be left blank. Code for peer discovery goes in the
        // onReceive method, detailed below.
    }

    @Override
    public void onFailure(int reasonCode) {
        // Code for when the discovery initiation fails goes here.
        // Alert the user that something went wrong.
    }
});
```

需要注意的是，在此的成功僅僅表示對Peer發現（Peer Discovery）的過程完成初始化。方法discoverPeers()開啓了發現過程並且立即返回。系統會通過調用WifiP2pManager.ActionListener中的方法通知應用對等節點發現過程初始化是否正確。同時，對等節點發現過程本身仍然繼續運行，直到一條連接或者一個P2P小組建立。

## 獲取對等節點列表

在完成對等節點發現過程的初始化後，我們需要進一步獲取附近的對等節點列表。第一步是實現WifiP2pManager.PeerListListener接口。該接口提供了Wi-Fi P2P框架發現的對等節點信息。下列代碼實現了相應功能：

```
private List peers = new ArrayList();
...

private PeerListListener peerListListener = new PeerListListener() {
    @Override
    public void onPeersAvailable(WifiP2pDeviceList peerList) {
```

```

        // Out with the old, in with the new.
        peers.clear();
        peers.addAll(peerList.getDeviceList());

        // If an AdapterView is backed by this data, notify it
        // of the change. For instance, if you have a ListView of available
        // peers, trigger an update.
        ((WiFiPeerListAdapter) getListAdapter()).notifyDataSetChanged();
        if (peers.size() == 0) {
            Log.d(WiFiDirectActivity.TAG, "No devices found");
            return;
        }
    }
}

```

接下來，完善上文廣播接收者（Broadcast Receiver）的onReceiver()方法。當收到[WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#)事件時，調用[requestPeer\(\)](#)方法獲取對等節點列表。在此，需要將WifiP2pManager.PeerListListener對象傳遞給該方法。一種方法是在廣播接收者構造時，就將對象作為參數傳入。

```

public void onReceive(Context context, Intent intent) {
    ...
    else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

        // Request available peers from the wifi p2p manager. This is an
        // asynchronous call and the calling activity is notified with a
        // callback on PeerListListener.onPeersAvailable()
        if (mManager != null) {
            mManager.requestPeers(mChannel, peerListListener);
        }
        Log.d(WiFiDirectActivity.TAG, "P2P peers changed");
    }...
}

```

現在，一個[WIFI\\_P2P\\_PEERS\\_CHANGED\\_ACTION](#)事件將觸發應用對Peer列表的更新了。

## 連接一個對等節點

為了連接到一個對等節點，你需要創一個新的[WifiP2pConfig](#)對象，並將要連接的設備信息從[WifiP2pDevice](#)拷貝到其中，最後調用[connect\(\)](#)方法。

```

@Override
public void connect() {
    // Picking the first device found on the network.
    WifiP2pDevice device = peers.get(0);

    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;
    config.wps.setup = WpsInfo.PBC;

    mManager.connect(mChannel, config, new ActionListener() {

        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore for now.
        }

        @Override
        public void onFailure(int reason) {
            Toast.makeText(WiFiDirectActivity.this, "Connect failed. Retry.",
                           Toast.LENGTH_SHORT).show();
        }
    });
}

```

```
}
```

在本段代碼中的`WifiP2pManager.ActionListener`僅能通知你初始化的成功或失敗。想要偵聽連接狀態的變化，需要實現`WifiP2pManager.ConnectionInfoListener`接口。接口中的`onConnectionInfoAvailable()`回調函數會在連接狀態發生改變時通知應用程序。當有多個設備同時試圖連接到一臺設備時（例如多人遊戲或者聊天羣），這一臺設備將被指定為“羣主”（group owner）。

```
@Override
public void onConnectionInfoAvailable(final WifiP2pInfo info) {

    // InetAddress from WifiP2pInfo struct.
    InetAddress groupOwnerAddress = info.groupOwnerAddress.getHostAddress();

    // After the group negotiation, we can determine the group owner.
    if (info.groupFormed && info.isGroupOwner) {
        // Do whatever tasks are specific to the group owner.
        // One common case is creating a server thread and accepting
        // incoming connections.
    } else if (info.groupFormed) {
        // The other device acts as the client. In this case,
        // you'll want to create a client thread that connects to the group
        // owner.
    }
}
```

此時，回頭繼續完善廣播接收者的`onReceive()`方法，並修改對`WIFI_P2P_CONNECTION_CHANGED_ACTION`intent的監聽部分的代碼。當該Intent接收到後，調用`requestConnectionInfo()`方法。此方法為異步，所以結果將會被你提供的`WifiP2pManager.ConnectionInfoListener`所獲取。

```
...
} else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {

    if (mManager == null) {
        return;
    }

    NetworkInfo networkInfo = (NetworkInfo) intent
        .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);

    if (networkInfo.isConnected()) {

        // We are connected with the other device, request connection
        // info to find group owner IP

        mManager.requestConnectionInfo(mChannel, connectionListener);
    }
    ...
}
```

# 使用WiFi P2P發現服務

編寫:[naizhengtan](http://naizhengtan) - 原文:<http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html>

在本章第一節“使得網絡服務可發現”中介紹瞭如何在局域網中發現並連接到其他設備的服務上。然而，即使在不接入網絡的環境中，Wi-Fi P2P的發現服務也可以使你的應用直接連接到附近的設備。與此同時，你也可以向外公佈自己設備上的服務。Wi-Fi P2P發現服務的這種能力可以在沒有局域網或者網絡熱點的情況下，幫助不同設備上的應用進行通信。

雖然本節所述的API與第一節NSD（Network Service Discovery）的API相似，但是具體的實現代碼卻截然不同。本節將講述如何通過Wi-Fi P2P技術發現附近可用設備中的服務。假設讀者已經對Wi-Fi P2P的API有一定瞭解。

## 配置Manifest

使用Wi-Fi P2P技術，需要添加`CHANGE_WIFI_STATE`,`ACCESS_WIFI_STATE`以及`INTERNET`三種權限到應用的manifest文件。Wi-Fi P2P技術雖然不需要訪問互聯網，但是它會使用Java中的標準socket。而使用socket需要具有`INTERNET`權限，這也是Wi-Fi P2P技術需要申請該權限的原因。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...
    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
}
```

## 添加本地服務

如果你想提供一個本地服務，就需要在服務發現框架中註冊該服務。當本地服務被成功註冊，系統將自動回覆所有來自附近的服務發現請求。

三步創建本地服務：

1. 新建`WifiP2pServiceInfo`對象
2. 加入相應服務的詳細信息
3. 調用`addLocalService()`註冊該服務

```
private void startRegistration() {
    // Create a string map containing information about your service.
    Map record = new HashMap();
    record.put("listenport", String.valueOf(SERVER_PORT));
    record.put("buddyname", "John Doe" + (int) (Math.random() * 1000));
    record.put("available", "visible");

    // Service information. Pass it an instance name, service type
```

```

// _protocol._transportlayer , and the map containing
// information other devices will want once they connect to this one.
WifiP2pDnsSdServiceInfo serviceInfo =
    WifiP2pDnsSdServiceInfo.newInstance("_test", "_presence._tcp", record);

// Add the local service, sending the service info, network channel,
// and listener that will be used to indicate success or failure of
// the request.
mManager.addLocalService(channel, serviceInfo, new ActionListener() {
    @Override
    public void onSuccess() {
        // Command successful! Code isn't necessarily needed here,
        // Unless you want to update the UI or add logging statements.
    }

    @Override
    public void onFailure(int arg0) {
        // Command failed. Check for P2P_UNSUPPORTED, ERROR, or BUSY
    }
});
}

```

## 發現附近的服務

Android使用回調函數通知應用程序附近可用的服務，因此發現服務的第一步是設置這些回調函數。新建一個[WifiP2pManager.DnsSdTxtRecordListener](#)實例偵聽實時收到的記錄（record）。這些記錄可以是來自其他設備的廣播。當收到記錄時，將其中的設備地址和其他相關信息拷貝出，供之後使用。下面的例子假設這條記錄不僅包含設備的身份，還包含一個名為“buddyname”的域（field）。

```

final HashMap<String, String> buddies = new HashMap<String, String>();
...
private void discoverService() {
    DnsSdTxtRecordListener txtListener = new DnsSdTxtRecordListener() {
        @Override
        /* Callback includes:
         * fullDomain: full domain name: e.g "printer._ipp._tcp.local."
         * record: TXT record data as a map of key/value pairs.
         * device: The device running the advertised service.
        */
        public void onDnsSdTxtRecordAvailable(
            String fullDomain, Map record, WifiP2pDevice device) {
            Log.d(TAG, "DnsSdTxtRecord available - " + record.toString());
            buddies.put(device.deviceAddress, record.get("buddyname"));
        }
    };
    ...
}

```

接下來創建[WifiP2pManager.DnsSdServiceResponseListener](#)對象，用以獲取目標服務的信息。這個對象將接受服務的實際描述以及連接信息。上一段代碼構建了一個包含設備地址和“buddyname”鍵值對的Map對象。[WifiP2pManager.DnsSdServiceResponseListener](#)對象使用這些配對信息將DNS記錄和對應的服務信息對應起來。當上述兩個監聽器構建完成了，調用[setDnsSdResponseListeners\(\)](#)將他們加入[WifiP2pManager](#)。

```

private void discoverService() {
    ...

    DnsSdServiceResponseListener servListener = new DnsSdServiceResponseListener() {
        @Override
        public void onDnsSdServiceAvailable(String instanceName, String registrationType,
            WifiP2pDevice resourceType) {

            // Update the device name with the human-friendly version from

```

```

        // the DnsTxtRecord, assuming one arrived.
        resourceType.deviceName = buddies
            .containsKey(resourceType.deviceAddress) ? buddies
            .get(resourceType.deviceAddress) : resourceType.deviceName;

        // Add to the custom adapter defined specifically for showing
        // wifi devices.
        WiFiDirectServicesList fragment = (WiFiDirectServicesList) getFragmentManager()
            .findFragmentById(R.id.frag_peerlist);
        WiFiDevicesAdapter adapter = ((WiFiDevicesAdapter) fragment
            .getListAdapter());

        adapter.add(resourceType);
        adapter.notifyDataSetChanged();
        Log.d(TAG, "onBonjourServiceAvailable " + instanceName);
    }
};

mManager.setDnsSdResponseListeners(channel, servListener, txtListener);
...
}

```

然後創建服務請求，並調用[addServiceRequest\(\)](#)方法。這個方法也需要一個監聽器（Listener）報告請求成功與失敗。

```

serviceRequest = WifiP2pDnsSdServiceRequest.newInstance();
mManager.addServiceRequest(channel,
    serviceRequest,
    new ActionListener() {
        @Override
        public void onSuccess() {
            // Success!
        }

        @Override
        public void onFailure(int code) {
            // Command failed. Check for P2P_UNSUPPORTED, ERROR, or BUSY
        }
    });

```

最後調用[discoverServices\(\)](#)。

```

mManager.discoverServices(channel, new ActionListener() {

    @Override
    public void onSuccess() {
        // Success!
    }

    @Override
    public void onFailure(int code) {
        // Command failed. Check for P2P_UNSUPPORTED, ERROR, or BUSY
        if (code == WifiP2pManager.P2P_UNSUPPORTED) {
            Log.d(TAG, "P2P isn't supported on this device.");
        } else if(...)

        ...
    }
});

```

如果所有部分都配置正確，你應該就能看到正確的結果了！如果遇到了問題，你可以查看[WifiP2pManager.ActionListener](#)中的回調函數。它們能夠指示操作是否成功。你可以將debug的代碼放置在[onFailure\(\)](#)中來診斷問題。其中的一些錯誤碼（Error Code）也許能為你帶來不小啓發。下面是一些常見的錯誤：

- **P2P\_UNSUPPORTED**

Wi-Fi P2P 不被現在的設備支持

- **BUSY**

系統忙於處理請求

- **ERROR**

內部錯誤

# 網絡連接操作

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/network-ops/index.html>

這一章會介紹一些基本的網絡操作，監視網絡連接（包括網絡改變），讓用戶來控制app對網絡的選擇使用。還會介紹如何解析與使用XML數據。

這節課包括一個示例應用，展示如何執行常見的網絡操作。你可以下載下面的範例，並把它作為可重用代碼在你自己的應用中使用。

[NetworkUsage.zip](#)

通過學習這章節的課程，你將會學習到一些有關於如何創建一個使用最少的網絡流量下載並解析數據的高效app的基本知識。

你還可以參考下面文章進階學習：

- [Optimizing Battery Life](#)
- [Transferring Data Without Draining the Battery](#)
- [Web Apps Overview](#)
- [Transmitting Network Data Using Volley](#)

Node:查看[Transmitting Network Data Using Volley](#)課程獲取Volley的相關信息，它是一個能幫助Android apps更方便快捷的執行網絡操作的HTTP庫。Volley可以在開源[AOSP](#)庫中找到。Volley可能能幫助你簡化網絡操作，提高你的app網絡操作的性能。

## Lessons

---

- [連接到網絡 - Connecting to the Network](#)

學習如何連接到網絡，選擇一個HTTP client，以及在UI線程外執行網絡操作。

- [管理網絡的使用情況 - Managing Network Usage](#)

學習如何檢查設備的網絡連接情況，創建偏好界面來控制網絡使用，以及響應連接變化。

- [解析XML數據 - Parsing XML Data](#)

學習如何解析和使用XML數據。

# 連接到網絡

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/network-ops/connecting.html>

這一課會演示如何實現一個簡單的連接到網絡的程序。它提供了一些你在創建即使最簡單的網絡連接程序時也應該遵循的最佳示例。請注意，想要執行本課的網絡操作首先需要在程序的manifest文件中添加以下permissions:

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

## 1)選擇一個HTTP Client

大多數連接網絡的Android app會使用HTTP來發送與接受數據。Android提供了兩種HTTP clients:

[HttpURLConnection](#) 與Apache [HttpClient](#)。二者均支持HTTPS，流媒體上傳和下載，可配置的超時，IPv6與連接池（connection pooling）。推薦從Android 2.3 Gingerbread版本開始使用 HttpURLConnection。關於這部分的更多詳情，請參考 [Android's HTTP Clients](#)。

## 2)檢查網絡連接

在你的app嘗試連接網絡之前，應通過函數[getActiveNetworkInfo](#)和[isConnected](#)檢測當前網絡是否可用。請注意，設備可能不在網絡覆蓋範圍內，或者用戶可能關閉Wi-Fi與移動網絡連接。關於這部分的更多詳情，請參考 [管理網絡的使用情況](#)

```
public void myClickHandler(View view) {
    ...
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // fetch data
    } else {
        // display error
    }
    ...
}
```

## 3)在一個單獨的線程中執行網絡操作

網絡操作會遇到不可預期的延遲。为了避免造成不好的用戶體驗，總是在UI線程之外執行網絡操作。[AsyncTask](#) 類提供了一種簡單的方式來處理這個問題。關於更多的詳情，請參考:[Multithreading For Performance](#).

在下面的代碼示例中，myClickHandler() 方法會執行new [DownloadWebpageTask\(\)](#).execute(stringUrl).DownloadWebpageTask是AsyncTask的子類，它實現了下面兩個方法:

- [doInBackground\(\)](#) 執行 [downloadUrl\(\)](#)方法。它以Web頁面的URL作為參數，方法[downloadUrl\(\)](#) 獲取並處理網頁返回的數據，執行完畢後，返回一個結果字符串。
- [onPostExecute\(\)](#) 接受結果字符串並把它顯示到UI上。

```

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);
    }

    // When user clicks button, calls AsyncTask.
    // Before attempting to fetch the URL, makes sure that there is a network connection.
    public void myClickHandler(View view) {
        // Gets the URL from the UI's text field.
        String urlString = urlText.getText().toString();
        ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
        if (networkInfo != null && networkInfo.isConnected()) {
            new DownloadWebpageText().execute(urlString);
        } else {
            textView.setText("No network connection available.");
        }
    }

    // Uses AsyncTask to create a task away from the main UI thread. This task takes a
    // URL string and uses it to create an HttpURLConnection. Once the connection
    // has been established, the AsyncTask downloads the contents of the webpage as
    // an InputStream. Finally, the InputStream is converted into a string, which is
    // displayed in the UI by the AsyncTask's onPostExecute method.
    private class DownloadWebpageText extends AsyncTask {
        @Override
        protected String doInBackground(String... urls) {

            // params comes from the execute() call: params[0] is the url.
            try {
                return downloadUrl(urls[0]);
            } catch (IOException e) {
                return "Unable to retrieve web page. URL may be invalid.";
            }
        }
        // onPostExecute displays the results of the AsyncTask.
        @Override
        protected void onPostExecute(String result) {
            textView.setText(result);
        }
    }
    ...
}

```

上面這段代碼的事件順序如下：

- 1.當用戶點擊按鈕時調用myClickHandler()，app將指定的URL傳給`AsyncTask`的子類`DownloadWebpageTask`。
- 2.`AsyncTask`的`doInBackground()`方法調用`downloadUrl()`方法。3.`downloadUrl()`以一個URL字符串作為參數，並用它創建一個`URL`對象。4.這個`URL`對象被用來創建一個`HttpURLConnection`。5.一旦建立連接，`HttpURLConnection`對象將獲取Web頁面的內容並得到一個`InputStream`。6.`InputStream`被傳給`readIt()`方法，該方法將流轉換成字符串。7.最後，`AsyncTask`的`onPostExecute()`方法將字符串展示在main activity的UI上。

## 4)連接並下載數據

在執行網絡交互的線程裏面，你可以使用`HttpURLConnection`來執行一個GET類型的操作並下載數據。在你調用`connect()`之後，你可以通過調用`getInputStream()`來得到一個包含數據的`InputStream`對象。

在下面的代碼示例中，`dolnBackground()` 方法會調用`downloadUrl()`。這個`downloadUrl()`方法使用給予的URL，通過`HttpURLConnection` 連接到網絡。一旦建立連接後，app就會使用`getInputStream()` 來獲取包含數據的`InputStream`。

```
// Given a URL, establishes an HttpURLConnection and retrieves
// the web page content as a InputStream, which it returns as
// a string.
private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
    // Only display the first 500 characters of the retrieved
    // web page content.
    int len = 500;

    try {
        URL url = new URL(myurl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        // Starts the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        is = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString = readIt(is, len);
        return contentAsString;

        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (is != null) {
            is.close();
        }
    }
}
```

請注意，`getResponseCode()` 會返回連接狀態碼( status code). 這是一種獲知額外網絡連接信息的有效方式。status code 是 200 則意味着連接成功。

## 5) 將輸入流(InputStream)轉換為字符串(String)

`InputStream` 是一種可讀的byte數據源。如果你獲得了一個 `InputStream`, 通常會進行解碼(decode)或者轉換為目標數據類型。例如，如果你是在下載圖片數據，你可能需要像下面這樣解碼並展示它：

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

在上面演示的示例中，`InputStream` 包含的是web頁面的文本內容。下面會演示如何把 `InputStream` 轉換為字符串，以便顯示在UI上。

```
// Reads an InputStream and converts it to a String.
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);
}
```

# 管理網絡

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/network-ops/managing.html>

這一課會介紹如何細化管理使用的網絡資源。如果你的程序需要執行很多網絡操作，你應該提供用戶設置選項來允許用戶控製程序的數據偏好。例如，同步數據的頻率，是否只在連接到WiFi才進行下載與上傳操作，是否在漫遊時使用套餐數據流量等等。這樣用戶纔不大可能在快到達流量上限時關閉你的程序獲取數據功能，因為他們可以精確控制您的app使用多少數據流量。

關於如何編寫一個最小化下載與網絡操作對電量影響的程序，請參考：

- [Optimizing Battery Life:](#)
- [Transferring Data Without Draining the Battery:](#)

示例：[NetworkUsage.zip](#)

## 1)檢查設備的網絡連接

設備可以有許多種網絡連接。這節課主要關注使用Wi-Fi或移動網絡連接的情況。關於所有可能的網絡連接類型，請看[ConnectivityManager](#).

通常Wi-Fi是比較快的。移動數據通常都是需要按流量計費，會比較貴。通常我們會選擇讓app在連接到WiFi時去獲取大量的數據。

在執行網絡操作之前檢查設備當前連接的網絡連接信息是個好習慣。這樣可以防止你的程序在無意間連接使用了非意向的網絡頻道。如果網絡連接不可用，你的應用應該優雅的做出響應。為了檢測網絡連接，我們需要使用到下面兩個類：

- [ConnectivityManager](#): 它會回答關於網絡連接狀態的查詢，並在網絡連接改變時通知應用程序。
- [NetworkInfo](#): 描述一個給定網絡類型(就本節而言是移動網絡或Wi-Fi)的網絡接口的狀態。

這段代碼檢查了Wi-Fi與移動網絡(Mobile)的網絡連接。它檢查了這些網絡接口是否可用(available,也就是說網絡連接存在)及是否已連接(connected,也就是說網絡連接存在,並且可以建立套接字(socket)來傳輸數據)：

```
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);
```

請注意你不應該僅僅靠網絡是否可用(available)做出決策。由於[isConnected\(\)](#)能夠處理不可靠的移動網絡(flaky mobile networks)，飛行模式(airplane mode)，受限制的後臺數據(restricted background data)等情況，你應該總是在執行網絡操作前檢查 [isConnected\(\)](#)。

一個更簡潔的檢查網絡是否可用的示例如下。[getActiveNetworkInfo\(\)](#)方法返回一個[NetworkInfo](#)實例，它表示可以找到的第一個已連接的網絡接口，如果返回null，則表示沒有已連接的網絡接口(意味着網絡連接不可用)：

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

你可以使用[NetworkInfo.DetailedState](#)，來獲取更加詳細的網絡信息，但很少有這樣的必要。

## 2)管理網絡的使用

你可以實現一個偏好設置的[activity](#)，使用戶能直接設置你的程序對網絡資源的使用。例如：

- 你可以允許用戶在僅僅連接到Wi-Fi時上傳視頻。
- 你可以根據諸如網絡可用，時間間隔等條件來選擇是否做同步的操作。

寫一個支持連接網絡和管理網絡使用的app，你的manifest需要有正確的權限(permission)和意圖過濾器(intent filter)：

- [android.permission.INTERNET](#)—允許應用程序打開網絡套接字。
- [android.permission.ACCESS\\_NETWORK\\_STATE](#)—允許應用程序訪問網絡連接信息。

你可以為你的[activity](#)聲明[ACTION\\_MANAGE\\_NETWORK\\_USAGE](#)動作(action)(Android 4.0中引入)的intent filter，這樣你的[activity](#)就能提供控制數據使用的選項了。[ACTION\\_MANAGE\\_NETWORK\\_USAGE](#)顯示管理指定應用程序網絡數據使用所需的設置。當你的app有一個允許用戶控制網絡使用的settings [activity](#)時，你應該為你的[activity](#)聲明這個intent filter。在章節概覽提供的Sample中，這個action被SettingsActivity類處理，它提供了偏好設置UI來讓用戶決定何時進行下載。

```
<?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" />

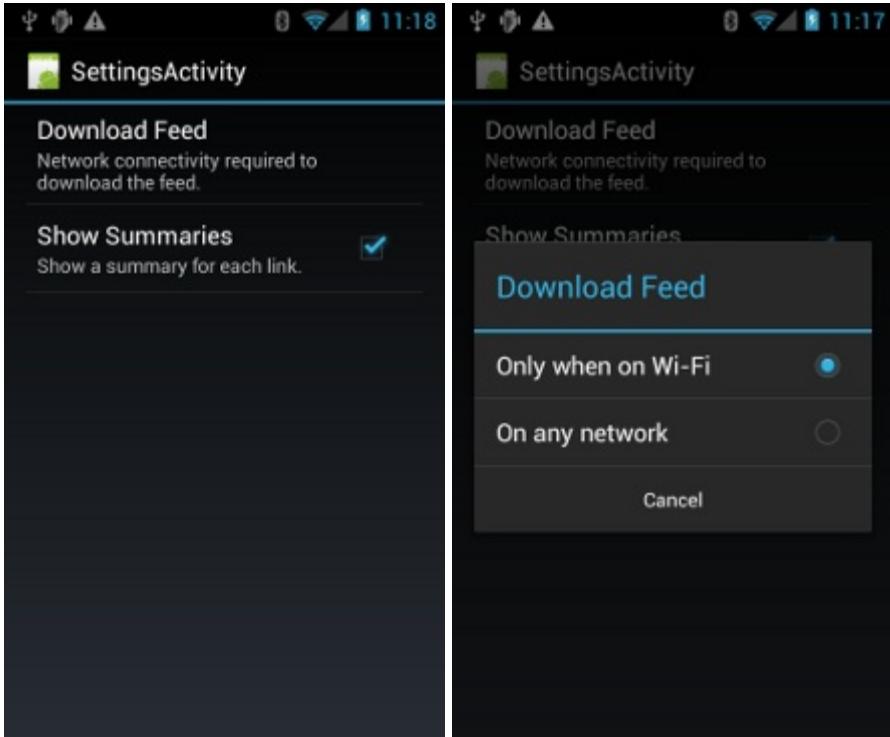
    <application
        ...>
        ...
        <activity android:label="SettingsActivity" android:name=".SettingsActivity">
            <intent-filter>
                <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
                <category android:name="android.intent.category.DEFAULT"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

## 3)實現一個Preferences Activity

正如上面manifest片段中看到的那樣，SettingsActivity有一個[ACTION\\_MANAGE\\_NETWORK\\_USAGE](#)的intent filter。SettingsActivity是PreferenceActivity的子類，它展示一個偏好設置頁面(如下兩張圖)讓用戶指定以下內容：

- 是否顯示每個XML提要條目的總結，或者只是每個條目一個鏈接。

- 是否在網絡連接可用時下載XML摘要，或者僅僅在Wi-Fi下下載。



下面是 `SettingsActivity`. 請注意它實現了 `OnSharedPreferenceChangeListener`. 當用戶改變了他的偏好，就會觸發 `onSharedPreferenceChanged()`, 這個方法會設置 `refreshDisplay` 為 `true`(這裏的變量存在於自己定義的 `activity`，見下一部分的代碼示例). 這會使的當用戶返回到 main `activity`的時候進行 `refresh`。(請注意，代碼中的註釋，不得不說，Googler寫的Code看起來就是舒服)

```
public class SettingsActivity extends PreferenceActivity implements OnSharedPreferenceChangeListener {

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

        // Loads the XML preferences file
        addPreferencesFromResource(R.xml.preferences);
    }

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

        // Registers a listener whenever a key changes
        getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }

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

        // Unregisters the listener set in onResume().
        // It's best practice to unregister listeners when your app isn't using them to cut down on
        // unnecessary system overhead. You do this in onPause().
        getPreferenceScreen().getSharedPreferences().unregisterOnSharedPreferenceChangeListener(this);
    }

    // When the user changes the preferences selection,
    // onSharedPreferenceChanged() restarts the main activity as a new
    // task. Sets the the refreshDisplay flag to "true" to indicate that
    // the main activity should update its display.
    // The main activity queries the PreferenceManager to get the latest settings.
}
```

```

@Override
public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
    // Sets refreshDisplay to true so that when the user returns to the main
    // activity, the display refreshes to reflect the new settings.
    NetworkActivity.refreshDisplay = true;
}

```

## 4)響應Preference Changes

當用戶在設置界面改變了偏好，它通常都會對app的行爲產生影響。在下面的代碼示例中，app會在onStart()方法中檢查偏好設置。如果設置的類型與當前設備的網絡連接類型相一致，那麼程序就會下載數據並刷新顯示。(例如，如果設置是"Wi-Fi" 並且設備連接了Wi-Fi)。(這是一個很好的代碼示例，如何選擇合適的網絡類型進行下載操作)

```

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?tagname=android&sort=newest";

    // Whether there is a Wi-Fi connection.
    private static boolean wifiConnected = false;
    // Whether there is a mobile connection.
    private static boolean mobileConnected = false;
    // Whether the display should be refreshed.
    public static boolean refreshDisplay = true;

    // The user's current network preference setting.
    public static String sPref = null;

    // The BroadcastReceiver that tracks network connectivity changes.
    private NetworkReceiver receiver = new NetworkReceiver();

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

        // Registers BroadcastReceiver to track network connection changes.
        IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
        receiver = new NetworkReceiver();
        this.registerReceiver(receiver, filter);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Unregisters BroadcastReceiver when app is destroyed.
        if (receiver != null) {
            this.unregisterReceiver(receiver);
        }
    }

    // Refreshes the display if the network connection and the
    // pref settings allow it.

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

        // Gets the user's network preference settings
        SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);

        // Retrieves a string value for the preferences. The second parameter
        // is the default value to use if a preference value is not found.
        sPref = sharedPrefs.getString("listPref", "Wi-Fi");

        updateConnectedFlags();

        if(refreshDisplay){

```

```

        loadPage();
    }

}

// Checks the network connection and sets the wifiConnected and mobileConnected
// variables accordingly.
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;
    }
}

// Uses AsyncTask subclass to download the XML feed from stackoverflow.com.
public void loadPage() {
    if (((sPref.equals(ANY)) && (wifiConnected || mobileConnected))
        || ((sPref.equals(WIFI)) && (wifiConnected))) {
        // AsyncTask subclass
        new DownloadXmlTask().execute(URL);
    } else {
        showErrorPage();
    }
}
...
}

```

## 5)檢測網絡連接變化

最後一部分是關於 [BroadcastReceiver](#) 的子類： [NetworkReceiver](#). 當設備網絡連接改變時， [NetworkReceiver](#) 會監聽到 [CONNECTIVITY\\_ACTION](#), 這時需要判斷當前網絡連接類型並相應的設置好 [wifiConnected](#) 與 [mobileConnected](#). 這樣做的結果是下次用戶回到app時， app只會下載最新的feed，如果 [NetworkActivity.refreshDisplay](#) 被設置為 true， app會更新顯示.

我們需要控制好 [BroadcastReceiver](#) 的使用，不必要的聲明註冊會浪費系統資源。通常是在 [onCreate\(\)](#) 去 [registers](#) 這個 [BroadcastReceiver](#)，在 [onDestroy\(\)](#) 時 [unregisters](#) 它。這樣做會比直接在 [manifest](#) 裏面直接註冊更輕量。當你在 [manifest](#) 裏面註冊了一個，你的程序可以在任何時候被喚醒，即使你已經好幾個星期沒有使用這個程序了。而通過前面的辦法進行註冊，可以確保用戶離開你的程序之後，不會因為那個 [Broadcast](#) 而被喚起。如果你確實要在 [manifest](#) 中註冊，且確保知道何時需要使用到它，你可以在合適的地方使用 [setComponentEnabledSetting\(\)](#) 來開啓或者關閉它。

下面是 [NetworkReceiver](#) 的代碼：

```

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();

    // Checks the user prefs and the network connection. Based on the result, decides whether
    // to refresh the display or keep the current display.
    // If the userpref is Wi-Fi only, checks to see if the device has a Wi-Fi connection.
    if (WIFI.equals(sPref) && networkInfo != null && networkInfo.getType() == ConnectivityManager.TYPE_WIFI) {
        // If device has its Wi-Fi connection, sets refreshDisplay
        // to true. This causes the display to be refreshed when the user
        // returns to the app.
    }
}

```

```
refreshDisplay = true;
Toast.makeText(context, R.string.wifi_connected, Toast.LENGTH_SHORT).show();

// If the setting is ANY network and there is a network connection
// (which by process of elimination would be mobile), sets refreshDisplay to true.
} else if (ANY.equals(sPref) && networkInfo != null) {
    refreshDisplay = true;

// Otherwise, the app can't download content--either because there is no network
// connection (mobile or Wi-Fi), or because the pref setting is WIFI, and there
// is no Wi-Fi connection.
// Sets refreshDisplay to false.
} else {
    refreshDisplay = false;
    Toast.makeText(context, R.string.lost_connection, Toast.LENGTH_SHORT).show();
}
}
```

# 解析XML數據

編寫:kesenhoo - 原文:<http://developer.android.com/training/basics/network-ops/xml.html>

Extensible Markup Language (XML) 是一組將文檔編碼成機器可讀形式的規則，也是一種在網絡上共享數據的普遍格式。經常更新內容的網站比如新聞網站和博客上都提供XML feed來記錄更新的信息，以便用戶進行訂閱讀取。

上傳[?]與解析XML數據是app的一個常見的功能。這一課會介紹如何解析XML文檔並使用他們的數據。

([?]這裏很奇怪，為什麼是Upload，看文章最後一段代碼示例的註釋，應該是Download纔對)

示例：[NetworkUsage.zip](#)

## 1)選擇一個Parser

我們推薦[XmlPullParser](#), 它是在Android上一個高效且可維護的解析XML方法。Android 上有這個接口的兩種實現方式：

- [KXmlParser](#)，通過 [XmlPullParserFactory.newPullParser\(\)](#)得到.
- [ExpatPullParser](#)，通過[Xml.newPullParser\(\)](#)得到.

兩個選擇都是比較好的。下面的示例中是使用ExpatPullParser，通過[Xml.newPullParser\(\)](#)得到.

## 2)分析Feed

解析一個feed的第一步是決定你需要獲取的字段。這樣解析器便去抽取出那些需要的字段而忽視其他的字段。

下面的XML片段是章節概覽Sample app中解析的Feed的片段。[StackOverflow](#)上每一個帖子都在feed中以時包含幾個嵌套的子標籤(tag)的entry標籤的形式出現。

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="http://backend.userland.com/creativecommonsModule" ...>
<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?tagname=android&sort=newest/tags" term="android"/>
    <category scheme="http://stackoverflow.com/feeds/tag?tagname=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>
```



Sample app從entry標籤與它的子標籤title,link和summary中提取數據。

### 3) 實例化Parser

下一步就是實例化一個parser並開始解析的操作。在下面的片段中，一個parser被初始化來處理名稱空間，並且將`InputStream`作為輸入。它通過調用`nextTag()`開始解析，並調用`readFeed()`方法，`readFeed()`方法會提取並處理app需要的數據：

```
public class StackOverflowXmlParser {  
    // We don't use namespaces  
    private static final String ns = null;  
  
    public List<Entry> 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();  
        }  
    }  
    ...  
}
```

### 4) 讀取Feed

`readFeed()`方法實際上並沒有處理feed的內容。它只是在尋找一個 "entry" 的標籤作為遞歸（recursively）處理整個feed的起點。`readFeed()`方法會跳過不是"entry"的標籤。當整個feed都被遞歸處理後，`readFeed()`會返回一個從feed中提取的包含了entry標籤（包括裏面的數據成員）的 `List`。然後這個`List`成為parser的返回值。

```
private List<Entry> readFeed(XmlPullParser parser) throws XmlPullParserException, IOException {  
    List<Entry> 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();  
        // Starts by looking for the entry tag  
        if (name.equals("entry")) {  
            entries.add(readEntry(parser));  
        } else {  
            skip(parser);  
        }  
    }  
    return entries;  
}
```

### 5) 解析XML

解析步驟如下：

- 正如在上面“分析Feed”所說的，判斷出你想要的tag。這個example抽取了entry標籤與它的內部標籤title,link,summary中的數據。
- 創建下面的方法：
  - 為每一個你想要獲取的標籤創建一個“read”方法。例如readEntry(),readTitle()等等。解析器從input stream中讀取tag。當讀取到entry,title,link或者summary標籤時，它會為那些標籤調用相應的方法，否則，跳過這個標籤。
  - 為每一個不同的標籤創建提取數據的方法，和使parser繼續解析下一個tag的方法。例如：
    - 對於title和summary標籤，解析器調用readText()。這個方法通過調用parser.getText()來獲取數據。
    - 對於link標籤，解析器先判斷這個link是否是我們想要的類型，然後再讀取數據。使用parser.getAttributeValue()來獲取link的數據。
    - 對於entry標籤，解析器調用readEntry()。這個方法解析entry的內部標籤並返回一個帶有title,link和summary數據成員的Entry對象。
  - 一個遞歸的輔助方法：skip()。關於這部分的討論，請看下面一部分內容：跳過你不需要的tag

下面的代碼演示瞭如何解析entries,titles,links與summaries。

```
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;  
    }  
}  
  
// Parses the contents of an entry. If it encounters a title, summary, or link tag, hands them off  
// to their respective "read" methods for processing. Otherwise, skips the tag.  
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);  
}  
  
// Processes title tags in the 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;  
}  
  
// Processes link tags in the 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;
}

// Processes summary tags in the 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;
}

// For the tags title and summary, extracts their text values.
private String readText(XmlPullParser parser) throws IOException, XmlPullParserException {
    String result = "";
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.getText();
        parser.nextTag();
    }
    return result;
}
...
}

```

## 6)跳過你不需要的tag

上面描述的XML解析步驟中有一步就是跳過不需要的tag，下面演示解析器的 skip() 方法：

```

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;
        }
    }
}

```

下面解釋這個方法如何工作：

- 如果下一個標籤不是一個 START\_TAG (開始標籤)，拋出異常。
- 它消耗掉 START\_TAG 以及接下來的所有內容，包括與開始標籤配對的 END\_TAG (結束標籤)。
- 為了保證方法在遇到正確的 END\_TAG 時停止，方法隨時記錄嵌套深度。

因此如果目前的標籤有子標籤，直到解析器已經處理了所有位於START\_TAG與對應的END\_TAG之間的事件之前，depth 的值不會為 0。例如，看解析器如何跳過 <author> 標籤，它有2個子標籤，<name> 與 <uri>：

- 第一次循環，在“”之後parser遇到的第一個標籤是 START\_TAG `<name>`。depth值變為2。
- 第二次循環，parser遇到的下一個標籤是 END\_TAG `</name>`。depth值變為1。
- 第三次循環，parser遇到的下一個標籤是 START\_TAG `<uri>`。depth值變為2。
- 第四次循環，parser遇到的下一個標籤是 END\_TAG `</uri>`。depth值變為1。
- 第五次同時也是最後一次循環，parser遇到的下一個標籤是 END\_TAG `</author>`。depth值變為0，表明成功跳過了`<author>`標籤。

## 6) 使用XML數據

示例程序是在 `AsyncTask` 中獲取與解析XML數據的。這樣處理工程不會在UI線程中執行。當處理完畢後，app會更新 main `activity`(`NetworkActivity`)的UI。

在下面示例代碼中，`loadPage()` 方法做了下面的事情：

- 初始化一個帶有URL地址的String變量，用來訂閱XML feed。
- 如果用戶設置與網絡連接都允許，會觸發 `new DownloadXmlTask().execute(url)`。這會初始化一個新的 `DownloadXmlTask` (`AsyncTask` 的子類) 對象並且開始執行它的 `execute()` 方法，這個方法會下載並解析feed，並返回結果字符串展示在UI上。

```
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";

    // Whether there is a Wi-Fi connection.
    private static boolean wifiConnected = false;
    // Whether there is a mobile connection.
    private static boolean mobileConnected = false;
    // Whether the display should be refreshed.
    public static boolean refreshDisplay = true;
    public static String sPref = null;

    ...

    // Uses AsyncTask to download the XML feed from 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 {
            // show error
        }
    }
}
```

下面展示的是 `AsyncTask` 的子類，`DownloadXmlTask`實現了 `AsyncTask` 的如下方法：

- `doInBackground()` 執行 `loadXmlFromNetwork()` 方法。它以feed的URL作為參數。`loadXmlFromNetwork()`獲取並處理feed。當它完成時，返回一個結果字符串。
- `onPostExecute()` 接受返回的字符串並將其展示在UI上。

```
// Implementation of AsyncTask used to download XML feed from 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);
    // Displays the HTML string in the UI via a WebView
    WebView myWebView = (WebView) findViewById(R.id.webview);
    myWebView.loadData(result, "text/html", null);
}
}

```

下面是loadXmlFromNetwork()方法做的事情：

1. 實例化一個StackOverflowXmlParser. 它同樣創建一個 `List<Entry>` , Entry對象包括title, url, summary字段來保存從XML feed中提取的數據.
2. 調用 `downloadUrl()` , 它會獲取feed, 並將其作爲 `InputStream` 返回.
3. 使用 StackOverflowXmlParser 解析 `InputStream` . StackOverflowXmlParser 用從feed中獲取的數據填充 `List<Entry>` .
4. 處理 `List<Entry>` , 將feed數據與HTML標記結合起來.
5. 返回一個HTML字符串, `AsyncTask` 的方法 `onPostExecute()` 會將其展示在main activity的UI上.

```

// Uploads XML from stackoverflow.com, parses it, and combines it with
// HTML markup. Returns HTML string.【這裏可以看出應該是Download】
private String loadXmlFromNetwork(String urlString) throws XmlPullParserException, IOException {
    InputStream stream = null;
    // Instantiate the parser
    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:mm:ss");

    // Checks whether the user set the preference to include summary text
    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);
        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (stream != null) {
            stream.close();
        }
    }

    // StackOverflowXmlParser returns a List (called "entries") of Entry objects.
    // Each Entry object represents a single post in the XML feed.
    // This section processes the entries list to combine each entry with HTML markup.
    // Each entry is displayed in the UI as a link that optionally includes
    // a text summary.
    for (Entry entry : entries) {
        htmlString.append("<p><a href='");
        htmlString.append(entry.link);
        htmlString.append("'>" + entry.title + "</a></p>");
    }
}

```

```
// If the user set the preference to include summary text,
// adds it to the display.
if (pref) {
    htmlString.append(entry.summary);
}
return htmlString.toString();
}

// Given a string representation of a URL, sets up a connection and gets
// an input stream.
【關於Timeout具體應該設置多少，可以借鑑這裏的數據，當然前提是一般情況下】
// Given a string representation of a URL, sets up a connection and gets
// an input stream.
private InputStream downloadUrl(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* milliseconds */);
    conn.setConnectTimeout(15000 /* milliseconds */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Starts the query
    conn.connect();
    return conn.getInputStream();
}
```

# 高效下載

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/efficient-downloads/index.html>

在這一章，我們將學習最小化下載，網絡連接，尤其是無線電連接對電量的影響。

下面幾節課會演示瞭如何使用緩存(caching)，輪詢(polling)，預取(prefetching)等技術來計劃與執行下載操作。我們還會學習無線電波的power-use屬性配置是如何影響我們對於在何時，用什麼，以何種方式來傳輸數據的選擇。當然這些選擇是為了最小化對電池壽命的影響。

你同樣需要閱讀 [Optimizing Battery Life](#)

## 課程列表

---

- [Optimizing Downloads for Efficient Network Access - 使用有效的網絡連接方式來最優化下載](#)

這節課介紹了無線廣播狀態機(wireless radio state machine)，解釋了app的連接模型(connectivity model)如何與它交互，以及如何最小化數據連接，使用預取(prefetching)和捆綁(bundling)最小化數據傳輸對電池消耗的影響。

- [Minimizing the Effect of Regular Updates - 優化常規更新操作的效果](#)

這節課將我們將瞭解如何調整你的刷新頻率以最大程度減輕對底層的無線廣播狀態機的後臺刷新的影響。

- [Redundant Downloads are Redundant - 重複的下載是冗餘的](#)

減少下載的最根本途徑是隻下載你需要的內容。這節課介紹了消除冗餘下載的一些最佳實踐。

- [Modifying your Download Patterns Based on the Connectivity Type - 根據網絡連接類型來更改下載模式](#)

不同連接類型都對電池電量的影響並不相同。不僅僅是Wi-Fi比無線電波更省電，不同的無線廣播技術對電池也有不同的影響。

# Optimizing Downloads for Efficient Network Access

編寫:kesenhoo - 原文:<http://developer.android.com/training/efficient-downloads/efficient-network-access.html>

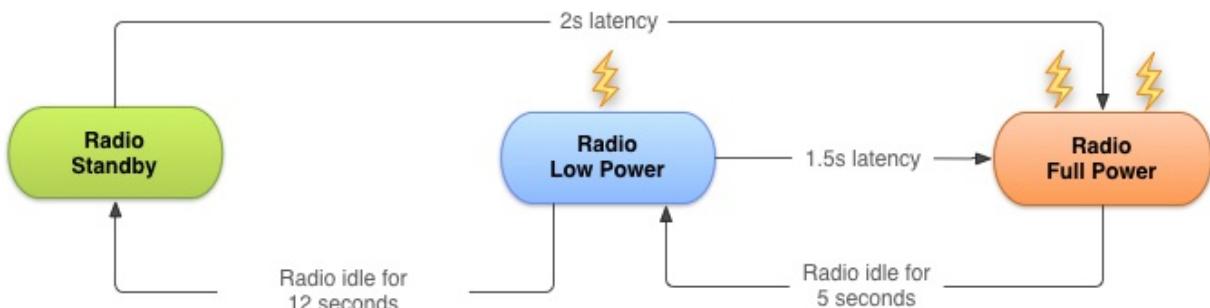
使用無線電波(wireless radio)進行傳輸數據這一行為也許是我們app最耗電的操作之一。為了最小化網絡連接對電量消耗，懂得連接模式(connectivity model)會如何影響底層的無線電硬件設備是至關重要的。這節課介紹了無線電波狀態機(wireless radio state machine)，並解釋了app的connectivity model是如何與狀態機進行交互的。然後會提出建議的方法來最小化我們的數據連接，使用預取(prefetching)與捆綁(bundle)的方式進行數據的傳輸，這些操作都是為了最小化電量的消耗。

## 1)The Radio State Machine

一個完全活動的無線電會大量消耗電量，因此需要學習如何在不同狀態下進行過渡，這樣能夠避免電量的浪費。典型的3G無線電網絡有三種能量狀態：

- Full power: 當無線連接被激活的時候，允許設備以最大的傳輸速率進行操作。
- Low power: 一種中間狀態，對電量的消耗差不多是Full power狀態下的50%。
- Standby: 最低的狀態，沒有數據連接需要傳輸。

在最低並且空閒的狀態下，電量消耗相對來說是少的。這裏需要介紹一延時(latency)的機制，從low status返回到full status大概需要花費1.5秒，從idle status返回到full status需要花費2秒。為了最小化延遲，狀態機使用了一種後滯過渡到更低能量狀態的機制。下圖是一個典型的3G無線電波狀態機的圖示(AT&T電信的一種制式)。



- 在每一臺設備上的無線狀態機會根據無線電波的制式(2G,3G,LTE等)而改變，並且由設備本身自己所使用的網絡進行定義與配置。
- 這一課描述了一種典型的3G無線電波狀態機，[data provided by AT&T](#)。這些原理是具有通用性的，在其他的無線電波上同樣適用。
- 這種方法在瀏覽通常的網頁操作上是特別有效的，因為它可以阻止用戶在瀏覽網頁時的一些不必要的浪費。而且相對較短的後期處理時間也保證了當一個session結束的時候，無線電波可以轉移到相對較低的能量狀態。
- 不幸的是，這個方法會導致在現代的智能機系統例如Android上的apps效率低下。因為Android上的apps不僅僅可以在前臺運行，也可以在後臺運行。(無線電波的狀態改變會影響到本來的設計，有些想在前臺運行的可能會因為切換到低能量狀態而影響程序效率。坊間說手機在電量低的狀態下無線電波的強度會增大好幾倍來保證信號，可能與這個有關。)

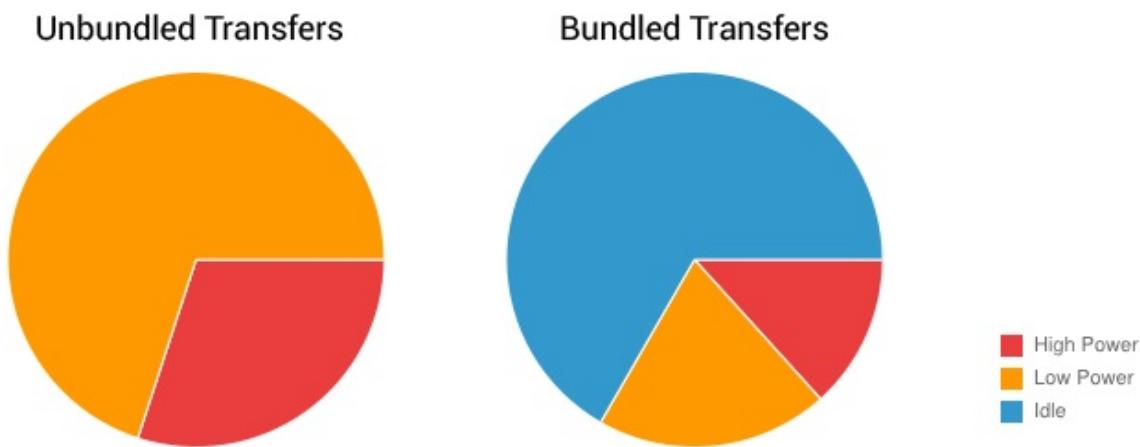
## 2)How Apps Impact the Radio State Machine

每一次新創建一個網絡連接，無線電波就切換到full power狀態。在上面典型的3G無線電波狀態機情況下，無線電波會

在傳輸數據時保持在full power的狀態，結束之後會有一個附加的5秒時間切換到low power,再之後會經過12秒進入到low energy的狀態。因此對於典型的3G設備，每一次數據傳輸的會話都會引起無線電波都會持續消耗大概20秒的能量。

實際上，這意味着一個app傳遞1秒鐘的unbundled data會使得無線電波持續活動18秒(18 = 1秒的傳輸數據+5秒過渡時間回到low power+12秒過渡時間回到standby)。因此每分鐘，它會消耗18秒high power的電量，42秒的low power的電量。

通過比較，如果每分鐘app會傳輸bundle的数据持續3秒的話，其中會使得無線電波持續在high power狀態僅僅8秒鐘，在low power狀態僅僅12秒鐘。上面第二種傳輸bundle data的例子，可以看到減少了大量的電量消耗。圖示如下：



### 3)Prefetch Data

預取(Prefetching)數據是一種減少獨立數據傳輸會話數量的有效方法。預取技術指的是在一定時間內，單次連接操作，以最大能力下載的情況下，下載所有用戶可能需要的數據。

通過前面的傳輸數據的技術，減少了大量下載數據所需的無線電波激活時間。這樣不僅保存了電量，也降低了帶寬，減少了下載時間。

預取技術通過減少了用戶閱讀所需等待的時間，提高了用戶體驗。

然而，過於頻繁的預取技術的使用，不僅僅會導致電量消耗快速增長，還有可能預取到一些並不需要的數據。同樣，確保app不會因為等待預取全部完成而卡到程序的開始播放也是非常重要的。從實踐的角度，那意味着需要逐步處理數據，並且按照有優先級的順序開始進行數據傳遞，這樣能確保不卡到程序的開始播放的同時數據也能夠得到持續的下載。

那麼應該如何控制預取的操作呢？這需要根據正在下載的數據大小與可能被用到的數據量來決定。一個基於上面狀態機情況的比較大概的建議是：對於數據來說，大概有50%的機會可能用在當前用戶的會話中，那麼我們可以預取大約6秒(大約1-2Mb)，這大概使得潛在可能要用的數據量與可能已經下載好的數據量相一致。

通常來說，預取1-5Mb會比較好，這種情況下，我們僅僅只需要每隔2-5分鐘開始另一段下載。根據這個原理，大數據的下載，比如視頻文件，應該每隔2-5分鐘開始另一段下載，這樣能有效的預取到下面幾分鐘內的數據進行預覽。

值得注意的是，下載需要是bundled的形式，而且上面那些大概的數據與時間可能會根據網絡連接的類型與速度有所變化，這些都將在下面兩部分內容講到。

讓我們來看一些例子：

A music player 你可以選擇預取整個專輯，然而這樣用戶在第一首歌曲之後停止監聽，那麼就浪費了大量的帶寬於電量。一個比較好的方法是維護一首歌曲的緩衝區。對於流媒體音樂，不應該去維護一段連續的數據流，因為這樣會使得無線電波一直保持激活狀態，應該考慮把HTTP的數據流集中一次傳輸到音頻流，就像上面描述的預取技術一樣(下載好2Mb，然後開始一次取出，再去下載下面的2Mb)。

A news reader 許多news apps嘗試通過只下載新聞標題來減少帶寬，完整的文章僅在用戶想要讀取的時候再去讀取，而且文章也會因為太長而剛開始只顯示部分信息，等用戶下滑時再去讀取完整信息。使用這個方法，無線電波僅僅會在用戶點擊更多信息的時候纔會被激活。但是，在切換文章分類預閱讀文章的時候仍然會造成大量潛在的消耗。

一個比較好的方法是在啓動的時候預取一個合理數量的數據，比如在啓動的時候預取一些文章的標題與縮略圖信息。之後開始獲取剩餘的標題預縮略信息。

另一個方法是預取所有的標題，縮略信息，文章文字，甚至是所有文章的圖片-根據既設的後臺程序進行逐一獲取。這樣做的風險是花費了大量的帶寬與電量去下載一些不會閱讀到的內容，因此這需要比較小心思考是否合適。其中的一個解決方案是，當在連接至Wi-Fi時有計劃的下載所有的內容，並且如果有可能最好是設備正在充電的時候。關於這個的細節的實現，我們將在後面的課程中涉及到。

## 4)Batch Transfers and Connections

---

使用典型3G無線網絡制式的時候，每一次初始化一個連接(與需要傳輸的數據量無關)，都有可能導致無線電波持續花費大約20秒的電量。

一個app，若是每20秒進行一次ping server的操作，假設這個app是正在運行且對用戶可見，那麼這會導致無線電波不確定什麼時候被開啓，這樣即使在沒有實際數據需要傳輸的情況下，仍可能造成大量電量的花費。

因此，對數據進行bundle操作並且創建一個序列來存放這些bundle好的數據就顯的非常重要。操作正確的話，可以使得大量的數據集中進行發送，這樣使得無線電波的激活時間儘可能的少，同時減少大部分電量的花費。這樣做的潛在好處是儘可能在每次傳輸數據的會話中儘可能多的傳輸數據而且減少了會話的次數。

## 5)Reduce Connections

---

重用已經存在的網絡連接比起重新建立一個新的連接通常來說是更有效率的。重用網絡連接同樣可以使得在擁擠不堪的網絡環境中進行更加智能的互動。當可以捆綁所有請求在一個GET裏面的時候不要同時創建多個網絡連接或者把多個GET請求進行串聯。

例如，可以一起請求所有文章的情況下，不要根據多個欄目進行多次請求。為進行termination / termination acknowledgement packets的收發，無線電波會在timeout之前保持激活狀，所以如果不需要的連接請立即關閉而不是等待他們timeout。

之前說道，如果過早對一個連接執行關閉操作，會導致後面再次請求時重新建立一個在Server與Client之間的連接，而我們說過要儘量避免建立重複的連接，那麼有個有效的折中辦法是不要立即關閉，而是在timeout之前關閉(即稍微晚點卻又不至於到timeout)。

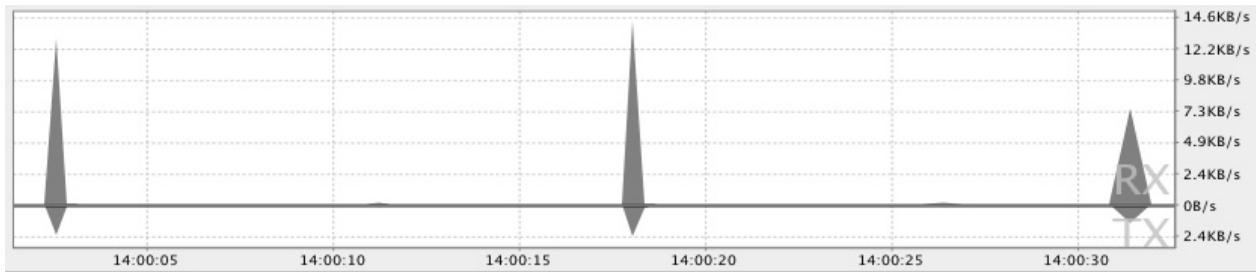
Note: 使用`HttpURLConnection`，而不是Apache的`HttpClient`，前者有做response cache.

## 6)Use the DDMS Network Traffic Tool to Identify Areas of Concern

---

The Android [DDMS \(Dalvik Debug Monitor Server\)](#) 包含了一個查看網絡使用詳情的欄目來允許跟蹤app的網絡請求。使用這個工具，可以監測app是在何時，如何傳輸數據的，從而進行代碼的優化。

下圖顯示了傳輸少量的網絡模型，可以看到每次差不多相隔15秒，這意味着可以通過預取技術或者批量上傳來大幅提高效率。



通過監測數據傳輸的頻率與每次傳輸的數據量，可以查看出哪些位置應該進行優化，通常的，圖中顯示的短小的類似釘子形狀的位置，可以進行與附近位置的請求進行做merge的動作。

為了更好的檢測出問題所在，Traffic Status API允許你使用TrafficStats.setThreadStatsTag()的方法標記數據傳輸發生在某個Thread裏面，然後可以手動的使用tagSocket()進行標記到或者使用untagSocket()來取消標記，例如：

```
TrafficStats.setThreadStatsTag(0xF00D);
TrafficStats.tagSocket(outputSocket);
// Transfer data using socket
TrafficStats.untagSocket(outputSocket);
```

Apache的HttpClient與URLConnection庫可以自動tag sockets使用當前getThreadStatusTag()的值。那些庫在通過keep-alive pools循環的時候也會tag與untag sockets。

```
TrafficStats.setThreadStatsTag(0xF00D);
try {
    // Make network request using HttpClient.execute()
} finally {
    TrafficStats.clearThreadStatsTag();
}
```

Socket tagging 是在Android 4.0上才被支持的，但是實際情況是僅僅會在運行Android 4.0.3 or higher的設備上纔會顯示。

# Minimizing the Effect of Regular Updates

編寫:kesenhoo - 原文:<http://developer.android.com/training/efficient-downloads/regular-update.html>

最佳的定時更新頻率是不確定的，通常由設備狀態，網絡連接狀態，用戶行為與用戶定義明確的偏好而決定。

**Optimizing Battery Life**這一章有討論如何根據設備狀態來修改更新頻率，從而達到編寫一個低電量消耗的程序。可執行的操作包括當斷開網絡連接的時候去關閉後臺服務，在電量比較低的時候減少更新的頻率等。

這一課會介紹更新頻率是多少纔會使得更新操作對無線電狀態機的影響最小。(C2DM與指數退避算法的使用)

## 1) Use Google Cloud Messaging as an Alternative to Polling

關於 Google Cloud Messaging for Android (GCM)詳情 ,請參考:<http://developer.android.com/google/gcm/index.html>

每次app去向server詢問檢查是否有更新操作的時候會激活無線電，這樣造成了不必要的能量消耗(在3G情況下，會差不多消耗20秒的能量)。

GCM是一個用來從server到特定app傳輸數據的輕量級的機制。使用GCM,server會在某個app有需要獲取新數據的時候通知app有這個消息。

比起輪詢方式(app為了即時拿到最新的數據需要定時向server請求數據)，GCM這種有事件驅動的模式會在僅僅有數據更新的時候通知app去創建網絡連接來獲取數據(很顯然這樣減少了app的大量操作，當然也減少了很多電量消耗)。

GCM需要通過使用固定TCP/IP連接來實現操作。當在你的設備上可以實現固定IP的時候，最好使用GCM。(這個地方應該不是傳統意義上的固定IP，可以理解為某個會話情況下)。很明顯，使用GCM既減少了網絡連接次數，也優化了帶寬，還減少了對電量的消耗。

Ps:大陸的Google框架通常被移除掉，這導致GCM實際上根本沒有辦法在大陸的App上使用

## 2) Optimize Polling with Inexact Repeating Alarms and Exponential Backoffs

如果需要使用輪詢機制，在不影響用戶體驗的前提下，當然設置默認更新頻率是越低越好(減少電量的浪費)。

一個簡單的方法是給用戶提供更新頻率的選擇，允許用戶自己來處理如何平衡數據及時性與電量的消耗。

當設置安排好更新操作後，可以使用不確定重複提醒的方式來允許系統把當前這個操作進行定向移動(比如推遲一會)。

```
int alarmType = AlarmManager.ELAPSED_REALTIME;
long interval = AlarmManager.INTERVAL_HOUR;
long start = System.currentTimeMillis() + interval;

alarmManager.setInexactRepeating(alarmType, start, interval, pi);
```

若是多個提醒都安排在某個點同時被觸發，那麼這樣就可以使得多個操作在同一個無線電狀態下操作完。

如果可以，請設置提醒的類型為 `ELAPSED_REALTIME` or `RTC` 而不是 `_WAKEUP`。這樣能夠更進一步的減少電量的消耗。

我們還可以通過根據app被使用的頻率來有選擇性的減少更新的頻率。

另一個方法是在app在上一次更新操作之後還未被使用的情況下，使用指數退避算法 exponential back-off algorithm 來減少更新頻率。當然我們也可以使用一些類似指數退避的方法。

```
SharedPreferences sp =
    context.getSharedPreferences(PREFS, Context.MODE_WORLD_READABLE);

boolean appUsed = sp.getBoolean(PREFS_APPUSED, false);
long updateInterval = sp.getLong(PREFS_INTERVAL, DEFAULT_REFRESH_INTERVAL);

if (!appUsed)
    if ((updateInterval *= 2) > MAX_REFRESH_INTERVAL)
        updateInterval = MAX_REFRESH_INTERVAL;

Editor spEdit = sp.edit();
spEdit.putBoolean(PREFS_APPUSED, false);
spEdit.putLong(PREFS_INTERVAL, updateInterval);
spEdit.apply();

rescheduleUpdates(updateInterval);
executeUpdateOrPrefetch();
```

初始化一個網絡連接的花費不會因為是否成功下載了數據而改變。我們可以使用指數退避算法來減少重複嘗試(retry)的次數，這樣能夠避免浪費電量。例如：

```
private void retryIn(long interval) {
    boolean success = attemptTransfer();

    if (!success) {
        retryIn(interval*2 < MAX_RETRY_INTERVAL ?
            interval*2 : MAX_RETRY_INTERVAL);
    }
}
```

筆者結語：這一課講到GCM與指數退避算法等，其實這些細節很值得我們注意，如果能在實際項目中加以應用，很明顯程序的質量上升了一個檔次！

# Redundant Downloads are Redundant

編寫:kesenhoo - 原文:<http://developer.android.com/training/efficient-downloads/redundant-redundant.html>

減少下載的最基本方法是僅僅下載那些你需要的。從數據的角度看，我們可以通過傳遞類似上次更新時間這樣的參數來制定查詢數據的條件。同樣，在下載圖片的時候，server那邊最好能夠減少圖片的大小，而不是讓我們下載完整大小的圖片。

## 1)Cache Files Locally

避免下載重複的數據是很重要的。可以使用緩存機制來處理這個問題。緩存static的資源，例如完整的圖片。這些緩存的資源需要分開存放。為了保證app不會因為緩存而導致顯示的是舊數據，請在緩存中獲取數據的同時檢測其是否過期，當數據過期的時候，會提示進行刷新。

```
long currentTime = System.currentTimeMillis();

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

long expires = conn.getHeaderFieldDate("Expires", currentTime);
long lastModified = conn.getHeaderFieldDate("Last-Modified", currentTime);

setDataExpirationDate(expires);

if (lastModified < lastUpdateTime) {
    // Skip update
} else {
    // Parse update
}
```

使用這種方法，可以有效保證緩存裏面一直是最新的數據。

可以使用下面的方法來獲取External緩存的目錄：(目錄會是sdcard下面的 Android/data/com.xxx.xxx/cache )

```
Context.getExternalCacheDir();
```

下面是獲取內部緩存的方法，請注意，存放在內存中的數據有可能因內部空間不夠而被清除。(類似: system/data/data/com.xxx.xxx./cache )

```
Context.getCache();
```

上面兩個Cache的文件都會在app卸載的時候被清除。

Ps:請注意這點:發現很多應用總是隨便在sdcard下面創建一個目錄用來存放緩存，可是這些緩存又不會隨着程序的卸載而被刪除，這其實是不符合規範，程序都被卸載了，為何還要留那麼多垃圾文件，而且這些文件有可能會泄漏一些隱私信息。除非你的程序是音樂下載，拍照程序等等，這些確定程序生成的文件是會被用戶需要留下的，不然都應該使用上面的那種方式來獲取Cache目錄

## 2)Use the HttpURLConnection Response Cache

在 Android 4.0 裏面為 HttpURLConnection 增加了一個 response cache(這是一個很好的減少 http 請求次數的機制，Android 官方推薦使用 HttpURLConnection 而不是 Apache 的 DefaultHttpClient，就是因為前者不僅僅有針對 android 做 http 請求的優化，還在 4.0 上增加了 Response Cache，這進一步提高了效率)

我們可以使用反射機制開啓 HTTP response cache，看下面的例子：

```
private void enableHttpServletResponseCache() {
    try {
        long httpCacheSize = 10 * 1024 * 1024; // 10 MiB
        File httpCacheDir = new File(getCacheDir(), "http");
        Class.forName("android.net.http.HttpResponseCache")
            .getMethod("install", File.class, long.class)
            .invoke(null, httpCacheDir, httpCacheSize);
    } catch (Exception httpResponseCacheNotAvailable) {
        Log.d(TAG, "HTTP response cache is unavailable.");
    }
}
```

上面的 sample code 會在 Android 4.0 以上的設備上開啓 response cache，同時不會影響到之前的程序。在 cache 被開啓之後，所有 cache 中的 HTTP 請求都可以直接在本地存儲中進行響應，並不需要開啓一個新的網絡連接。被 cache 起來的 response 可以被 server 所確保沒有過期，這樣就減少了下載所需的帶寬。沒有被 cached 的 response 會為了方便下次請求而被存儲在 response cache 中。

# Modifying your Download Patterns Based on the Connectivity Type

編寫:kesenhoo - 原文:<http://developer.android.com/training/efficient-downloads/connectivity-patterns.html>

並不是所有的網絡類型(Wi-Fi,3G,2G,etc)對電量的消耗是同等的。不僅僅Wi-Fi電波比無線電波的耗電量要少很多，而且不同的無線電波(3G,2G,LTE.....)也存在使用不同電量的區別。

## 1)Use Wi-Fi

在大多數情況下，Wi-Fi電波會在使用相對較低的電量的情況下提供一個相對較大的帶寬。因此，我們需要爭取儘量使用Wi-Fi來傳遞數據。我們可以使用Broadcast Receiver來監聽當網絡連接狀態。當切換為Wi-Fi時，我們可以進行大量的數據傳遞操作，例如下載，執行定時的更新操作，甚至是在這個時候暫時加大更新頻率。這些內容都可以在前面的課程中找到。

## 2)Use Greater Bandwidth to Download More Data Less Often

當通過無線電進行連接的時候，更大的帶寬通常伴隨着更多的電量消耗。這意味着LTE(一種4G網絡制式)會比3G制式消耗更多，當然比起2G更甚。

從Lesson 1我們知道了無線電狀態機是怎麼回事，通常來說相對更寬的帶寬網絡制式會有更長的狀態切換時間(也就是從full power過渡到standby有更長一段時間的延遲)。同時，更高的帶寬意味着可以更大量的進行prefetch，下載更多的數據。也許這個說法不是很直觀，因為過渡的時間比較長，而過渡時間的長短我們無法控制，也就是過渡時間的電量消耗差不多是固定了，既然這樣，我們在每次傳輸會話中為了減少更新的頻率而把無線電激活的時間拉長，這樣顯的更有效率。也就是儘量一次性把事情做完，而不是斷斷續續的請求。

例如：如果LTE無線電的帶寬與電量消耗都是3G無線電的2倍，我們應該在每次會話的時候都下載4倍於3G的數據量，或者是差不多10Mb(前面文章有說明3G一般每次下載2Mb)。當然，下載到這麼多數據的時候，我們需要好好考慮prefetch本地存儲的效率並且需要經常刷新預取的cache。我們可以使用connectivity manager來判斷當前激活的無線電波，並且根據不同結果來修改prefetch操作。

```
ConnectivityManager cm =
    (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);

TelephonyManager tm =
    (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);

NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

int PrefetchCacheSize = DEFAULT_PREFETCH_CACHE;

switch (activeNetwork.getType()) {
    case (ConnectivityManager.TYPE_WIFI):
        PrefetchCacheSize = MAX_PREFETCH_CACHE; break;
    case (ConnectivityManager.TYPE_MOBILE): {
        switch (tm.getNetworkType()) {
            case (TelephonyManager.NETWORK_TYPE_LTE |
                TelephonyManager.NETWORK_TYPE_HSPAP):
                PrefetchCacheSize *= 4;
                break;
            case (TelephonyManager.NETWORK_TYPE_EDGE |
```

```
    TelephonyManager.NETWORK_TYPE_GPRS):
    PrefetchCacheSize /= 2;
    break;
    default: break;
}
break;
}
default: break;
}
```

Ps：想要最大化效率與最小化電量的消耗，需要考慮的東西太多了，通常來說，會根據app的功能需求來選擇有所側重，那麼前提就是需要瞭解到底哪些對效率的影響比較大，這有利於我們做出最優選擇。

# 雲同步

---

編寫:kesenhoo , jdneo - 原文:<http://developer.android.com/training/cloudsync/index.html>

通過為網絡連接提供強大的API，Android Framework可以幫助你建立豐富的，具有云功能的App。這些App可以同步數據到遠程服務器端，確保所有你的設備都能保持數據同步，並且重要的數據都能夠備份在雲端。

本章節會介紹幾種不同的策略來實現具有云功能的App。包括：使用你自己的後端網絡應用進行數據云同步，以及使用雲對數據進行備份，這樣的話，當用戶將你的app安裝到一臺新的設備上時，他們之前的使用數據就可以得到恢復了。

## Lessons

---

- [使用備份API](#)

學習如何將Backup API集成到你的應用中。通過Backup API可以將用戶數據（比如配置信息，筆記，高分記錄等）無縫地在多臺設備上進行同步更新。

- [使用Google Cloud Messaging](#)

學習如何高效的發送多播消息，如何正確地響應接收到的Google Cloud Messaging (GCM) 消息，以及如何使用GCM消息與服務器進行高效同步。

# 使用備份API

編寫:kesenhoo - 原文:<http://developer.android.com/training/cloudsync/backupapi.html>

當用戶購買了一臺新的設備或者是對當前的設備做了的恢復出廠設置的操作，用戶會希望在進行初始化設置的時候，Google Play能夠把之前安裝過的應用恢復到設備上。默認情況是，用戶的這些期望並不會發生，他們之前的設置與數據都會丟失。

對於一些數據量相對較少的情況(通常少於1MB)，例如用戶偏好設置，筆記，遊戲分數或者是其他的一些狀態數據，可以使用Backup API來提供一個輕量級的解決方案。這節課會介紹如何將Backup API集成到你的應用當中，以及如何利用Backup API將數據恢復到新的設備上。

## 註冊Android Backup Service

這節課中所使用的Android Backup Service需要進行註冊。你可以點擊[這裏](#)進行註冊。註冊成功後，服務器會提供一段類似於下面的代碼，你需要將它添加到應用的Manifest文件中：

```
<meta-data android:name="com.google.android.backup.api_key"  
    android:value="ABcDe1FGHij2KlmN3oPQRs4TUVw5XYZ" />
```

請注意，每一個備份Key都只能在特定的包名下工作，如果你有不同的應用需要使用這個方法進行備份，那麼需要為他們分別進行註冊。

## 配置你的Manifest清單文件

使用Android的備份服務需要將兩個額外的內容添加到應用的Manifest清單文件中。首先，聲明你的備份代理的類名，然後添加一段類似上面的代碼作為Application標籤的子標籤。假設你的備份代理叫作TheBackupAgent，下面的例子展示瞭如何在Manifest清單文件中添加這些信息：

```
<application android:label="MyApp"  
    android:backupAgent="TheBackupAgent">  
    ...  
    <meta-data android:name="com.google.android.backup.api_key"  
        android:value="ABcDe1FGHij2KlmN3oPQRs4TUVw5XYZ" />  
    ...  
</application>
```

## 編寫你的備份代理

創建備份代理最簡單的方法是繼承BackupAgentHelper。創建這個幫助類實際上非常簡便。首先創建一個類，其類名和你在上述Manifest清單文件中聲明的類名一致（本例中，它叫做TheBackupAgent），然後繼承BackupAgentHelper，之後重寫onCreate()方法。在onCreate()中創建一個BackupHelper。這些幫助類是專門用來備份某些數據的，目前Android Framework包含了兩種幫助類：FileBackupHelper與SharedPreferencesBackupHelper。在你創建一個幫助類並且指向需要備份的數據的時候，僅僅需要使用addHelper()方法將它們添加到BackupAgentHelper當中，之後再增加一個Key用來恢復數據。大多數情況下，完整的實現差不多只需要10行左右的代碼。

下面是一個對高分數據進行備份的例子：

```

import android.app.backup.BackupAgentHelper;
import android.app.backup.FileBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    // The name of the SharedPreferences file
    static final String HIGH_SCORES_FILENAME = "scores";

    // A key to uniquely identify the set of backup data
    static final String FILES_BACKUP_KEY = "myfiles";

    // Allocate a helper and add it to the backup agent
    @Override
    void onCreate() {
        FileBackupHelper helper = new FileBackupHelper(this, HIGH_SCORES_FILENAME);
        addHelper(FILES_BACKUP_KEY, helper);
    }
}

```

為了使得程序更加靈活，[FileBackupHelper](#)的構造函數可以帶有任意數量的文件名。你只需簡單地通過增加一個額外的參數，就能實現同時對最高分文件與遊戲進度文件進行備份，如下所述：

```

@Override
void onCreate() {
    FileBackupHelper helper = new FileBackupHelper(this, HIGH_SCORES_FILENAME, PROGRESS_FILENAME);
    addHelper(FILES_BACKUP_KEY, helper);
}

```

備份用戶偏好同樣比較簡單。和創建[FileBackupHelper](#)一樣來創建一個[SharedPreferencesBackupHelper](#)。在這種情況下，不是添加文件名到構造函數當中，而是添加被你的應用所使用的Shared Preference Groups的名稱。下面的例子展示的是，如果高分數據是以Preference的形式而非文件的形式存儲的，你的備份代理幫助類應該如何設計：

```

import android.app.backup.BackupAgentHelper;
import android.app.backup.SharedPreferencesBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    // The names of the SharedPreferences groups that the application maintains. These
    // are the same strings that are passed to getSharedPreferences(String, int).
    static final String PREFS_DISPLAY = "displayprefs";
    static final String PREFS_SCORES = "highscores";

    // An arbitrary string used within the BackupAgentHelper implementation to
    // identify the SharedPreferencesBackupHelper's data.
    static final String MY_PREFS_BACKUP_KEY = "myprefs";

    // Simply allocate a helper and install it
    void onCreate() {
        SharedPreferencesBackupHelper helper =
            new SharedPreferencesBackupHelper(this, PREFS_DISPLAY, PREFS_SCORES);
        addHelper(MY_PREFS_BACKUP_KEY, helper);
    }
}

```

雖然你可以根據你的喜好增加任意數量的備份幫助類到你的備份代理幫助類中，但是請記住每一種類型的備份幫助類只需要一個就夠了。一個[FileBackupHelper](#)可以處理所有你想要備份的文件，而一個[SharedPreferencesBackupHelper](#)則能夠處理所有你想要備份的Shared Preference Groups。

## 請求備份

---

為了請求一個備份，僅僅需要創建一個[BackupManager](#)的實例，然後調用它的[dataChanged\(\)](#)方法即可：

```
import android.app.backup.BackupManager;  
...  
  
public void requestBackup() {  
    BackupManager bm = new BackupManager(this);  
    bm.dataChanged();  
}
```

該調用會告知備份管理器即將有數據會被備份到雲端。在之後的某個時間點，備份管理器會執行備份代理的[onBackup\(\)](#)方法。無論任何時候，只要你的數據發生了改變，你都可以去調用它，並且你不用擔心這樣會增加網絡的負荷。如果你在備份正式發生之前請求了兩次備份，那麼最終備份操作僅僅會出現一次。

## 恢復備份數據

一般而言，你不應該手動去請求恢復，而是應該讓應用安裝到設備上的時候自動進行恢復。然而，如果確實有必要手動去觸發恢復，只需要調用[requestRestore\(\)](#)方法就可以了。

# 使用Google Cloud Messaging

編寫:jdneo - 原文:<http://developer.android.com/training/cloudsync/gcm.html>

谷歌雲消息（GCM）是一個用來給Android設備發送消息的免費服務，它可以極大地提升用戶體驗。利用GCM消息，你的應用可以一直保持更新的狀態，同時不會使你的設備在服務器端沒有可用更新時，喚醒無線電並對服務器發起輪詢（這會消耗大量的電量）。同時，GCM可以讓你最多一次性將一條消息發送給1,000個人，使得你可以在恰當地時機很輕鬆地聯繫大量的用戶，同時大量地減輕你的服務器負擔。

這節課將包含一些把GCM集成到應用中的最佳實踐方法，前提是假定你已經對該服務的基本實現有了一個瞭解。如果不是這樣的話，你可以先閱讀一下：[GCM demo app tutorial](#)。

## 高效地發送多播消息

一個GCM最有用的特性之一是單條消息最多可以發送給1,000個接收者。這個功能可以更加簡單地將重要消息發送給你的所有用戶羣體。例如，比方說你有一條消息需要發送給1,000,000個人，而你的服務器每秒能發送500條消息。如果你的每條消息只能發送給一個接收者，那麼整個消息發送過程將會耗時 $1,000,000/500=2,000$ 秒，大約半小時。然而，如果一條消息可以一次性地發送給1,000個人的話，那麼耗時將會是 $(1,000,000/1,000)/500=2$ 秒。這不僅僅體現出了GCM的實用性，同時對於一些實時消息而言，其重要性也是不言而喻的。就比如災難預警或者體育比分播報，如果延遲了30分鐘，消息的價值就大打折扣了。

想要利用這一功能非常簡單。如果你使用的是Java語言版本的[GCM helper library](#)，只需要向 `send` 或者 `sendNoRetry` 方法提供一個註冊ID的List就行了（不要只給單個的註冊ID）：

```
// This method name is completely fabricated, but you get the idea.  
List regIds = whoShouldISendThisTo(message);  
  
// If you want the SDK to automatically retry a certain number of times, use the  
// standard send method.  
MulticastResult result = sender.send(message, regIds, 5);  
  
// Otherwise, use sendNoRetry.  
MulticastResult result = sender.sendNoRetry(message, regIds);
```

如果想用除了Java之外的語言實現GCM支持，可以構建一個帶有下列頭部信息的HTTP POST請求：

```
Authorization: key=YOUR_API_KEY  
Content-type: application/json
```

之後將你想要使用的參數編碼成一個JSON對象，列出所有在 `registration_ids` 這個Key下的註冊ID。下面的代碼片段是一個例子。除了 `registration_ids` 之外的所有參數都是可選的，在 `data` 內的項目代表了用戶定義的載荷數據，而非GCM定義的參數。這個HTTP POST消息將會發送到：<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"]}
```

```
}
```

關於更多GCM多播消息的格式，可以閱讀：[Sending Messages](#)。

## 對可替換的消息執行摺疊

GCM經常被用作爲一個觸發器，它告訴移動應用向服務器發起鏈接並更新數據。在GCM中，可以（也推薦）在新消息要替代舊消息時，使用可摺疊的消息（Collapsible Messages）。我們用體育比賽作爲例子，如果你向所有用戶發送了一條包含了當前比賽比分的消息，15分鐘之後，又發送了一條消息更新比分，那麼第一條消息就沒有意義了。對於那些還沒有收到第一條消息的用戶，就沒有必要將這兩條消息全部接收下來，何況如果要接收兩條消息，那麼設備不得不進行兩次響應（比如對用戶發出通知或警告），但實際上兩條消息中只有一條是重要的。

當你定義了一個摺疊Key，此時如果有多個消息在GCM服務器中，以隊列的形式等待發送給同一個用戶，那麼只有最後的那一條消息會被發出。對於之前所說的體育比分的例子，這樣做能讓設備免於處理不必要的任務，也不會讓設備對用戶造成太多打擾。對於其他的一些場景比如與服務器同步數據（檢查郵件接收），這樣做的話可以減少設備需要執行同步的次數。例如，如果有10封郵件在服務器中等待被接收，並且有10條GCM消息發送到設備提醒它有新的郵件，那麼實際上只需要一個GCM就夠了，因爲設備可以一次性把10封郵件都同步了。

爲了使用這一特性，只需要在你要發出的消息中添加一個消息摺疊Key。如果你在使用[GCM helper library](#)，那麼就使用Message類的 collapseKey(String key) 方法。

```
Message message = new Message.Builder(regId)
    .collapseKey("game4_scores") // The key for game 4.
    .ttl(600) // Time in seconds to keep message queued if device offline.
    .delayWhileIdle(true) // Wait for device to become active before sending.
    .addPayload("key1", "value1")
    .addPayload("key2", "value2")
    .build();
```

如果你沒有使用[GCM helper library](#)，那麼就直接在你要構建的POST頭部中添加一個字段。將 collapse\_key 作爲字段名，並將Key的名稱作爲該字段的值。

## 在GCM消息中嵌入數據

通常，GCM消息被用作爲一個觸發器，或者用來告訴設備，在服務器或者別的地方有一些待更新的數據。然而，一條GCM消息的大小最大可以有4kb，因此，有時候可以在GCM消息中放置一些簡單的數據，這樣的話設備就不需要再去和服務器發起連接了。在下列條件都滿足的情況下，我們可以將數據放置在GCM消息中：

- 數據的總大小在4kb以內。
- 每一條消息都很重要，且需要保留。
- 這些消息不適用於消息摺疊的使用情形。

例如，短消息或者回合制網遊中玩家的移動數據等都是將數據直接嵌入在GCM消息中的例子。而電子郵件就是反面例子了，因爲電子郵件的數據量一般都大於4kb，而且用戶一般不需要對每一封新郵件都收到一個GCM提醒的消息。

同時在發送多播消息時，也可以考慮這一方法，這樣的話就不會導致大量用戶在接收到GCM的更新提醒後，同時向你的服務器發起連接。

這一策略不適用於發送大量的數據，有這麼一些原因：

- 為了防止惡意軟件發送垃圾消息，GCM有發送頻率的限制。

- 無法保證消息按照既定的發送順序到達。
- 無法保證消息可以在你發送後立即到達。假設設備每一秒都接收一條消息，消息的大小限制在1K，那麼傳輸速率為8kbps，或者說是1990年代的家庭撥號上網的速度。那麼如此大量的消息，一定會讓你的應用在Google Play上的評分非常尷尬。

如果恰當地使用，直接將數據嵌入到GCM消息中，可以加速你的應用的“感知速度”，因為這樣一來它就不必再去服務器獲取數據了。

## 智能地響應GCM消息

---

你的應用不應該僅僅對收到的GCM消息進行響應就夠了，還應該響應地更智能一些。至於如何響應需要結合具體情況而定。

### 不要太過激進

當提醒用戶去更新數據時，很容易不小心從“有用的消息”變成“干擾消息”。如果你的應用使用狀態欄通知，那麼應該**更新現有的通知**，而不是創建第二個。如果你通過鈴聲或者震動的方式提醒用戶，一定要設置一個計時器。不要讓應用每分鐘的提醒頻率超過1次，不然的話用戶很可能會不堪其擾而卸載你的應用，關機，甚至把手機扔到河裏。

### 用聰明的辦法同步數據，別用笨辦法

當使用GCM告知設備有數據需要從服務器下載時，記住你有4kb大小的數據可以和消息一起發出，這可以幫助你的應用做出更智能的響應。例如，如果你有一個支持訂閱的閱讀應用，而你的用戶訂閱了100個源，那麼這就可以幫助你的應用更智能地決定應該去服務器下載什麼數據。下面的例子說明了在GCM載荷中可以發送什麼樣的數據，以及設備可以做出什麼樣的反應：

- `refresh` - 你的應用被告知向每一個源請求數據。此時你的應用可以向100個不同的服務器發起獲取訂閱內容的請求，或者如果你在服務器上有一個聚合服務，那麼可以只發送一個請求，將100個源的數據進行打包並讓設備獲取，這樣一次性就完成更新。
- `refresh, freshID` - 一種更好的解決方案，你的應用可以有針對性的完成更新。
- `refresh, freshID, timestamp` - 三種方案中最好的，如果正好用戶在收到GCM消息之前手動做了更新，那麼應用可以利用時間戳和當前的更新時間進行對比，並決定是否有必要執行下一步的行動。

# 解決雲同步的保存衝突

編寫:jdneo - 原文:<http://developer.android.com/training/cloudsave/conflict-res.html>

這篇文章介紹了當應用使用Cloud Save service存儲數據到雲端時，如何設計一個魯棒性較高的衝突解決策略。雲存儲服務允許你為每一個在Google服務上的應用用戶，存儲他們的應用數據。你的應用可以通過使用雲存儲API，從Android設備，iOS設備或者web應用恢復或更新這些數據。

雲存儲中的保存和加載過程非常直接：它只是一個數據和byte數組之間序列化轉換，並將這些數組存儲在雲端的過程。然而，當你的用戶有多個設備，並且有兩個以上的設備嘗試將它們的數據存儲在雲端時，這一保存可能會引起衝突，因此你必須決定應該如何處理這類問題。雲端數據的結構在很大程度上決定了衝突解決方案的魯棒性，所以務必小心地設計你的數據存儲結構，使得衝突解決方案的邏輯可以正確地處理每一種情況。

本篇文章從一些有缺陷的解決方案入手，並解釋他們為何具有缺陷。之後會呈現一個可以避免衝突的解決方案。用於討論的例子關注於遊戲，但解決問題的核心思想是可以適用於任何將數據存儲於雲端的應用的。

## 衝突時獲得通知

OnStateLoadedListener方法負責從Google服務器下載應用的狀態數據。回調函

數OnStateLoadedListener.onStateConflict用來給你的應用在本地狀態和雲端存儲的狀態發生衝突時，提供了一個解決機制：

```
@Override  
public void onStateConflict(int stateKey, String resolvedVersion,  
    byte[] localData, byte[] serverData) {  
    // resolve conflict, then call mAppStateClient.resolveConflict()  
    ...  
}
```

此時你的應用必須決定要保留哪一個數據，或者它自己提交一個新的數據來表示合併後的數據狀態，解決衝突的邏輯由你自己來實現。

我們必須要意識到雲存儲服務是在後臺執行同步的。所以你應該確保你的應用能夠在你創建這一數據的Context之外接收回調。特別地，如果Google Play服務應用在後臺檢測到了一個衝突，該回調函數會在你下一次加載數據時被調用，通常來說會是在下一次用戶啓動該應用時。

因此，你的雲存儲代碼和衝突解決代碼的設計必須是和當前context無關的：也就是說當你拿到了兩個衝突的數據，你必須僅通過數據集內獲取的數據去解決衝突，而不依賴於任何其它任何外部Context。

## 處理簡單情況

下面列舉一些解決衝突的簡單例子。對於很多應用而言，用這些策略或者其變體就足夠解決大多數問題了：

新的比舊的更有效：在一些情況下，新的數據可以替代舊的數據。例如，如果數據代表了用戶所選擇角色的衣服顏色，那麼最近的新的選擇就應該覆蓋老的選擇。在這種情況下，你可能會選擇在雲存儲數據中存儲時間戳。當處理這些衝突時，選擇時間戳最新的數據（記住要選擇一個可靠的時鐘，並注意對不同時區的處理）。

一個數據好於其他數據：在一些情況下，我們是可以有方法在若干數據集中選取一個最好的。例如，如果數據代表了玩家在賽車比賽中的最佳時間，那麼顯然，在衝突發生時，你應該保留成績最好的那個數據。

進行合併：有可能通過計算兩個數據集的合併版本來解決衝突。例如，你的數據代表了用戶解鎖關卡的進度，那麼我們需要的數據就是兩個衝突數據的並集。通過這個方法，用戶的關卡解鎖進度就不會丟失了。這裏的[例子](#)使用了這一策略的一個變形。

## 爲更複雜的情況設計一個策略

當你的遊戲允許玩家收集可交換物品時（比如金幣或者經驗點數），情況會變得更加複雜一些。我們來假想一個遊戲，叫做“金幣跑酷”，遊戲中的角色通過跑步不斷地收集金幣使自己變的富有。每個收集到的金幣都會加入到玩家的儲蓄罐中。

下面的章節將展示三種在多個設備間解決衝突的方案：有兩個看上去還不錯，可惜最終還是不能適用於所有情況，最後一個解決方案可以解決多個設備間的數據衝突。

### 第一個嘗試：只保存總數

首先，這個問題看上去像是說：雲存儲的數據只要存儲金幣的數量就行了。但是如果就只有這些數據是可用的，那麼解決衝突的方案將會嚴重受到限制。此時最佳的方案只能是在衝突發生時存儲數值最大的數據。

想一下表1中所展現的場景。假設玩家一開始有20枚硬幣，然後在設備A上收集了10個，在設備B上收集了15個。然後設備B將數據存儲到了雲端。當設備A嘗試去存儲的時候，衝突發生了。“只保存總數”的衝突解決方案會存儲35作為這一數據的值（兩數之間最大的）。

表1. 值保存最大的數（不佳的策略）

事件	設備A的數據	設備B的數據	雲端的數據	實際的總數
開始階段	20	20	20	20
玩家在A設備上收集了10個硬幣	30	20	20	30
玩家在B設備上收集了15個硬幣	30	35	20	45
設備B將數據存儲至雲端	30	35	35	45
設備A嘗試將數據存儲至雲端，發生衝突	30	35	35	45
設備A通過選擇兩數中最大的數來解決衝突	35	35	35	45

這一策略顯然會失敗：玩家的金幣數從20變成35，但實際上玩家總共收集了25個硬幣（A設備10個，B設備15個），所以有10個硬幣丟失了。只在雲端存儲硬幣的總數是不足以實現一個健壯的衝突解決算法的。

### 第二個嘗試：存儲總數和變化值

另一個方法是在存儲數據中包括一些額外的數據，如：自上次提交後硬幣增加的數量（delta）。在這一方法中，存儲的數據可以用一個二元組來表示（T, d），其中T是硬幣的總數，而d是硬幣增加的數量。

通過這樣的數據存儲結構，你的衝突檢測算法在魯棒性上會有更大的提升空間。但是這個方法在某些情況下依然會存在問題。

下面是包含delta數值的衝突解決算法過程：

- 本地數據：(T, d)
- 雲端數據：(T', d')
- 解決後的數據：(T'+d, d)

例如，當你在本地狀態 ( $T, d$ ) 和雲端狀態 ( $T', d$ ) 之間發生了衝突時，你可以將它們合併成 ( $T'+d, d$ )。意味着你從本地拿出delta數據，並將它和雲端的數據結合起來，乍一看，這種方法可以很好的計量多個設備所收集的金幣。

該方法看上去很可靠，但它在具有移動網絡的環境中難以適用：

- 用戶可能在設備不在線時存儲數據。這些改變會以隊列形式等待手機聯網後提交。
- 這個方法的同步機制是用最新的變化覆蓋掉任何之前的變化。換句話說，第二次寫入的變化會提交到雲端（當設備聯網了以後），而第一次寫入的變化就被忽略了。

為了進一步說明，我們考慮一下表2所列的場景。在表2列出的一系列操作發生後，雲端的狀態將是 (130, +5)，最終衝突解決後的狀態是 (140, +10)。這是不正確的，因為從總體上而言，用戶一共在A上收集了110枚硬幣而在B上收集了120枚硬幣。總數應該為250。

表2. “總數+增量”策略的失敗案例

事件	設備A的數據	設備B的數據	雲端的數據	實際的總數
開始階段	(20, x)	(20, x)	(20, x)	20
玩家在A設備上收集了100個硬幣	(120, +100)	(20, x)	(20, x)	120
玩家在A設備上又收集了10個硬幣	(130, +10)	(20, x)	(20, x)	130
玩家在B設備上收集了115個硬幣	(130, +10)	(125, +115)	(20, x)	245
玩家在B設備上又收集了5個硬幣	(130, +10)	(130, +5)	(20, x)	250
設備B將數據存儲至雲端	(130, +10)	(130, +5)	(130, +5)	250
設備A嘗試將數據存儲至雲端，發生衝突	(130, +10)	(130, +5)	(130, +5)	250
設備A通過將本地的增量和雲端的總數相加來解決衝突	(140, +10)	(130, +5)	(140, +10)	250

注：x代表與該場景無關的數據

你可能會嘗試在每次保存後不重置增量數據來解決此問題，這樣的話在每個設備上第二次存儲的數據就能夠代表用戶至今為止收集到的所有硬幣。此時，設備A在第二次本地存儲完成後，數據將是 (130, +110) 而不是 (130, +10)。然而，這樣做的話就會發生如表3所述的情況：

表3. 算法改進後的失敗案例

事件	設備A的數據	設備B的數據	雲端的數據	實際的總數
開始階段	(20, x)	(20, x)	(20, x)	20
玩家在A設備上收集了100個硬幣	(120, +100)	(20, x)	(20, x)	120
設備A將狀態存儲到雲端	(120, +100)	(20, x)	(120, +100)	120
玩家在A設備上又收集了10個硬幣	(130, +110)	(20, x)	(120, +100)	130
玩家在B設備上收集了1個硬幣	(130,	(21, +1)	(120,	131

玩家在B設備上收集了1個硬幣	+110)	(21, +1)	+100)	131
設備B嘗試向雲端存儲數據，發生衝突	(130, +110)	(21, +1)	(120, +100)	131
設備B通過將本地的增量和雲端的總數相加來解決衝突	(130, +110)	(121, +1)	(121, +1)	131
設備A嘗試將數據存儲至雲端，發生衝突	(130, +110)	(121, +1)	(121, +1)	131
設備A通過將本地的增量和雲端的總數相加來解決衝突	(231, +110)	(121, +1)	(231, +110)	131

注：x代表與該場景無關的數據

現在你碰到了另一個問題：你給予了玩家過多的硬幣。這個玩家拿到了211枚硬幣，但實際上他只收集了111枚。

## 解決辦法：

分析之前的幾次嘗試，我們發現這些策略存在這樣的缺陷：無法知曉哪些硬幣已經計數了，哪些硬幣沒有被計數，尤其是當多個設備連續提交的時候，算法會出現混亂。

該問題的解決辦法是將你在雲端的數據存儲結構改為字典類型，使用字符串+整形的鍵值對。每一個鍵值對都代表了一個包含硬幣的“委託人”，而總數就應該是將所有記錄的值加起來。這一設計的宗旨是每個設備有它自己的“委託人”，並且只有設備自己可以把硬幣放到它的“委託人”當中。

字典的結構是： $(A:a, B:b, C:c, \dots)$ ，其中a代表了“委託人”A所擁有的硬幣，b是“委託人”B所擁有的硬幣，以此類推。

這樣的話，新的衝突解決策略算法將如下所示：

- 本地數據： $(A:a, B:b, C:c, \dots)$
- 雲端數據： $(A:a', B:b', C:c', \dots)$
- 解決後的數據： $(A:\max(a,a'), B:\max(b,b'), C:\max(c,c'), \dots)$

例如，如果本地數據是(A:20, B:4, C:7)並且雲端數據是(B:10, C:2, D:14)，那麼解決衝突後的數據將會是(A:20, B:10, C:7, D:14)。當然，應用的衝突解決邏輯可以根據具體的需求而有所差異。比如，有一些應用你可能希望挑選最小的值。

為了測試新的算法，將它應用於任何一個之前提到過的場景。你將會發現它都能取得正確地結果。

表4闡述了這一點，它使用了表3中所提到的場景。注意下面所列出的關鍵點：

在初始狀態，玩家有20枚硬幣。該數據準確體現在了所有設備和雲端中，我們用字典： $(X:20)$ 來代表它，其中X我們不用太多關心，初始化的數據是哪兒來對該問題沒有影響。

當玩家在設備A上收集了100枚硬幣，這一變化會作為一個字典保存到雲端。字典的值是100是因為這就是玩家在設備A上收集的硬幣數量。在這一過程中，沒有要執行數據的計算（設備A僅僅是將玩家所收集的數據彙報給了雲端）。

每一個新的硬幣提交會打包成一個與設備關聯的字典並保存到雲端。例如，假設玩家又在設備A上收集了100枚硬幣，那麼對應字典的值被更新為110。

最終的結果就是，應用知道了玩家在每個設備上收集硬幣的總數。這樣它就能輕易地計算出實際的總數了。

表4. 鍵值對策略的成功應用案例

事件	設備A的數據	設備B的數據	雲端的數據	實際的總數
開始階段	(X:20, x)	(X:20, x)	(X:20, x)	20
玩家在A設備上收集了100個硬幣	(X:20, A:100)	(X:20)	(X:20)	120
設備A將狀態存儲到雲端	(X:20, A:100)	(X:20)	(X:20, A:100)	120
玩家在A設備上又收集了10個硬幣	(X:20, A:110)	(X:20)	(X:20, A:100)	130
玩家在B設備上收集了1個硬幣	(X:20, A:110)	(X:20, B:1)	(X:20, A:100)	131
設備B嘗試向雲端存儲數據，發生衝突	(X:20, A:110)	(X:20, B:1)	(X:20, A:100)	131
設備B解決衝突	(X:20, A:110)	(X:20, A:100, B:1)	(X:20, A:100, B:1)	131
設備A嘗試將數據存儲至雲端，發生衝突	(X:20, A:110)	(X:20, A:100, B:1)	(X:20, A:100, B:1)	131
設備A解決衝突	(X:20, A:110, B:1)	(X:20, A:100, B:1)	(X:20, A:110, B:1), total 131	131

## 清除你的數據

---

在雲端允許存儲數據的大小是有限制的，所以在後續的論述中，我們將會關注如何避免創建過大的詞典。一開始，看上去每個設備只會有一條詞典記錄，即使是非常激進的用戶也不太會擁有上千種不同的設備（對應上千條字典記錄）。然而，獲取設備ID的方法很難，並且我們認為這是一種不好的實踐方式，所以你應該使用一個安裝ID，這更容易獲取也更可靠。這樣的話就意味着，每一次用戶在每臺設備安裝一次就會產生一個ID。假設每個鍵值對佔據32字節，由於一個人云存儲緩存最多可以有128K的大小，那麼你最多可以存儲4096條記錄。

在現實場景中，你的數據可能更加複雜。在這種情況下，存儲數據的記錄條數也會進一步受到限制。具體而言則需要取決於實現，比如可能需要添加時間戳來指明每條記錄是何時修改的。當你檢測到有一條記錄在過去幾個禮拜或者幾個月的時間內都沒有被修改，那麼就可以安全地將金幣數據轉移到另一條記錄中並刪除老的記錄。

# 使用Sync Adapter傳輸數據

編寫:jdneo - 原文:<http://developer.android.com/training/sync-adapters/index.html>

如果你的應用允許Android設備和網絡服務器之間進行數據同步，那麼它無疑將變得更加實用，更加吸引用戶的注意。例如，將數據傳輸到服務器可以實現數據的備份，另一方面，從服務器獲取數據可以讓用戶隨時隨地都能使用你的應用。有時候，用戶可能會覺得在線編輯他們的數據並將其發送到設備上，會是一件很方便的事情；或者他們有時會希望將收集到的數據上傳到一個統一的存儲區域中。

儘管你可以設計一套自己的系統來實現應用中的數據傳輸，但你也可以考慮一下使用Android的同步適配器框架（Android's Sync Adapter Framework）。該框架可以用來幫助管理數據，自動傳輸數據，以及協調不同應用間的同步問題。當你使用這個框架時，你可以利用它的一些特性，而這些特性可能是你自己設計的傳輸方案中所沒有的：

插件架構（Plug-in Architecture）：

允許你以可調用組件的形式，將傳輸代碼添加到系統中。

自動執行（Automated Execution）：

允許你基於不同的準則自動地執行數據傳輸，比如：當數據變更時，或者每隔固定一段時間，亦或者每天，來自動執行一次數據傳輸。另外，系統會自動把當前無法執行的傳輸添加到一個隊列中，並且在合適的時候運行它們。

自動網絡監測（Automated Network Checking）：

系統只在有網絡連接的時候纔會運行數據傳輸。

提升電池使用效率：

允許你將所有的數據傳輸任務統一地進行一次性批量傳輸，這樣的話多個數據傳輸任務會在同一段時間內運行。你的應用的數據傳輸任務也會和其它應用的傳輸任務相結合，並一起傳輸。這樣做可以減少系統連接網絡的次數，進而減少電量的使用。

帳戶管理和授權：

如果你的應用需要用戶登錄授權，那麼你可以將帳戶管理和授權的功能集成到你的數據傳輸組件中。

本系列課程將向你展示如何創建一個Sync Adapter，如何創建一個綁定了Sync Adapter的服務（Service），如何提供其它組件來幫助你將Sync Adapter集成到框架中，以及如何通過不同的方法來運行Sync Adapter。

\*\* Note : Sync Adapter是異步執行的，它可以定期且有效地傳輸數據，但在實時性上一般難以滿足要求。如果你想要實時地傳輸數據，那麼你應該在[AsyncTask](#)或[IntentService](#)中完成這一任務。

## Sample Code

[BasicSyncAdapter.zip](#)

## Lessons

- [創建Stub授權器](#)

學習如何在你的應用中添加一個Sync Adapter框架需要的賬戶處理組件。這節課將向你展示如何簡單地創建一個Stub Authenticator組件。

- [創建Stub Content Provider](#)

學習如何在你的應用中添加一個Sync Adapter框架需要的Content Provider組件。在這節課中，我們假設你的應用實際上不需要使用Content Provider，所以它將教你如何添加一個Stub組件。如果你的應用已經有了一個Content Provider組件，那麼可以跳過這節課。

- [創建Sync Adapter](#)

學習如何將你的數據傳輸代碼封裝到組件當中，並讓其可以被Sync Adapter框架自動執行。

- [執行Sync Adapter](#)

學習如何使用Sync Adapter框架激活並調度數據傳輸。

# 創建Stub授權器

編寫:jdneo - 原文:<http://developer.android.com/training/sync-adapters/creating-authenticator.html>

Sync Adapter框架假定你的Sync Adapter在同步數據時，設備存儲端關聯了一個賬戶，且服務器端需要進行登錄驗證。因此，你需要提供一個叫做授權器（Authenticator）的組件作為Sync Adapter的一部分。該組件會集成在Android賬戶及認證框架中，並提供一個標準的接口來處理用戶憑據，比如登錄信息。

如果你的應用不使用賬戶，你仍然需要提供一個授權器組件。在這種情況下，授權器所處理的信息將被忽略，所以你可以提供一個包含了方法存根（Stub Method）的授權器組件。同時你需要提供一個綁定Service，來允許Sync Adapter框架來調用授權器的方法。

這節課將向你展示如何定義一個能夠滿足Sync Adapter框架要求的Stub授權器。如果你想要提供可以處理用戶賬戶的實際的授權器，可以閱讀：[AbstractAccountAuthenticator](#)。

## 添加一個Stub授權器組件

要在你的應用中添加一個Stub授權器，首先我們需要創建一個繼承[AbstractAccountAuthenticator](#)的類，在所有需要重寫的方法中，我們不進行任何處理，僅返回null或者拋出異常。

下面的代碼片段是一個Stub授權器的例子：

```
/*
 * Implement AbstractAccountAuthenticator and stub out all
 * of its methods
 */
public class Authenticator extends AbstractAccountAuthenticator {
    // Simple constructor
    public Authenticator(Context context) {
        super(context);
    }
    // Editing properties is not supported
    @Override
    public Bundle editProperties(
            AccountAuthenticatorResponse r, String s) {
        throw new UnsupportedOperationException();
    }
    // Don't add additional accounts
    @Override
    public Bundle addAccount(
            AccountAuthenticatorResponse r,
            String s,
            String s2,
            String[] strings,
            Bundle bundle) throws NetworkErrorException {
        return null;
    }
    // Ignore attempts to confirm credentials
    @Override
    public Bundle confirmCredentials(
            AccountAuthenticatorResponse r,
            Account account,
            Bundle bundle) throws NetworkErrorException {
        return null;
    }
    // Getting an authentication token is not supported
    @Override
    public Bundle getAuthToken(
            AccountAuthenticatorResponse r,
            Account account,
```

```

        String s,
        Bundle bundle) throws NetworkErrorException {
    throw new UnsupportedOperationException();
}
// Getting a label for the auth token is not supported
@Override
public String getAuthTokenLabel(String s) {
    throw new UnsupportedOperationException();
}
// Updating user credentials is not supported
@Override
public Bundle updateCredentials(
    AccountAuthenticatorResponse r,
    Account account,
    String s, Bundle bundle) throws NetworkErrorException {
    throw new UnsupportedOperationException();
}
// Checking features for the account is not supported
@Override
public Bundle hasFeatures(
    AccountAuthenticatorResponse r,
    Account account, String[] strings) throws NetworkErrorException {
    throw new UnsupportedOperationException();
}
}
}

```

## 將授權器綁定到框架

為了讓Sync Adapter框架可以訪問你的授權器，你必須為它創建一個綁定服務。這一服務提供一個Android Binder對象，允許框架調用你的授權器，並且在授權器和框架間傳遞數據。

因為框架會在它第一次需要訪問授權器時啓動該Service，你也可以使用該服務來實例化授權器，具體而言，我們需要在服務的Service.onCreate())方法中調用授權器的構造函數。

下面的代碼樣例展示瞭如何定義綁定Service：

```

/**
 * A bound Service that instantiates the authenticator
 * when started.
 */
public class AuthenticatorService extends Service {
    ...
    // Instance field that stores the authenticator object
    private Authenticator mAuthenticator;
    @Override
    public void onCreate() {
        // Create a new authenticator object
        mAuthenticator = new Authenticator(this);
    }
    /*
     * When the system binds to this Service to make the RPC call
     * return the authenticator's IBinder.
     */
    @Override
    public IBinder onBind(Intent intent) {
        return mAuthenticator.getIBinder();
    }
}

```

## 添加授權器的元數據（Metadata）文件

若要將你的授權器組件集成到Sync Adapter框架和賬戶框架中，你需要為這些框架提供帶有描述組件信息的元數據。該

元數據聲明瞭你爲Sync Adapter創建的賬戶類型以及系統所顯示的UI元素（如果你希望用戶可以看到你創建的賬戶類型）。在你的項目目錄 `/res/xml/` 下，將元數據聲明於一個XML文件中。你可以自己爲該文件按命名，通常我們將它命名爲 `authenticator.xml`。

在這個XML文件中，包含了一個 `<account-authenticator>` 標籤，它有下列一些屬性：

#### android:accountType

Sync Adapter框架要求每一個適配器都有一個域名形式的賬戶類型。框架會將它作爲Sync Adapter內部標識的一部分。如果服務端需要登陸，賬戶類型會和賬戶一起發送到服務端作爲登錄憑據的一部分。

如果你的服務端不需要登錄，你仍然需要提供一個賬戶類型（該屬性的值用你能控制的一個域名即可）。雖然框架會使用它來管理Sync Adapter，但該屬性的值不會發送到你的服務端。

#### android:icon

指向一個包含圖標的Drawable資源。如果你在 `res/xml/syncadapter.xml` 中通過指定 `android:userVisible="true"` 讓Sync Adapter可見，那麼你必須提供圖標資源。它會在系統的設置中的賬戶（Accounts）這一欄內顯示。

#### android:smallIcon

指向一個包含微小版本圖標的Drawable資源。當屏幕尺寸較小時，這一資源可能會替代 `android:icon` 中所指定的圖標資源。

#### android:label

指明瞭用戶賬戶類型的本地化String。如果你在 `res/xml/syncadapter.xml` 中通過指定 `android:userVisible="true"` 讓Sync Adapter可見，那麼你需要提供該String。它會在系統的設置中的賬戶這一欄內顯示，就在你爲授權器定義的圖標旁邊。

下面的代碼樣例展示了你之前爲授權器創建的XML文件：

```
<?xml version="1.0" encoding="utf-8"?>
<account-authenticator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="example.com"
    android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher"
    android:label="@string/app_name"/>
```

## 在Manifest清單文件中聲明授權器

在之前的步驟中，你已經創建了一個綁定服務，將授權器和Sync Adapter框架連接了起來。爲了讓系統可以識別該服務，你需要在清單文件中添加 `<service>` 標籤，將它作爲 `<application>` 的子標籤：

```
<service
    android:name="com.example.android.syncadapter.AuthenticatorService">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator" />
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/authenticator" />
</service>
```

`<intent-filter>` 標籤配置了一個可以被 `android.accounts.AccountAuthenticator` 這一Action所激活的過濾器，這一Intent會在系統要運行授權器時由系統發出。當過濾器被激活後，系統會啓動 `AuthenticatorService`，即之前用來封裝授權器的Service。

`<meta-data>` 標籤聲明瞭授權器的元數據。`android:name`屬性將元數據和授權器框架連接起來。`android:resource`指定了你之前所創建的授權器元數據文件的名字。

除了授權器之外，Sync Adapter框架也需要一個Content Provider。如果你的應用並沒有使用Content Provider，可以閱讀下一節課程學習如何創建一個Stub Content Provider；如果你的應用已經使用了ContentProvider，可以直接閱讀：[創建Sync Adapter](#)。

# 創建Stub Content Provider

編寫:jdneo - 原文:<http://developer.android.com/training/sync-adapters/creating-stub-provider.html>

Sync Adapter框架是設計成用來和設備數據一起工作的，而這些設備數據應該被靈活且安全的Content Provider管理。因此，Sync Adapter框架會期望應用已經為它的本地數據定義了Content Provider。如果Sync Adapter框架嘗試去運行你的Sync Adapter，而你的應用沒有一個Content Provider的話，那麼你的Sync Adapter將會崩潰。

如果你正在開發一個新的應用，它將數據從服務器傳輸到一臺設備上，那麼你務必應該考慮將本地數據存儲於Content Provider中。除了它對於Sync Adapter的重要性之外，Content Provider還可以提供許多安全上的好處，更何況它是專門為了在Android設備上處理數據存儲而設計的。要學習如何創建一個Content Provider，可以閱讀：[Creating a Content Provider](#)。

然而，如果你已經通過別的形式來存儲本地數據了，你仍然可以使用Sync Adapter來處理數據傳輸。為了滿足Sync Adapter框架對於Content Provider的要求，可以在你的應用中添加一個Stub Content Provider。一個Stub Content Provider實現了Content Provider類，但是所有的方法都返回null或者0。如果你添加了一個Stub Content Provider，無論你的數據存儲機制是什麼，你都可以使用Sync Adapter來傳輸數據。

如果在你的應用中已經有了一個Content Provider，那麼你就不需要創建Stub Content Provider了。在這種情況下，你可以略過這節課程，直接進入：[創建Sync Adapter](#)。如果你還沒有創建Content Provider，這節課將向你展示如何通過添加一個Stub Content Provider，將你的Sync Adapter添加到框架中。

## 添加一個Stub Content Provider

要為你的應用創建一個Stub Content Provider，首先繼承[ContentProvider](#)類，並且在所有需要重寫的方法中，我們一律不進行任何處理而是直接返回。下面的代碼片段展示了你應該如何創建一個Stub Content Provider：

```
/*
 * Define an implementation of ContentProvider that stubs out
 * all methods
 */
public class StubProvider extends ContentProvider {
    /*
     * Always return true, indicating that the
     * provider loaded correctly.
     */
    @Override
    public boolean onCreate() {
        return true;
    }
    /*
     * Return an empty String for MIME type
     */
    @Override
    public String getType() {
        return new String();
    }
    /*
     * query() always returns no results
     */
    @Override
    public Cursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder) {
```

```
        return null;
    }
    /*
     * insert() always returns null (no URI)
     */
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }
    /*
     * delete() always returns "no rows affected" (0)
     */
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
    /*
     * update() always returns "no rows affected" (0)
     */
    public int update(
        Uri uri,
        ContentValues values,
        String selection,
        String[] selectionArgs) {
        return 0;
    }
}
```

## 在Manifest清單文件中聲明提供器

Sync Adapter框架會通過查看應用的清單文件中是否含有 `<provider>` 標籤，來驗證你的應用是否使用了Content Provider。為了在清單文件中聲明我們的Stub Content Provider，添加一個 `<provider>` 標籤，並讓它擁有下列屬性字段：

`android:name="com.example.android.datasync.provider.StubProvider"`

指定實現Stub Content Provider類的完整包名。

`android:authorities="com.example.android.datasync.provider"`

指定Stub Content Provider的URI Authority。用應用的包名加上字符串 `".provider"` 作為該屬性字段的值。雖然你在這裏向系統聲明瞭你的Stub Content Provider，但是這並不會導致對該Provider的訪問。

`android:exported="false"`

確定其它應用是否可以訪問Content Provider。對於Stub Content Provider而言，由於沒有讓其它應用訪問該Provider的必要，所以我們將該值設置為 `false`。該值並不會影響Sync Adapter框架和Content Provider之間的交互。

`android:syncable="true"`

該標識指明Provider是可同步的。如果將這個值設置為 `true`，你將不需要在代碼中調用`setIsSyncable()`。這一標識將會允許Sync Adapter框架和Content Provider進行數據傳輸，但是僅僅在你顯式地執行相關調用時，這一傳輸時纔會進行。

下面的代碼片段展示了你應該如何將 `<provider>` 標籤添加到應用的清單文件中：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.network.sync.BasicSyncAdapter"
    android:versionCode="1"
    android:versionName="1.0" >
```

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
<provider
    android:name="com.example.android.datasync.provider.StubProvider"
    android:authorities="com.example.android.datasync.provider"
    android:exported="false"
    android:syncable="true"/>
    ...
</application>
</manifest>
```

現在你已經創建了所有Sync Adapter框架所需要的依賴項，接下來你可以創建封裝數據傳輸代碼的組件了。該組件就叫做Sync Adapter。在下節課中，我們將會展示如何將這一組件添加到你的應用中。

# 創建Sync Adapter

編寫:jdneo - 原文:<http://developer.android.com/training/sync-adapters/creating-sync-adapter.html>

設備和服務器之間執行數據傳輸的代碼會封裝在應用的Sync Adapter組件中。Sync Adapter框架會基於你的調度和觸發操作，運行Sync Adapter組件中的代碼。要將同步適配組件添加到應用當中，你需要添加下列部件：

## Sync Adapter類

將你的數據傳輸代碼封裝到一個與Sync Adapter框架兼容的接口當中。

## 綁定Service

通過一個綁定服務，允許Sync Adapter框架運行Sync Adapter類中的代碼。

## Sync Adapter的XML元數據文件

該文件包含了有關Sync Adapter的信息。框架會根據該文件確定應該如何加載並調度你的數據傳輸任務。

## 應用清單文件的聲明

需要在應用的清單文件中聲明綁定服務；同時還需要指出Sync Adapter的元數據。

這節課將會向你展示如何定義他們。

# 創建一個Sync Adapter類

在這部分課程中，你將會學習如何創建封裝了數據傳輸代碼的Sync Adapter類。創建該類需要繼承Sync Adapter的基類；為該類定義構造函數；以及實現相關的方法，在這些方法中，我們定義數據傳輸任務。

## 繼承Sync Adapter基類：AbstractThreadedSyncAdapter

要創建Sync Adapter組件，首先繼承AbstractThreadedSyncAdapter，然後編寫它的構造函數。同使用Activity.onCreate()配置Activity時一樣，每次你的Sync Adapter組件重新被創建的時候，使用構造函數執行相關的配置。例如，如果你的應用使用一個Content Provider來存儲數據，那麼使用構造函數來獲取一個ContentResolver實例。由於從Android 3.0開始添加了第二種形式的構造函數，來支持 parallelSyncs 參數，所以你需要創建兩種形式的構造函數來保證兼容性。

Note : Sync Adapter框架是設計成和Sync Adapter組件的單例一起工作的。實例化Sync Adapter組件的更多細節，會在後面的章節中展開。

下面的代碼展示瞭如何實現AbstractThreadedSyncAdapter和它的構造函數：

```
/*
 * Handle the transfer of data between a server and an
 * app, using the Android sync adapter framework.
 */
public class SyncAdapter extends AbstractThreadedSyncAdapter {
    ...
    // Global variables
    // Define a variable to contain a content resolver instance
    ContentResolver mContentResolver;
```

```
/**  
 * Set up the sync adapter  
 */  
public SyncAdapter(Context context, boolean autoInitialize) {  
    super(context, autoInitialize);  
    /*  
     * If your app uses a content resolver, get an instance of it  
     * from the incoming Context  
     */  
    mContentResolver = context.getContentResolver();  
}  
...  
/**  
 * Set up the sync adapter. This form of the  
 * constructor maintains compatibility with Android 3.0  
 * and later platform versions  
 */  
public SyncAdapter(  
    Context context,  
    boolean autoInitialize,  
    boolean allowParallelSyncs) {  
    super(context, autoInitialize, allowParallelSyncs);  
    /*  
     * If your app uses a content resolver, get an instance of it  
     * from the incoming Context  
     */  
    mContentResolver = context.getContentResolver();  
    ...  
}
```

## 在onPerformSync()中添加數據傳輸代碼

Sync Adapter組件並不會自動地執行數據傳輸。它對你的數據傳輸代碼進行封裝，使得Sync Adapter框架可以在後臺執行數據傳輸，而不會牽連到你的應用。當框架準備同步你的應用數據時，它會調用你所實現的[onPerformSync\(\)](#)方法。

為了便於將數據從你的應用程序轉移到Sync Adapter組件中，Sync Adapter框架調用[onPerformSync\(\)](#)，它具有下面的參數：

### Account

該[Account](#)對象與觸發Sync Adapter的事件相關聯。如果服務端不需要使用賬戶，那麼你不需要使用這個對象內的信息。

### Extras

一個[Bundle](#)對象，它包含了一些標識，這些標識由觸發Sync Adapter的事件所發送。

### Authority

系統中某個Content Provider的Authority。你的應用必須要有訪問它的權限。通常，該Authority對應於你的應用的Content Provider。

### Content provider client

[ContentProviderClient](#)針對於由 `Authority` 參數所指向的Content Provider。[ContentProviderClient](#)是一個Content Provider的輕量級共有接口。它的基本功能和[ContentResolver](#)一樣。如果你正在使用Content Provider來存儲你的應用數據，你可以利用它連接Content Provider。反之，則將其忽略。

### Sync result

一個SyncResult對象，你可以使用它將信息發送給Sync Adapter框架。

下面的代碼片段展示了onPerformSync()函數的整體結構：

```
/*
 * Specify the code you want to run in the sync adapter. The entire
 * sync adapter runs in a background thread, so you don't have to set
 * up your own background processing.
 */
@Override
public void onPerformSync(
    Account account,
    Bundle extras,
    String authority,
    ContentProviderClient provider,
    SyncResult syncResult) {
    /*
     * Put the data transfer code here.
     */
    ...
}
```

雖然實際的onPerformSync()實現是要根據應用數據的同步需求以及服務器的連接協議來制定，但是你的實現應當包含下列這些基本任務：

#### 連接到一個服務器

儘管你可以假定在你開始傳輸數據時，已經獲取到了網絡連接，但是Sync Adapter框架並不會自動地連接到一個服務器。

#### 下載和上傳數據

Sync Adapter不會自動執行數據傳輸。如果你想要從服務器下載數據並將它存儲到Content Provider中，你必須提供請求數據，下載數據和將數據插入到Provider中的代碼。類似地，如果你想把數據發送到服務器，你需要從一個文件，數據庫或者Provider中讀取數據，並且發送必需的上傳請求。同時你還需要處理在你執行數據傳輸時所發生的網絡錯誤。

#### 處理數據衝突或者確定當前數據是否最新

Sync Adapter不會自動地解決服務器數據與設備數據之間的衝突。同時，它也不會自動檢測服務器上的數據是否比設備上的數據要新，反之亦然。因此，你必須自己提供處理這些狀況的算法。

#### 清理

在數據傳輸的尾聲，記得要關閉網絡連接，清除臨時文件和緩存。

Note : Sync Adapter框架會在一個後臺線程中執行onPerformSync()方法，所以你不需要配置後臺處理任務。

除了和同步相關的任務之外，你應該嘗試將一些週期性的網絡相關的任務合並起來，並將它們添加到onPerformSync()中。將所有網絡任務集中到該方法內處理，可以節省由啓動和停止網絡接口所造成的電量損失。有關更多如何在進行網絡訪問時更高效地使用電池方面的知識，可以閱讀：[Transferring Data Without Draining the Battery](#)，它描述了一些在數據傳輸代碼中可以包含的網絡訪問任務。

## 將Sync Adapter綁定到框架上

現在，你已經將你的數據傳輸代碼封裝在了Sync Adapter組建中，但是你必須讓框架可以訪問你的代碼。為了做到這一點，你需要創建一個綁定Service，它將一個特殊的Android Binder對象從Sync Adapter組件傳遞給框架。有了這一

Binder對象，框架就可以調用[onPerformSync\(\)](#)方法並將數據傳遞給它。

在服務的[onCreate\(\)](#)方法中將你的Sync Adapter組件實例化為一個單例。通過在[onCreate\(\)](#)方法中實例化該組件，你可以推遲到服務啓動後再創建它，這會在框架第一次嘗試執行你的數據傳輸時發生。你需要通過一種線程安全的方法來實例化組件，以防止Sync Adapter框架在響應觸發和調度時，形成含有多個Sync Adapter執行的隊列。

作為例子，下面的代碼片段展示了你應該如何實現一個綁定Service的類，實例化你的Sync Adapter組件，並獲取Android Binder對象：

```
package com.example.android.syncadapter;
/**
 * Define a Service that returns an IBinder for the
 * sync adapter class, allowing the sync adapter framework to call
 * onPerformSync().
 */
public class SyncService extends Service {
    // Storage for an instance of the sync adapter
    private static SyncAdapter sSyncAdapter = null;
    // Object to use as a thread-safe lock
    private static final Object sSyncAdapterLock = new Object();
    /*
     * Instantiate the sync adapter object.
     */
    @Override
    public void onCreate() {
        /*
         * Create the sync adapter as a singleton.
         * Set the sync adapter as syncable
         * Disallow parallel syncs
         */
        synchronized (sSyncAdapterLock) {
            if (sSyncAdapter == null) {
                sSyncAdapter = new SyncAdapter(getApplicationContext(), true);
            }
        }
    }
    /**
     * Return an object that allows the system to invoke
     * the sync adapter.
     *
     */
    @Override
    public IBinder onBind(Intent intent) {
        /*
         * Get the object that allows external processes
         * to call onPerformSync(). The object is created
         * in the base class code when the SyncAdapter
         * constructors call super()
         */
        return sSyncAdapter.getSyncAdapterBinder();
    }
}
```

Note：要看更多Sync Adapter綁定服務的例子，可以閱讀樣例代碼。

## 添加框架所需的賬戶

Sync Adapter框架需要每個Sync Adapter擁有一個賬戶類型。在[創建Stub授權器](#)章節中，你已經聲明了賬戶類型的值。現在你需要在Android系統中配置該賬戶類型。要配置賬戶類型，通過調用[addAccountExplicitly\(\)](#)添加一個使用其賬戶類型的虛擬賬戶。

調用該方法最合適的地方是在應用的啓動Activity的[onCreate\(\)](#)方法中。如下面的代碼樣例所示：

```
public class MainActivity extends FragmentActivity {
    ...
    ...
    // Constants
    // The authority for the sync adapter's content provider
    public static final String AUTHORITY = "com.example.android.datasync.provider"
    // An account type, in the form of a domain name
    public static final String ACCOUNT_TYPE = "example.com";
    // The account name
    public static final String ACCOUNT = "dummyaccount";
    // Instance fields
    Account mAccount;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Create the dummy account
        mAccount = CreateSyncAccount(this);
        ...
    }
    ...
    /**
     * Create a new dummy account for the sync adapter
     *
     * @param context The application context
     */
    public static Account CreateSyncAccount(Context context) {
        // Create the account type and default account
        Account newAccount = new Account(
            ACCOUNT, ACCOUNT_TYPE);
        // Get an instance of the Android account manager
        AccountManager accountManager =
            (AccountManager) context.getSystemService(
                ACCOUNT_SERVICE);
        /*
         * Add the account and account type, no password or user data
         * If successful, return the Account object, otherwise report an error.
         */
        if (accountManager.addAccountExplicitly(newAccount, null, null)) {
            /*
             * If you don't set android:syncable="true" in
             * in your <provider> element in the manifest,
             * then call context.setIsSyncable(account, AUTHORITY, 1)
             * here.
             */
        } else {
            /*
             * The account exists or some other error occurred. Log this, report it,
             * or handle it internally.
             */
        }
    }
    ...
}
```

## 添加Sync Adapter的元數據文件

要將你的Sync Adapter組件集成到框架中，你需要向框架提供描述組件的元數據，以及額外的標識信息。元數據指定了你為Sync Adapter所創建的賬戶類型，聲明瞭一個和你的應用相關聯的Content Provider Authority，對和Sync Adapter相關的一部分系統用戶接口進行控制，同時還聲明瞭其它同步相關的標識。在你的項目中的 /res/xml/ 目錄下的一個特定的文件內聲明這一元數據，你可以為這個文件命名，不過通常來說我們將其命名為 syncadapter.xml 。

在這一文件中包含了一個XML標籤 `<sync-adapter>`，它包含了下列的屬性字段：

`android:contentAuthority`

Content Provider的URI Authority。如果你在前一節課程中爲你的應用創建了一個Stub Content Provider，那麼請使用你在清單文件中添加在 `<provider>` 標籤內的 `android:authorities` 屬性值。這一屬性的更多細節在本章後續章節中有更多的介紹。

如果你正使用Sync Adapter將數據從Content Provider傳輸到服務器上，該屬性的值應該和數據的Content URI Authority保持一致。這個值也是你在清單文件中添加在 `<provider>` 標籤內的 `android:authorities` 屬性的值。

`android:accountType`

Sync Adapter框架所需要的賬戶類型。這個值必須和你所創建的驗證器元數據文件內所提供的賬戶類型一致（詳細內容可以閱讀：[創建Stub授權器](#)）。這也是在上一節的代碼片段中。常量 `ACCOUNT_TYPE` 的值。

配置相關屬性 `android:userVisible`：該屬性設置Sync Adapter框架的賬戶類型是否可見。默認地，和賬戶類型相關聯的賬戶圖標和標簽在系統設置的賬戶選項中可以看見，所以你應該將你的Sync Adapter設置爲對用戶不可見，除非你確實擁有一個賬戶類型或者域名，它們可以輕鬆地和你的應用相關聯。如果你將你的賬戶類型設置爲不可見，你仍然可以允許用戶通過一個Activity中的用戶接口來控制你的Sync Adapter。`android:supportsUploading`：允許你將數據上傳到雲。如果你的應用僅僅下載數據，那麼請將該屬性設置爲 `false`。`android:allowParallelSyncs`：允許多個Sync Adapter組件的實例同時運行。如果你的應用支持多個用戶賬戶並且你希望多個用戶並行地傳輸數據，那麼你可以使用該特性。如果你從不執行多個數據傳輸，這個選項是沒用的。`android:isAlwaysSyncable`：指明Sync Adapter框架可以在任何你指定的時間運行你的Sync Adapter。如果你希望通過代碼來控制Sync Adapter的運行時機，請將該屬性設置爲 `false`，然後調用[requestSync\(\)](#)來運行Sync Adapter。要學習更多關於運行Sync Adapter的知識，可以閱讀：[執行Sync Adapter](#)。

下面的代碼展示了應該如何通過XML配置一個使用單個虛擬賬戶，並且只執行下載的Sync Adapter：

```
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:contentAuthority="com.example.android.datasync.provider"
    android:accountType="com.android.example.datasync"
    android:userVisible="false"
    android:supportsUploading="false"
    android:allowParallelSyncs="false"
    android:isAlwaysSyncable="true"/>
```

## 在Manifest清單文件中聲明Sync Adapter

一旦你將Sync Adapter組件集成到了你的應用中，你需要聲明相關的權限來使用它，並且你還需要聲明你所添加的綁定Service。

由於Sync Adapter組件會運行設備與網絡之間傳輸數據的代碼，所以你需要請求使用網絡的權限。同時，你的應用還需要讀寫Sync Adapter配置信息的權限，這樣你才能通過應用中的其它組件去控制Sync Adapter。另外，你還需要一個特殊的權限，來允許你的應用使用你在[創建Stub授權器](#)中所創建的授權器組件。

要請求這些權限，將下列內容添加到應用清單文件中，並作爲 `<manifest>` 標籤的子標籤：

`android.permission.INTERNET`

允許Sync Adapter訪問網絡，使得它可以從設備下載和上傳數據到服務器。如果之前已經請求了該權限，那麼你就不需要重複請求了。

`android.permission.READ_SYNC_SETTINGS`

允許你的應用讀取當前的Sync Adapter配置。例如，你需要該權限來調用[getIsSyncable\(\)](#)。

#### android.permission.WRITE\_SYNC\_SETTINGS

允許你的應用對Sync Adapter的配置進行控制。你需要這一權限來通過[addPeriodicSync\(\)](#)方法設置執行同步的時間間隔。另外，調用[requestSync\(\)](#)方法不需要用到該權限。更多信息可以閱讀：[執行Sync Adapter](#)。

#### android.permission.AUTHENTICATE\_ACCOUNTS

允許你使用在[創建Stub授權器](#)中所創建的驗證器組件。

下面的代碼片段展示瞭如何添加這些權限：

```
<manifest>
...
<uses-permission
    android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.READ_SYNC_SETTINGS"/>
<uses-permission
    android:name="android.permission.WRITE_SYNC_SETTINGS"/>
<uses-permission
    android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
...
</manifest>
```

最後，要聲明框架用來和你的Sync Adapter進行交互的綁定Service，添加下列的XML代碼到應用清單文件中，作為[`<application>`](#)標籤的子標籤：

```
<service
    android:name="com.example.android.datasync.SyncService"
    android:exported="true"
    android:process=":sync">
    <intent-filter>
        <action android:name="android.content.SyncAdapter"/>
    </intent-filter>
    <meta-data android:name="android.content.SyncAdapter"
        android:resource="@xml/syncadapter" />
</service>
```

`<intent-filter>` 標籤配置了一個過濾器，它會被帶有 `android.content.SyncAdapter` 這一Action的Intent所觸發，該Intent一般是由系統為了運行Sync Adapter而發出的。當過濾器被觸發後，系統會啓動你所創建的綁定服務，在本例中它叫做 SyncService 。屬性`android:exported="true"`允許你應用之外的其它進程（包括系統）訪問這一Service。屬性`android:process=":sync"`告訴系統應該在一個全局共享的，且名字叫做 sync 的進程內運行該Service。如果你的應用中有多个Sync Adapter，那麼它們可以共享該進程，這有助於減少開銷。

`<meta-data>` 標籤提供了你之前為Sync Adapter所創建的元數據XML文件的文件名。屬性`android:name`指出這一元數據是針對Sync Adapter框架的。而`android:resource`標籤則指定了元數據文件的名稱。

現在你已經為Sync Adapter準備好所有相關的組件了。下一節課將講授如何讓Sync Adapter框架運行你的Sync Adapter，要實現這一點，既可以通過響應一個事件的方式，也可以通過執行一個週期性任務的方式。

# 執行Sync Adapter

編寫:jdneo - 原文:<http://developer.android.com/training/sync-adapters/running-sync-adapter.html>

在本節課之前，你已經學習瞭如何創建一個封裝了數據傳輸代碼的Sync Adapter組件，以及如何添加其它的組件，使得你可以將Sync Adapter集成到系統當中。現在你已經擁有了所有部件，來安裝一個包含有Sync Adapter的應用，但是這裏還沒有任何代碼是負責去運行Sync Adapter的。

執行Sync Adapter的時機，一般應該基於某個計劃任務或者一些事件的間接結果。例如，你可能希望你的Sync Adapter以一個定期計劃任務的形式運行（比如每隔一段時間或者在每天的一個固定時間運行）。或者你也可能希望當設備上的數據發生變化後，執行你的Sync Adapter。你應該避免將運行Sync Adapter作為用戶某個行為的直接結果，因為這樣做的話你就無法利用Sync Adapter框架可以按計劃調度的特性。例如，你應該在UI中避免使用刷新按鈕。

下列情況可以作為運行Sync Adapter的時機：

當服務端數據變更時：

當服務端發送消息告知服務端數據發生變化時，運行Sync Adapter以響應這一來自服務端的消息。這一選項允許從服務器更新數據到設備上，該方法可以避免由於輪詢服務器所造成的執行效率下降，或者電量損耗。

當設備的數據變更時：

當設備上的數據發生變化時，運行Sync Adapter。這一選項允許你將修改後的數據從設備發送給服務器，如果你需要保證服務器端一直擁有設備上最新的數據，那麼這一選項非常有用。如果你將數據存儲於你的Content Provider，那麼這一選項的實現將會非常直接。如果你使用的是一個Stub Content Provider，檢測數據的變化可能會比較困難。

當系統發送了一個網絡消息：

當Android系統發送了一個網絡消息來保持TCP/IP連接開啓時，運行Sync Adapter。這個消息是網絡框架（Networking Framework）的一個基本部分。可以將這一選項作為自動運行Sync Adapter的一個方法。另外還可以考慮將它和基於時間間隔運行Sync Adapter的策略結合起來使用。

每隔固定的時間間隔後：

可以每隔一段你指定的時間間隔後，運行Sync Adapter，或者在每天的固定時間運行它。

根據需求：

運行Sync Adapter以響應用戶的行為。然而，為了提供最佳的用戶體驗，你應該主要依賴那些更加自動式的選項。使用自動式的選項，你可以節省大量的電量以及網絡資源。

本課程的後續部分會詳細介紹每個選項。

## 當服務器數據變化時，運行Sync Adapter

如果你的應用從服務器傳輸數據，且服務器的數據會頻繁地發生變化，你可以使用一個Sync Adapter通過下載數據來響應服務端數據的改變。要運行Sync Adapter，我們需要讓服務端響應用的BroadcastReceiver發送一條特殊的消息。為了響應這條消息，可以調用ContentResolver.requestSync()方法，向Sync Adapter框架發出信號，讓它運行你的Sync Adapter。

谷歌雲消息（[Google Cloud Messaging](#)，GCM）提供了你需要的服務端組件和設備端組件，來讓上述消息系統能夠運行。使用GCM觸發數據傳輸比通過向服務器輪詢的方式要更加可靠，也更加有效。因為輪詢需要一個一直處於活躍狀態的Service，而GCM使用的BroadcastReceiver僅在消息到達時會被激活。另外，即使沒有更新的內容，定期的輪詢也會消耗大量的電池電量，而GCM僅在需要時纔會發出消息。

Note：如果你使用GCM，將廣播消息發送到所有安裝了你的應用的設備，來激活你的Sync Adapter，要記住他們會在同一時間（粗略地）收到你的消息。這會導致在同一時段內有多個Sync Adapter的實例在運行，進而導致服務器和網絡的負載過重。要避免這一情況，你應該考慮為不同的設備設定不同的Sync Adapter延遲啓動時間。

下面的代碼展示瞭如何通過[requestSync\(\)](#)響應一個接收到的GCM消息：

```
public class GcmBroadcastReceiver extends BroadcastReceiver {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.datasync.provider"
    // Account type
    public static final String ACCOUNT_TYPE = "com.example.android.datasync";
    // Account
    public static final String ACCOUNT = "default_account";
    // Incoming Intent key for extended data
    public static final String KEY_SYNC_REQUEST =
        "com.example.android.datasync.KEY_SYNC_REQUEST";
    ...
    @Override
    public void onReceive(Context context, Intent intent) {
        // Get a GCM object instance
        GoogleCloudMessaging gcm =
            GoogleCloudMessaging.getInstance(context);
        // Get the type of GCM message
        String messageType = gcm.getMessageType(intent);
        /*
         * Test the message type and examine the message contents.
         * Since GCM is a general-purpose messaging system, you
         * may receive normal messages that don't require a sync
         * adapter run.
         * The following code tests for a boolean flag indicating
         * that the message is requesting a transfer from the device.
         */
        if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType)
            && intent.getBooleanExtra(KEY_SYNC_REQUEST)) {
            /*
             * Signal the framework to run your sync adapter. Assume that
             * app initialization has already created the account.
             */
            ContentResolver.requestSync(ACCOUNT, AUTHORITY, null);
            ...
        }
        ...
    }
}
```

## 當Content Provider的數據變化時，運行Sync Adapter

如果你的應用在一個Content Provider中收集數據，並且你希望當你更新了Content Provider的時候，同時更新服務器的數據，你可以配置你的Sync Adapter來讓它自動運行。要做到這一點，你首先應該為Content Provider註冊一個Observer。當Content Provider的數據發生了變化之後，Content Provider框架會調用Observer。在Observer中，調用[requestSync\(\)](#)來告訴框架現在應該運行你的Sync Adapter了。

Note：如果你使用的是一個Stub Content Provider，那麼你不會在Content Provider中有任何數據，並

且`onChange()`方法也從來不會被調用。在這種情況下，你不得不提供自己的某種機制來檢測設備數據的變化。這一機制還要負責在數據發生變化時調用`requestSync()`。

為了給你的Content Provider創建一個Observer，繼承`ContentObserver`類，並且實現`onChange()`方法的兩種形式。在`onChange()`中，調用`requestSync()`來啓動Sync Adapter。

要註冊Observer，需要將它作為參數傳遞給`registerContentObserver()`。在該方法中，你還要傳遞一個你想要監視的Content URI。Content Provider框架會將這個需要監視的URI與其它一些Content URLs進行比較，這些其它的Content URLs來自於`ContentResolver`中那些可以修改Provider的方法（如`ContentResolver.insert()`）所傳入的參數，如果出現了變化，那麼你所實現的`ContentObserver.onChange()`將會被調用。

下面的代碼片段展示瞭如何定義一個`ContentObserver`，它在表數據發生變化後調用`requestSync()`：

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider scheme
    public static final String SCHEME = "content://";
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.datasync.provider";
    // Path for the content provider table
    public static final String TABLE_PATH = "data_table";
    // Account
    public static final String ACCOUNT = "default_account";
    // Global variables
    // A content URI for the content provider's data table
    Uri mUri;
    // A content resolver for accessing the provider
    ContentResolver mResolver;
    ...
    public class TableObserver extends ContentObserver {
        /*
         * Define a method that's called when data in the
         * observed content provider changes.
         * This method signature is provided for compatibility with
         * older platforms.
         */
        @Override
        public void onChange(boolean selfChange) {
            /*
             * Invoke the method signature available as of
             * Android platform version 4.1, with a null URI.
             */
            onChange(selfChange, null);
        }
        /*
         * Define a method that's called when data in the
         * observed content provider changes.
         */
        @Override
        public void onChange(boolean selfChange, Uri changeUri) {
            /*
             * Ask the framework to run your sync adapter.
             * To maintain backward compatibility, assume that
             * changeUri is null.
             */
            ContentResolver.requestSync(ACCOUNT, AUTHORITY, null);
        }
        ...
    }
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Get the content resolver object for your app
        mResolver = getContentResolver();
        // Construct a URI that points to the content provider data table
        mUri = new Uri.Builder()
```

```
.scheme(SCHEME)
.authority(AUTHORITY)
.path(TABLE_PATH)
.build();

/*
 * Create a content observer object.
 * Its code does not mutate the provider, so set
 * selfChange to "false"
 */
TableObserver observer = new TableObserver(false);
/*
 * Register the observer for the data table. The table's path
 * and any of its subpaths trigger the observer.
 */
mResolver.registerContentObserver(mUri, true, observer);
...
}

...
```

## 在一個網絡消息之後，運行Sync Adapter

當可以獲得一個網絡連接時，Android系統會每隔幾秒發送一條消息來保持TCP/IP連接處於開啓狀態。這一消息也會傳遞到每個應用的ContentResolver中。通過調用[setSyncAutomatically\(\)](#)，你可以在ContentResolver收到消息後，運行Sync Adapter。

每當網絡消息被發送後運行你的Sync Adapter，通過這樣的調度方式可以保證每次運行Sync Adapter時都可以訪問網絡。如果不是每次數據變化時就要以數據傳輸來響應，但是又希望自己的數據會被定期地更新，那麼你可以用這一選項。類似地，如果你不想要定期執行你的Sync Adapter，但你希望經常運行它，你也可以使用這一選項。

由於[setSyncAutomatically\(\)](#)方法不會禁用[addPeriodicSync\(\)](#)，所以你的Sync Adapter可能會在一小段時間內重複地被觸發激活。如果你想要定期地運行你的Sync Adapter，應該禁用[setSyncAutomatically\(\)](#)。

下面的代碼片段向你展示如何配置你的ContentResolver，利用它來響應網絡消息，從而運行你的Sync Adapter，：

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.datasync.provider";
    // Account
    public static final String ACCOUNT = "default_account";
    // Global variables
    // A content resolver for accessing the provider
    ContentResolver mResolver;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Get the content resolver for your app
        mResolver = getContentResolver();
        // Turn on automatic syncing for the default account and authority
        mResolver.setSyncAutomatically(ACCOUNT, AUTHORITY, true);
        ...
    }
    ...
}
```

## 定期地運行Sync Adapter

你可以設置一個每次運行之間的時間間隔來定期運行你的Sync Adapter，或者在每天的固定時間運行它，還可以兩種策略同時使用。定期地運行你的Sync Adapter可以讓你與服務器的更新間隔大致保持一致。

同樣地，當你的服務器相對來說比較空閒時，你可以通過在夜間定期調用Sync Adapter，把設備上的數據上傳到服務器。大多數用戶在晚上不會關機，併為手機充電，所以這一方法是可行的。而且，通常來說，設備不會在深夜運行除了你的Sync Adapter之外的其他的任務。然而，如果你使用這個方法的話，你需要注意讓每臺設備在略微不同的時間觸發數據傳輸。如果所有設備在同一時間運行你的Sync Adapter，那麼你的服務器和移動運營商的網絡將很有可能負載過重。

一般來說，當你的用戶不需要實時更新，而希望定期更新時，使用定期運行的策略會很有用。如果你希望在數據的實時性和Sync Adapter的資源消耗之間進行一個平衡，那麼定期執行是一個不錯的選擇。

要定期運行你的Sync Adapter，調用[addPeriodicSync\(\)](#)。這樣每隔一段時間，Sync Adapter就會運行。由於Sync Adapter框架會考慮其他Sync Adapter的執行，並嘗試最大化電池效率，所以間隔時間會動態地進行細微調整。同時，如果當前無法獲得網絡連接，框架不會運行你的Sync Adapter。

注意，[addPeriodicSync\(\)](#)方法不會讓Sync Adapter每天在某個時間自動運行。要讓你的Sync Adapter在每天的某個時刻左右自動執行，可以使用一個重複計時器作為觸發器。重複計時器的更多細節可以閱讀：[AlarmManager](#)。如果你使用[setInexactRepeating\(\)](#)方法設置了一個每天的觸發時刻會有粗略變化的觸發器，你仍然應該將不同設備的Sync Adapter的運行時間隨機化，使得它們的執行交錯開來。

[addPeriodicSync\(\)](#)方法不會禁用[setSyncAutomatically\(\)](#)，所以你可能會在一小段時間內產生多個Sync Adapter的運行實例。另外，僅有一部分Sync Adapter的控制標識可以在調用[addPeriodicSync\(\)](#)時使用。不被允許的標識在該方法的[文檔](#)中可以查看。

下面的代碼樣例展示瞭如何定期執行Sync Adapter：

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.datasync.provider";
    // Account
    public static final String ACCOUNT = "default_account";
    // Sync interval constants
    public static final long SECONDS_PER_MINUTE = 60L;
    public static final long SYNC_INTERVAL_IN_MINUTES = 60L;
    public static final long SYNC_INTERVAL =
        SYNC_INTERVAL_IN_MINUTES *
        SECONDS_PER_MINUTE;
    // Global variables
    // A content resolver for accessing the provider
    ContentResolver mResolver;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Get the content resolver for your app
        mResolver = getContentResolver();
        /*
         * Turn on periodic syncing
         */
        ContentResolver.addPeriodicSync(
            ACCOUNT,
            AUTHORITY,
            Bundle.EMPTY,
            SYNC_INTERVAL);
        ...
    }
    ...
}
```

# 按需求執行Sync Adapter

以響應用戶請求的方式運行Sync Adapter是最不推薦的策略。要知道，該框架是被特別設計的，它可以讓Sync Adapter在根據某個調度規則運行時，能夠儘量最高效地使用手機電量。顯然，在數據改變的時候執行同步可以更有效的使用手機電量，因為電量都消耗在了更新新的數據上。

相比之下，允許用戶按照自己的需求運行Sync Adapter意味着Sync Adapter會自己運行，這將無法有效地使用電量和網絡資源。如果根據需求執行同步，會誘導用戶即便沒有證據表明數據發生了變化也請求一個更新，這些無用的更新會導致對電量的低效率使用。一般來說，你的應用應該使用其它信號來觸發一個同步更新或者讓它們定期地去執行，而不是依賴於用戶的輸入。

不過，如果你仍然想要按照需求運行Sync Adapter，可以將Sync Adapter的配置標識設置為手動執行，之後調用[ContentResolver.requestSync\(\)](#)來觸發一次更新。

通過下列標識來執行按需求的數據傳輸：

## [SYNC\\_EXTRAS\\_MANUAL](#)

強制執行手動的同步更新。Sync Adapter框架會忽略當前的設置，比如通過[setSyncAutomatically\(\)](#)方法設置的標識。

## [SYNC\\_EXTRAS\\_EXPEDITED](#)

強制同步立即執行。如果你不設置此項，系統可能會在運行同步請求之前等待一小段時間，因為它會嘗試將一小段時間內的多個請求集中在一起調度，目的是為了優化電量的使用。

下面的代碼片段將向你展示如何調用[requestSync\(\)](#)來響應一個按鈕點擊事件：

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY =
        "com.example.android.datasync.provider"
    // Account type
    public static final String ACCOUNT_TYPE = "com.example.android.datasync";
    // Account
    public static final String ACCOUNT = "default_account";
    // Instance fields
    Account mAccount;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        /*
         * Create the dummy account. The code for CreateSyncAccount
         * is listed in the lesson Creating a Sync Adapter
         */
        mAccount = CreateSyncAccount(this);
        ...
    }
    /**
     * Respond to a button click by calling requestSync(). This is an
     * asynchronous operation.
     *
     * This method is attached to the refresh button in the layout
     * XML file
     *
     * @param v The View associated with the method call,
    
```

```
* in this case a Button
*/
public void onRefreshButtonClick(View v) {
    ...
    // Pass the settings flags by inserting them in a bundle
    Bundle settingsBundle = new Bundle();
    settingsBundle.putBoolean(
        ContentResolver.SYNC_EXTRAS_MANUAL, true);
    settingsBundle.putBoolean(
        ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
    /*
     * Request the sync for the default account, authority, and
     * manual sync settings
     */
    ContentResolver.requestSync(mAccount, AUTHORITY, settingsBundle);
}
```

# 使用Volley傳輸網絡數據

編寫:kesenhoo - 原文:<http://developer.android.com/training/volley/index.html>

Volley 是一個HTTP庫，它能夠幫助Android apps更方便的執行網絡操作，最重要的是，它更快速高效。可以通過開源的 [AOSP](#) 倉庫獲取到Volley。

## YOU SHOULD ALSO SEE

使用Volley來編寫一個app，請參考[2013 Google I/O schedule app](#). 另外需要特別關注下面2個部分：

- [ImageLoader](#)
- [BitmapCache](#)

[VIDEO - Volley:Easy,Fast Networking for Android](#)

Volley 有如下的優點：

- 自動調度網絡請求。
- 高並髮網絡連接。
- 通過標準的HTTP的cache coherence(高速緩存一致性)使得磁盤與內存緩存不可見(Transparent)。
- 支持指定請求的優先級。
- 支持取消已經發出的請求。你可以取消單個請求，或者指定取消請求隊列中的一個區域。
- 框架容易被定製，例如，定製重試或者回退功能。
- 強大的指令(Strong ordering)可以使得異步加載網絡數據並顯示到UI的操作更加簡單。
- 包含了Debugging與tracing工具。

Volley擅長執行用來顯示UI的RPC操作， 例如獲取搜索結果的數據。它輕鬆的整合了任何協議，並輸出操作結果的數據，可以是raw strings，也可以是images，或者是JSON。通過提供內置你可能使用到得功能，Volley可以使得你免去重複編寫樣板代碼，使你可以把關注點放在你的app的功能邏輯上。

Volley不適合用來下載大的數據文件。因爲Volley會在解析的過程中保留持有所有的響應數據在內存中。對於下載大量的數據操作，請考慮使用[DownloadManager](#)。

Volley框架的核心代碼是託管在AOSP倉庫的 `frameworks/volley` 中，相關的工具放在 `toolbox` 下。把Volley添加到你的項目中的最簡便的方法是Clone倉庫然後把它設置爲一個library project：

- 通過下面的命令來Clone倉庫：

```
git clone https://android.googlesource.com/platform/frameworks/volley
```

- 以一個Android library project的方式導入下載的源代碼到你的項目中。(如果你是使用Eclipse，請參考[Managing Projects from Eclipse with ADT](#))，或者編譯成一個 `.jar` 文件。

## Lessons

- [發送一個簡單的網絡請求\(Sending a Simple Request\)](#)

學習如何通過Volley默認的行爲發送一個簡單的請求，以及如何取消一個請求。

- [建立一個請求隊列\(Setting Up a RequestQueue\)](#)

學習如何建立一個請求隊列，以及如何實現一個單例模式來創建一個請求隊列，使RequestQueue能夠持續保持在你的app的生命週期中。

- [生成一個標準的請求\(Making a Standard Request\)](#)

學習如何使用Volley的out-of-the-box（可直接使用、無需配置）的請求類型(raw strings, images, and JSON)來發送一個請求。

- [實現自定義的請求\(Implementing a Custom Request\)](#)

學習如何實現一個自定義的請求

# 發送簡單的網絡請求(Sending a Simple Request)

編寫:kesenhoo - 原文:<http://developer.android.com/training/volley/simple.html>

使用Volley的方式是，你通過創建一個 `RequestQueue` 並傳遞 `Request` 對象給它。`RequestQueue`管理用來執行網絡操作的工作線程，從Cache中讀寫數據，並解析Http的響應內容。`Requests` 執行raw responses的解析，Volley會把響應的數據分發給主線程。

這節課會介紹如何使用 `Volley.newRequestQueue` 這個建立請求隊列`RequestQueue`的方法來發送一個請求，在下一節課[建立一個請求隊列Setting Up a RequestQueue](#)中會介紹你自己如何建立一個請求隊列。

這節課也會介紹如何添加一個請求到`RequestQueue`以及如何取消一個請求。

## 1)Add the INTERNET Permission

為了使用Volley，你必須添加 `android.permission.INTERNET` 權限到你的manifest文件中。沒有這個權限，你的app將無法訪問網絡。

## 2)Use newRequestQueue

Volley提供了一個簡便的方法：`Volley.newRequestQueue` 用來為你建立一個 `RequestQueue`，使用默認值，並啓動這個隊列。例如：

```
final TextView mTextView = (TextView) findViewById(R.id.text);
...
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string.
        mTextView.setText("Response is: "+ response.substring(0,500));
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        mTextView.setText("That didn't work!");
    }
});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Volley總是將解析後的數據返回至主線程中。在主線程中更加合適使用接收到的數據用來操作UI控件，這樣你可以在響應的handler中輕鬆的修改UI，但是對於庫提供的一些其他方法是有些特殊的，例如與取消有關的。

關於如何創建你自己的請求隊列，而不是使用`Volley.newRequestQueue`方法，請查看[建立一個請求隊列Setting Up a RequestQueue](#)。

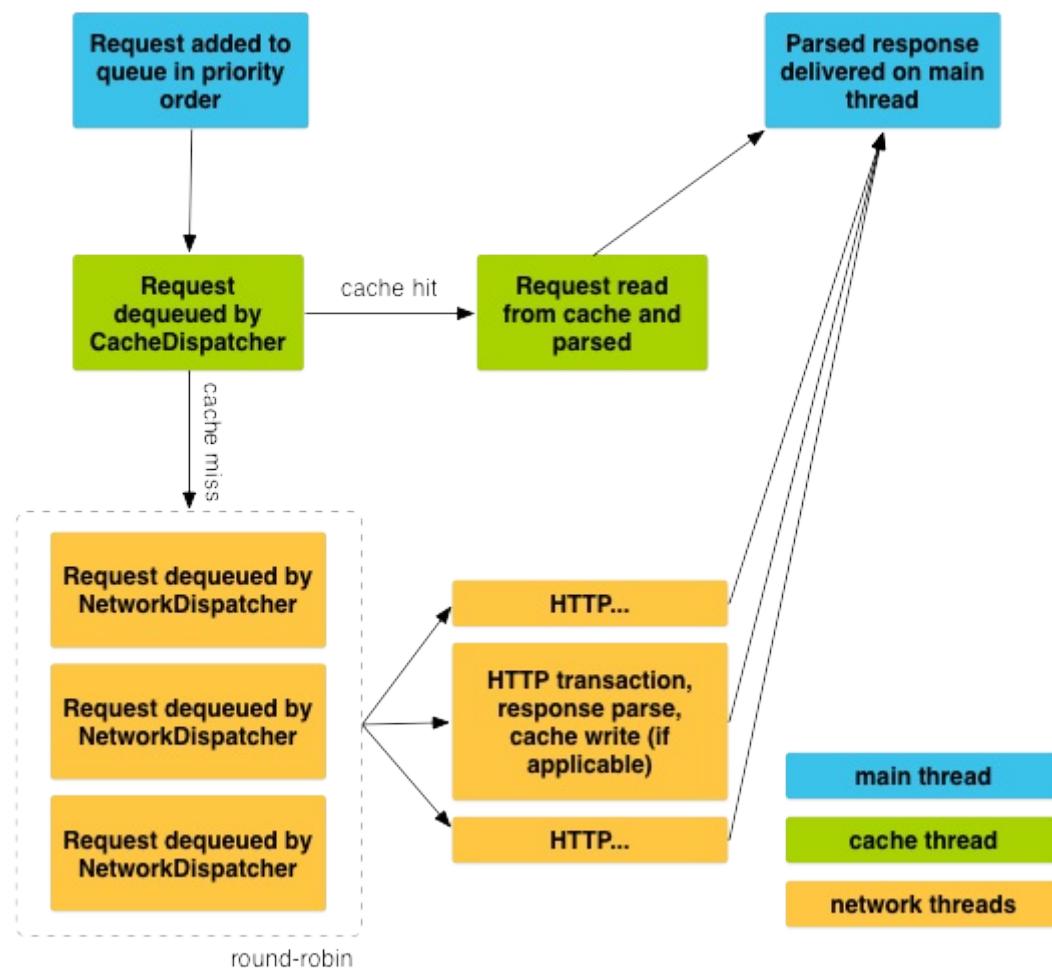
## 3)Send a Request

為了發送一個請求，你只需要構造一個請求並通過 `add()` 方法添加到 `RequestQueue` 中。一旦你添加了這個請求，它會通過隊列，得到處理，然後得到原始的響應數據並返回。

當你執行 `add()` 方法時，Volley觸發執行一個緩存處理線程以及一系列網絡處理線程。當你添加一個請求到隊列中，它將被緩存線程所捕獲並觸發：如果這個請求可以被緩存處理，那麼會在緩存線程中執行響應數據的解析並返回到主線程。如果請求不能被緩存所處理，它會被放到網絡隊列中。網絡線程池中的第一個可用的網絡線程會從隊列中獲取到這個請求並執行HTTP操作，解析工作線程的響應數據，把數據寫到緩存中並把解析之後的數據返回到主線程。

請注意那些比較耗時的操作，例如I/O與解析parsing/decoding都是執行在工作線程。你可以在任何線程中添加一個請求，但是響應結果都是返回到主線程的。

下圖1，演示了一個請求的生命週期：



## 4)Cancel a Request

對請求Request對象調用 `cancel()` 方法取消一個請求。一旦取消，Volley會確保你的響應Handler不會被執行。這意味着在實際操作中你可以在[activity](#)的 `onStop()` 方法中取消所有pending在隊列中的請求。你不需要通過檢測 `getActivity() == null` 來丟棄你的響應handler，其他類似 `onSaveInstanceState()` 等保護性的方法裏面也都不需要檢測。

為了利用這種優勢，你應該跟蹤所有已經發送的請求，以便在需要的時候可以取消他們。有一個簡便的方法：你可以為每一個請求對象都綁定一個tag對象。然後你可以使用這個tag來提供取消的範圍。例如，你可以為你的所有請求都綁定

到執行的Activity上，然後你可以在 `onStop()` 方法執行 `requestQueue.cancelAll(this)`。同樣的，你可以為ViewPager中的所有請求縮略圖Request對象分別打上對應Tab的tag。並在滑動時取消這些請求，用來確保新生成的tab不會被前面tab的請求任務所卡到。

下面一個使用String來打Tag的例子：

1. 定義你的tag並添加到你的請求任務中。

```
public static final String TAG = "MyTag";
StringRequest stringRequest; // Assume this exists.
RequestQueue mRequestQueue; // Assume this exists.

// Set the tag on the request.
stringRequest.setTag(TAG);

// Add the request to the RequestQueue.
mRequestQueue.add(stringRequest);
```

1. 在activity的`onStop()`方法裏面，取消所有的包含這個tag的請求任務。

```
@Override
protected void onStop () {
    super.onStop();
    if (mRequestQueue != null) {
        mRequestQueue.cancelAll(TAG);
    }
}
```

當取消請求時請注意：如果你依賴你的響應handler來標記狀態或者觸發另外一個進程，你需要對此進行考慮。再說一次，response handler是不會被執行的。

# 建立請求隊列(Setting Up a RequestQueue)

編寫:kesenhoo - 原文:<http://developer.android.com/training/volley/requestqueue.html>

前一節課演示瞭如何使用 `Volley.newRequestQueue` 這一簡便的方法來建立一個 `RequestQueue`，這是利用了Volley默認的優勢。這節課會介紹如何顯式的建立一個`RequestQueue`，以便滿足你自定義的需求。

這節課同樣會介紹一種推薦的實現方式：創建一個單例的`RequestQueue`，這使得`RequestQueue`能夠持續保持在你的app的生命週期中。

## 1) Set Up a Network and Cache

一個`RequestQueue`需要兩部分來支持它的工作：一部分是網絡操作，用來傳輸請求，另外一個是用來處理緩存操作的Cache。在Volley的工具箱中包含了標準的實現方式：`DiskBasedCache` 提供了每個文件與對應響應數據一一映射的緩存實現。`BasicNetwork` 提供了一個網絡傳輸的實現，連接方式可以是`AndroidHttpClient` 或者是 `HttpURLConnection`。

`BasicNetwork` 是Volley默認的網絡操作實現方式。一個`BasicNetwork`必須使用HTTP Client進行初始化。這個Client通常是`AndroidHttpClient` 或者 `HttpURLConnection`:

- 對於app target API level低於API 9(Gingerbread)的使用`AndroidHttpClient`。在Gingerbread之前，`HttpURLConnection`是不可靠的。對於這個的細節，請參考[Android's HTTP Clients](#)。
- 對於API Level 9以及以上的，使用`HttpURLConnection`。

為了創建一個能夠執行在所有Android版本上的應用，你可以通過檢查系統版本選擇合適的HTTP Client。例如：

```
HttpStack stack;
...
// If the device is running a version >= Gingerbread...
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
    // ...use HttpURLConnection for stack.
} else {
    // ...use AndroidHttpClient for stack.
}
Network network = new BasicNetwork(stack);
```

下面的代碼片段演示瞭如何一步步建立一個`RequestQueue`:

```
RequestQueue mRequestQueue;

// Instantiate the cache
Cache cache = new DiskBasedCache(getCacheDir(), 1024 * 1024); // 1MB cap

// Set up the network to use HttpURLConnection as the HTTP client.
Network network = new BasicNetwork(new HurlStack());

// Instantiate the RequestQueue with the cache and network.
mRequestQueue = new RequestQueue(cache, network);

// Start the queue
mRequestQueue.start();

String url ="http://www.myurl.com";

// Formulate the request and handle the response.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
```

```

@Override
public void onResponse(String response) {
    // Do something with the response
}
},
new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        // Handle error
    }
});
// Add the request to the RequestQueue.
mRequestQueue.add(stringRequest);
...

```

如果你僅僅是想做一個單次的請求並且不想要線程池一直保留，你可以通過使用在前面一課：[發送一個簡單的請求 \(Sending a Simple Request\)](#)文章中提到 `Volley.newRequestQueue()` 方法在任何需要的時刻創建RequestQueue，然後在你的響應回調裏面執行 `stop()` 方法來停止操作。但是更通常的做法是創建一個RequestQueue並設置為一個單例。下面將演示這種做法。

## 2) Use a Singleton Pattern

如果你的程序需要持續的使用網絡，更加高效的方式應該是建立一個RequestQueue的單例，這樣它能夠持續保持在整個app的生命週期中。你可以通過多種方式來實現這個單例。推薦的方式是實現一個單例類，裏面封裝了RequestQueue對象與其他Volley的方法。另外一個方法是繼承Application類，並在 `Application.onCreate()` 方法裏面建立RequestQueue。但是這個方法是不推薦的。因為一個static的單例能夠以一種更加模塊化的方式提供同樣的功能。

一個關鍵的概念是RequestQueue必須和Application context所關聯的。而不是Activity的context。這確保了RequestQueue在你的app生命週期中一直存活，而不會因為activity的重新創建而重新創建RequestQueue。(例如，當用戶旋轉設備時)。

下面是一個單例類，提供了RequestQueue與ImageLoader的功能：

```

private static MySingleton mInstance;
private RequestQueue mRequestQueue;
private ImageLoader mImageLoader;
private static Context mContext;

private MySingleton(Context context) {
    mContext = context;
    mRequestQueue = getRequestQueue();

    mImageLoader = new ImageLoader(mRequestQueue,
        new ImageLoader.ImageCache() {
            private final LruCache<String, Bitmap> cache =
                new LruCache<String, Bitmap>(20);

            @Override
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }

            @Override
            public void putBitmap(String url, Bitmap bitmap) {
                cache.put(url, bitmap);
            }
        });
}

public static synchronized MySingleton getInstance(Context context) {
    if (mInstance == null) {
        mInstance = new MySingleton(context);
    }
}

```

```
        }
        return mInstance;
    }

    public RequestQueue getRequestQueue() {
        if (mRequestQueue == null) {
            // getApplicationContext() is key, it keeps you from leaking the
            // Activity or BroadcastReceiver if someone passes one in.
            mRequestQueue = Volley.newRequestQueue(mCtx.getApplicationContext());
        }
        return mRequestQueue;
    }

    public <T> void addToRequestQueue(Request<T> req) {
        getRequestQueue().add(req);
    }

    public ImageLoader getImageLoader() {
        return mImageLoader;
    }
}
```

下面演示了利用單例類來執行RequestQueue的操作：

```
// Get a RequestQueue
RequestQueue queue = MySingleton.getInstance(this.getApplicationContext()).
    getRequestQueue();
...

// Add a request (in this example, called stringRequest) to your RequestQueue.
MySingleton.getInstance(this).addToRequestQueue(stringRequest);
```

# 創建標準的網絡請求(Making a Standard Request)

編寫:kesenhoo - 原文:<http://developer.android.com/training/volley/request.html>

這一課會介紹如何使用Volley支持的常用請求類型：

- `StringRequest` 。指定一個URL並在相應回調中接受一個原始的raw string數據。請參考前一課的示例。
- `ImageRequest` 。指定一個URL並在相應回調中接受一個image。
- `JsonObjectRequest` 與 `JsonArrayRequest` (均為 `JsonRequest` 的子類)。指定一個URL並在相應回調中獲取到一個JSON對象或者JSON數組。

如果你需要的是上面演示的請求類型，那麼你應該不需要自己實現一個自定義的請求。這節課會演示如何使用那些標準的請求類型。關於如何實現自定義的請求，請看下一課：[實現自定義的請求](#)。

## 1) Request an Image

Volley為請求圖片提供瞭如下的類。這些類依次有着依賴關係，用來支持在不同的層級進行圖片處理：

- `ImageRequest` - 一個封裝好的，用來處理URL請求圖片並且返回一張decode好的bitmap的類。它同樣提供了一些簡便的接口方法，例如指定一個大小進行重新裁剪。它的主要好處是Volley會確保類似decode，resize等耗時的操作執行在工作線程中。
- `ImageLoader` - 一個用來處理加載與緩存從網絡上獲取到的圖片的幫助類。`ImageLoader`是管理協調大量的`ImageRequest`的類。例如，在`ListView`中需要顯示大量縮略圖的時候。`ImageLoader`為通常的Volley cache提供了更加前瞻的內存緩存，這個緩存對於防止圖片抖動非常有用。。這還使得能夠在避免阻擋或者延遲主線程的前提下在緩存中能夠被Hit到。`ImageLoader`還能夠實現響應聯合Coalescing，每一個響應回調裏面都可以設置bitmap到view上面。聯合Coalescing使得能夠同時提交多個響應，這提升了性能。
- `NetworkImageView` - 在`ImageLoader`的基礎上建立，替換`ImageView`進行使用。對於需要對`ImageView`設置網絡圖片的情況下使用很有效。`NetworkImageView`同樣可以在view被detached的時候取消pending的請求。

### 1.1) Use ImageRequest

下面是一個使用`ImageRequest`的示例。它會獲取指定URL的image並顯示到app上。裏面演示的`RequestQueue`是通過上一課提到的單例類實現的。

```
ImageView mImageView;
String url = "http://i.imgur.com/7spzG.png";
mImageView = (ImageView) findViewById(R.id.myImage);
...

// Retrieves an image specified by the URL, displays it in the UI.
ImageRequest request = new ImageRequest(url,
    new Response.Listener() {
        @Override
        public void onResponse(Bitmap bitmap) {
            mImageView.setImageBitmap(bitmap);
        }
    },
    0, 0, null,
    new Response.ErrorListener() {
        public void onErrorResponse(VolleyError error) {
            mImageView.setImageResource(R.drawable.image_load_error);
        }
});
});
```

```
// Access the RequestQueue through your singleton class.  
MySingleton.getInstance(this).addToRequestQueue(request);
```

## 1.2)Use ImageLoader and NetworkImageView

你可以使用ImageLoader與NetworkImageView用來處理類似ListView等大量顯示圖片的情況。在你的layout XML文件中，你可以使用NetworkImageView來替代通常的ImageView，例如：

```
<com.android.volley.toolbox.NetworkImageView  
    android:id="@+id/networkImageView"  
    android:layout_width="150dp"  
    android:layout_height="170dp"  
    android:layout_centerHorizontal="true" />
```

你可以使用ImageLoader來顯示一張圖片，例如：

```
ImageLoader mImageLoader;  
ImageView mImageView;  
// The URL for the image that is being loaded.  
private static final String IMAGE_URL =  
    "http://developer.android.com/images/training/system-ui.png";  
...  
mImageView = (ImageView) findViewById(R.id.regularImageView);  
  
// Get the ImageLoader through your singleton class.  
mImageLoader = MySingleton.getInstance(this).getImageLoader();  
mImageLoader.get(IMAGE_URL, ImageLoader.getImageListener(mImageView,  
    R.drawable.def_image, R.drawable.err_image));
```

然而，如果你要做得是為ImageView進行圖片設置，你可以使用NetworkImageView來實現，例如：

```
ImageLoader mImageLoader;  
NetworkImageView mNetworkImageView;  
private static final String IMAGE_URL =  
    "http://developer.android.com/images/training/system-ui.png";  
...  
  
// Get the NetworkImageView that will display the image.  
mNetworkImageView = (NetworkImageView) findViewById(R.id.networkImageView);  
  
// Get the ImageLoader through your singleton class.  
mImageLoader = MySingleton.getInstance(this).getImageLoader();  
  
// Set the URL of the image that should be loaded into this view, and  
// specify the ImageLoader that will be used to make the request.  
mNetworkImageView.setImageUrl(IMAGE_URL, mImageLoader);
```

上面的代碼是通過前一節課的單例模式來實現訪問到RequestQueue與ImageLoader的。之所以這樣做得原因是：對於ImageLoader(一個用來處理加載與緩存圖片的幫助類)來說，單例模式可以避免旋轉所帶來的抖動。使用單例模式可以使得bitmap的緩存與activity的生命週期無關。如果你在activity中創建ImageLoader，這個ImageLoader有可能會在手機進行旋轉的時候被重新創建。這可能會導致抖動。

## 1.3)Example LRU cache

Volley工具箱中提供了通過DiskBasedCache實現的一種標準緩存。這個類能夠緩存文件到磁盤的制定目錄。但是為了使用ImageLoader，你應該提供一個自定義的內存LRC緩存，這個緩存需要實現 ImageLoader.ImageCache 的接口。你可能想把你的緩存設置成一個單例。關於更多的有關內容，請參考[建立請求隊列Setting Up a Request Queue](#).

下面是一個內存LRU Cache的實例。它繼承自LruCache並實現了ImageLoader.ImageCache的接口：

```
import android.graphics.Bitmap;
import android.support.v4.util.LruCache;
import android.util.DisplayMetrics;
import com.android.volley.toolbox.ImageLoader.ImageCache;

public class LruBitmapCache extends LruCache<String, Bitmap>
    implements ImageCache {

    public LruBitmapCache(int maxSize) {
        super(maxSize);
    }

    public LruBitmapCache(Context ctx) {
        this(getCacheSize(ctx));
    }

    @Override
    protected int sizeOf(String key, Bitmap value) {
        return value.getRowBytes() * value.getHeight();
    }

    @Override
    public Bitmap getBitmap(String url) {
        return get(url);
    }

    @Override
    public void putBitmap(String url, Bitmap bitmap) {
        put(url, bitmap);
    }

    // Returns a cache size equal to approximately three screens worth of images.
    public static int getCacheSize(Context ctx) {
        final DisplayMetrics displayMetrics = ctx.getResources().
            getDisplayMetrics();
        final int screenWidth = displayMetrics.widthPixels;
        final int screenHeight = displayMetrics.heightPixels;
        // 4 bytes per pixel
        final int screenBytes = screenWidth * screenHeight * 4;

        return screenBytes * 3;
    }
}
```

下面是如何初始化ImageLoader並使用cache的實例：

```
RequestQueue mRequestQueue; // assume this exists.
ImageLoader mImageLoader = new ImageLoader(mRequestQueue, new LruBitmapCache(LruBitmapCache.getCacheSize()));
```

## 2) Request JSON

Volley提供了以下的類用來執行JSON請求：

- `JSONArrayRequest` - 一個為了獲取JSONArray返回數據的請求。
- `JSONObjectRequest` - 一個為了獲取JSONObject返回數據的請求。允許把一個JSONObject作為請求參數。

這兩個類都是繼承自JsonRequest的。你可以使用類似的方法來處理這兩種類型的請求。如下演示瞭如果獲取一個JSON feed並顯示到UI上：

```
TextView mTxtDisplay;
ImageView mImageView;
mTxtDisplay = (TextView) findViewById(R.id.txtDisplay);
String url = "http://my-json-feed";

JsonObjectRequest jsObjRequest = new JsonObjectRequest
        (Request.Method.GET, url, null, new Response.Listener() {

    @Override
    public void onResponse(JSONObject response) {
        mTxtDisplay.setText("Response: " + response.toString());
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        // TODO Auto-generated method stub

    }
});

// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(jsObjRequest);
```

關於基於[Gson](#)實現一個自定義的JSON請求對象，請參考下一節課：[實現一個自定義的請求Implementing a Custom Request.](#)

# 實現自定義的網絡請求Implementing a Custom Request

編寫:kesenhoo - 原文:<http://developer.android.com/training/volley/request-custom.html>

這節課會介紹如何實現你自定義的請求類型，這些自定義的類型不屬於Volley內置支持包裏面。

## Write a Custom Request

大多數的請求類型都已經包含在Volley的工具箱裏面。如果你的請求返回數值是一個string，image或者JSON，那麼你是不需要自己去實現請求類的。

對於那些你需要自定義的請求類型，你需要執行以下操作：

- 繼承 `Request<T>` 類， `<T>` 表示請求返回的數據類型。因此如果你需要解析的響應類型是一個String，可以通過繼承 `Request<String>` 來創建你自定義的請求。請參考Volley工具類中的`StringRequest`與 `ImageRequest`來學習如何繼承`Request`。
- 實現抽象方法 `parseNetworkResponse()` 與 `deliverResponse()`，下面會詳細介紹。

### parseNetworkResponse

為了能夠提交一種指定類型的數據(例如，string，image，JSON等)，需要對解析後的結果進行封裝。下面會演示如何實現 `parseNetworkResponse()`。

```
@Override  
protected Response<T> parseNetworkResponse(  
    NetworkResponse response) {  
    try {  
        String json = new String(response.data,  
            HttpHeaderParser.parseCharset(response.headers));  
        return Response.success(gson.fromJson(json, clazz),  
            HttpHeaderParser.parseCacheHeaders(response));  
    }  
    // handle errors  
    ...  
}
```

請注意：

- `parseNetworkResponse()` 的參數是類型是 `NetworkResponse`，這種參數包含了的響應數據內容有一個`byte[]`，HTTP status code以及response headers。
- 你實現的方法必須返回一個`Response`，它包含了你響應對象與緩存`metadata`或者是一個錯誤。

如果你的協議沒有標準的cache機制，你可以自己建立一個 `Cache.Entry`，但是大多數請求都可以用下面的方式來處理：

```
return Response.success(myDecodedObject,  
    HttpHeaderParser.parseCacheHeaders(response));
```

Volley在工作線程中執行`parseNetworkResponse()`方法。這確保了耗時的解析操作，例如decode一張JPEG圖片成bitmap，不會阻塞UI線程。

## deliverResponse

Volley會把parseNetworkResponse()方法返回的數據帶到主線程的回調中。如下所示：

```
protected void deliverResponse(T response) {
    listener.onResponse(response);
```

## Example: GsonRequest

Gson是一個使用映射支持JSON與Java對象之間相互轉換的庫文件。你可以定義和JSON keys相對應名稱的Java對象。把對象傳遞給傳遞Gson，然後Gson會幫你為對象填充字段值。下面是一個完整的示例：演示了使用Gson解析Volley數據：

```
public class GsonRequest<T> extends Request<T> {
    private final Gson gson = new Gson();
    private final Class<T> clazz;
    private final Map<String, String> headers;
    private final Listener<T> listener;

    /**
     * Make a GET request and return a parsed object from JSON.
     *
     * @param url URL of the request to make
     * @param clazz Relevant class object, for Gson's reflection
     * @param headers Map of request headers
     */
    public GsonRequest(String url, Class<T> clazz, Map<String, String> headers,
                       Listener<T> listener, ErrorListener errorListener) {
        super(Method.GET, url, errorListener);
        this.clazz = clazz;
        this.headers = headers;
        this.listener = listener;
    }

    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        return headers != null ? headers : super.getHeaders();
    }

    @Override
    protected void deliverResponse(T response) {
        listener.onResponse(response);
    }

    @Override
    protected Response<T> parseNetworkResponse(NetworkResponse response) {
        try {
            String json = new String(
                response.data,
                HttpHeaderParser.parseCharset(response.headers));
            return Response.success(
                gson.fromJson(json, clazz),
                HttpHeaderParser.parseCacheHeaders(response));
        } catch (UnsupportedEncodingException e) {
            return Response.error(new ParseError(e));
        } catch (JsonSyntaxException e) {
            return Response.error(new ParseError(e));
        }
    }
}
```

如果你願意使用的話，Volley提供了現成的 `JsonArrayRequest` 與 `JsonObjectRequest` 類。參考上一課創建標準的網絡請求

# Android聯繫人信息與位置信息

---

編寫:spencer198711,Muyangmin - 原文:<http://developer.android.com/training/building-userinfo.html>

這幾節課為大家介紹如何在我們的app中添加用戶個人信息。我們可以通過識別用戶，提供用戶相關信息和提供用戶周圍的位置信息等方法來添加個人信息。

## Lessons

---

### 訪問聯繫人數據 - Accessing Contacts Data

如何使用Android的Contacts Provider來顯示和修改聯繫人信息。

### 位置信息- Making Your App Location-Aware

如何通過獲得用戶當前位置來給我們的App添加定位功能(位置感知)。

# 聯繫人信息

---

編寫:spencer198711 - 原文:<http://developer.android.com/training/contacts-provider/index.html>

**Contacts Provider**是用戶聯繫人信息的集中倉庫，它包含了來自聯繫人應用與社交應用的聯繫人數據。在我們的應用中，我們可以通過調用**ContentResolver**方法或者通過發送Intent給聯繫人應用來訪問**Contacts Provider**的信息。

這個章節會講解獲取聯繫人列表，顯示指定聯繫人詳情以及通過intent來修改聯繫人信息。這裏介紹的基礎技能能夠擴展到執行更複雜的任務。另外，這個章節也會幫助我們瞭解**Contacts Provider**的整個架構與操作方法。

## Lessons

---

### 獲取聯繫人列表

學習如何獲取聯繫人列表。你可以使用下面的技術來篩選需要的信息：

- 通過聯繫人名字進行篩選
- 通過聯繫人類型進行篩選
- 通過類似電話號碼等指定的一類信息進行篩選。

### 獲取聯繫人詳情

學習如何獲取單個聯繫人的詳情。一個聯繫人的詳細信息包括電話號碼與郵件地址等等。你可以獲取所有的詳細信息，也可以只獲取指定類型的詳細數據，例如郵件地址。

### 使用Intents修改聯繫人信息

學習如何通過發送intent給聯繫人應用來修改聯繫人信息。

### 顯示聯繫人頭像

學習如何顯示QuickContactBadge小組件。當用戶點擊聯繫人臂章(頭像)組件時，會打開一個對話框，這個對話框會顯示聯繫人詳情，並提供操作按鈕來處理詳細信息。例如，如果聯繫人信息有郵件地址，這個對話框可以顯示一個啓動默認郵件應用的操作按鈕。

# 獲取聯繫人列表

編寫:spencer198711 - 原文:<http://developer.android.com/training/contacts-provider/retrieve-names.html>

這一課展示瞭如何根據要搜索的字符串去匹配聯繫人的數據，從而得到聯繫人列表，你可以使用以下方法去實現：

## 匹配聯繫人名字

通過搜索字符串來匹配聯繫人名字的全部或者部分來獲得聯繫人列表。因為Contacts Provider允許多個實例擁有相同的名字，所以這種方法能夠返回匹配的列表。

## 匹配特定的數據類型，比如電話號碼

通過搜索字符串來匹配聯繩人的某一特定數據類型（如電子郵件地址），來取得符合要求的聯繫人列表。例如，這種方法可以列出電子郵件地址與搜索字符串相匹配的所有聯繫人。

## 匹配任意類型的數據

通過搜索字符串來匹配聯繫人詳情的所有數據類型，包括名字、電話號碼、地址、電子郵件地址等等。例如，這種方法接受任意數據類型的搜索字符串，並列出與這個搜索字符串相匹配的聯繫人。

Note：這一課的所有例子都使用CursorLoader獲取Contacts Provider中的數據。CursorLoader在一個與UI線程相獨立的工作線程進行查詢操作。這保證了數據查詢不會降低UI響應的時間，以免引起糟糕的用戶體驗。更多信息，請參照[在後臺加載數據](#)。

# 請求讀取聯繫人的權限

爲了能夠在Contacts Provider中做任意類型的搜索，我們的應用必須擁有READ\_CONTACTS權限。爲了擁有這個權限，我們需要在項目的manifest文件的節點中添加子結點，如下：

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

# 根據名字取得聯繫人並列出結果

這種方法根據搜索字符串，去匹配Contacts Provider的ContactsContract.Contacts表中的聯繫人名字。通常希望在ListView中展示結果，去讓用戶在所有匹配的聯繫人中做選擇。

## 定義ListView和列表項的佈局

爲了能夠將搜索結果展示在列表中，我們需要一個包含ListView以及其他佈局控件的主佈局文件，和一個定義列表中每一項的佈局文件。例如，可以使用以下XML代碼去創建主佈局文件res/layout/contacts\_list\_view.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

這個XML代碼使用了Android內建的ListView控件,他的id是android:id/list。

使用以下XML代碼定義列表項佈局文件contacts\_list\_item.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:clickable="true"/>
```

這個XML代碼使用了Android內建的TextView控件,他的id是android:text1。

Note：本課並不會描述如何從用戶那裏獲取搜索字符串的界面，因為我們可能會間接地獲取這個字符串。比如說，我們可能會給用戶一個選項去輸入文字信息，把這些文字信息作為搜索字符串去匹配聯繫人的名字。

剛剛寫的這兩個佈局文件定義了一個顯示ListView的用戶界面。下一步是編寫使用這個用戶界面顯示聯繫人列表的代碼。

## 定義一個顯示聯繫人列表的Fragment

為了顯示聯繫人列表，需要定義一個由Activity加載的Fragment。使用Fragment是一個比較靈活的方法，因為我們可以使用一個Fragment去顯示列表，用另一個Fragment顯示用戶在列表中選擇的聯繫人的詳情。使用這種方式，我們可以將本課程中展示的方法和另外一課獲取聯繫人詳情的方法聯繫起來。

想要學習如何在Activity中使用一個或者多個Fragment，請閱讀培訓課程[使用Fragment建立動態UI](#)。

為了方便我們編寫對Contacts Provider的查詢，Android框架提供了一個叫做ContactsContract的契約類，這個類定義了一些對查詢Contacts Provider很有用的常量和方法。當我們使用這個類的時候，我們不用自己定義內容URI、表名、列名等常量。使用這個類，只需要引入以下類聲明：

```
import android.provider.ContactsContract;
```

由於代碼中使用了CursorLoader去獲取provider的數據，所以我們必須實現加載器接口  
LoaderManager.LoaderCallbacks。同時，為了檢測用戶從結果列表中選擇了哪一個聯繫人，必須實現適配器接口  
AdapterView.OnItemClickListener。例如：

```
...
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 {
```

## 定義全局變量

定義在其他代部分碼中使用的全局變量：

```
...
/*
```

```

    * Defines an array that contains column names to move from
    * the Cursor to the ListView.
    */
@SuppressLint("InlinedApi")
private final static String[] FROM_COLUMNS = {
    Build.VERSION.SDK_INT
        >= Build.VERSION_CODES.HONEYCOMB ?
        Contacts.DISPLAY_NAME_PRIMARY :
        Contacts.DISPLAY_NAME
};

/*
    * Defines an array that contains resource ids for the layout views
    * that get the Cursor column contents. The id is pre-defined in
    * the Android framework, so it is prefaced with "android.R.id"
    */
private final static int[] TO_IDS = {
    android.R.id.text1
};

// Define global mutable variables
// Define a ListView object
ListView mContactsList;
// Define variables for the contact the user selects
// The contact's _ID value
long mContactId;
// The contact's LOOKUP_KEY
String mContactKey;
// A content URI for the selected contact
Uri mContactUri;
// An adapter that binds the result Cursor to the ListView
private SimpleCursorAdapter mCursorAdapter;
...

```

Note：由於Contacts.DISPLAY\_NAME\_PRIMARY需要在Android 3.0（API版本11）或者更高的版本才能使用，如果應用的minSdkVersion是10或者更低，會在eclipse中產生警告信息。為了關閉這個警告，我們可以在FROM\_COLUMNS定義之前加上@SuppressLint("InlinedApi")註解。

## 初始化Fragment

爲了初始化Fragment，Android系統需要我們爲這個Fragment添加空的、公有的構造方法，同時在回調方法onCreateView()中綁定界面。例如：

```

// Empty public constructor, required by the system
public ContactsFragment() {}
// A UI Fragment must inflate its View
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    // Inflate the fragment layout
    return inflater.inflate(R.layout.contact_list_fragment,
        container, false);
}

```

## 爲ListView設置CursorAdapter

設置SimpleCursorAdapter，將搜索結果綁定到ListView。爲了獲得顯示聯繫人列表的ListView控件，需要使用Fragment的父Activity調用Activity.findViewById()。當調用setAdapter()的時候，需要使用父Activity的上下文(Context)。

```

public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    ...
    // Gets the ListView from the View list of the parent activity
}

```

```
mContactsList =  
    (ListView) getActivity().findViewById(R.layout.contact_list_view);  
// Gets a CursorAdapter  
mCursorAdapter = new SimpleCursorAdapter(  
    getActivity(),  
    R.layout.contact_list_item,  
    null,  
    FROM_COLUMNS, TO_IDS,  
    0);  
// Sets the adapter for the ListView  
mContactsList.setAdapter(mCursorAdapter);  
}
```

## 為選擇的聯繫人設置監聽器

當我們顯示搜索列表結果的時候，我們通常會讓用戶選擇某一個聯繫人去做進一步的處理。例如，當用戶選擇某一個聯繫人的时候，可以在地圖上顯示這個聯繫人的地址。為了能夠提供這個功能，我們需要定義當前的Fragment為一個點擊監聽器，這需要這個類實現AdapterView.OnItemClickListener接口，就像前面介紹的定義顯示聯繫人列表的Fragment那樣。

繼續設置這個監聽器，需要在onActivityCreated()方法中調用setOnItemClickListener()以使得這個監聽器綁定到ListView。例如：

```
public void onActivityCreated(Bundle savedInstanceState) {  
    ...  
    // Set the item click listener to be the current fragment.  
    mContactsList.setOnItemClickListener(this);  
    ...  
}
```

由於指定了當前的Fragment作為ListView的點擊監聽器，現在我們需要實現處理點擊事件的onItemClick()方法。這個會在隨後討論。

## 定義查詢映射

定義一個常量，這個常量包含我們想要從查詢結果中返回的列。Listview中的每一項顯示了一個聯繫人的名字。在Android 3.0 (API version 11) 或者更高的版本，這個列的名字是Contacts.DISPLAY\_NAME\_PRIMARY；在Android 3.0之前，這個列的名字是Contacts.DISPLAY\_NAME。

在SimpleCursorAdapter綁定過程中會用到Contacts.\_ID列。Contacts.\_ID和LOOKUP\_KEY一同用來構建用戶選擇的聯繫人的內容URI。

```
...  
@SuppressLint("InlinedApi")  
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  
};
```

## 定義Cursor的列索引常量

為了從Cursor中獲得單獨某一列的數據，我們需要知道這一列在Cursor中的索引值。我們需要定義Cursor列的索引值，

這些索引值與我們定義查詢映射的列的順序是一樣的。例如：

```
// The column index for the _ID column
private static final int CONTACT_ID_INDEX = 0;
// The column index for the LOOKUP_KEY column
private static final int LOOKUP_KEY_INDEX = 1;
```

## 指定查詢標準

為了指定我們想要的數據，我們需要創建一個包含文本表達式和變量的組合，去告訴provider我們需要的數據列和想要的值。

對於文本表達式，定義一個常量，列出所有搜索到的列。儘管這個表達式可以包含變量值，但是建議用"?佔位符來替代這個值。在搜索的時候，佔位符裏的值會被數組裏的值所取代。使用"?佔位符確保了搜索條件是由綁定產生而不是由SQL編譯產生。這個方法消除了惡意SQL注入的可能。例如：

```
// Defines the text expression
@SuppressLint("InlinedApi")
private static final String SELECTION =
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
        Contacts.DISPLAY_NAME_PRIMARY + " LIKE ?" :
        Contacts.DISPLAY_NAME + " LIKE ?";
// Defines a variable for the search string
private String mSearchString;
// Defines the array to hold values that replace the ?
private String[] mSelectionArgs = { mSearchString };
```

## 定義onItemClick()方法

在之前的內容中，我們為Listview設置了列表項點擊監聽器，現在需要定義AdapterView.OnItemClickListener.onClick()方法以實現監聽器行爲：

```
@Override
public void onItemClick(
    AdapterView<?> parent, View item, int position, long rowID) {
    // Get the Cursor
    Cursor cursor = parent.getAdapter().getCursor();
    // Move to the selected contact
    cursor.moveToPosition(position);
    // Get the _ID value
    mContactId = getLong(CONTACT_ID_INDEX);
    // Get the selected LOOKUP KEY
    mContactKey = getString(CONTACT_KEY_INDEX);
    // Create the contact's content Uri
    mContactUri = Contacts.getLookupUri(mContactId, mContactKey);
    /*
     * You can use mContactUri as the content URI for retrieving
     * the details for a contact.
     */
}
```

## 初始化Loader

由於使用了CursorLoader獲取數據，我們必須初始化後臺線程和其他的控制異步獲取數據的變量。需要在onActivityCreated()方法中做初始化的工作，這個方法是在Fragment的界面顯示之前被調用的，相關代碼展示如下：

```
public class ContactsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor> {
...
    // Called just before the Fragment displays its UI
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        // Always call the super method first
        super.onActivityCreated(savedInstanceState);
        ...
        // Initializes the loader
        getLoaderManager().initLoader(0, null, this);
    }
}
```

## 實現onCreateLoader()方法

我們需要實現onCreateLoader()方法，這個方法是在調用initLoader()後馬上被loader框架調用的。

在onCreateLoader()方法中，設置搜索字符串模式。為了讓一個字符串符合一個模式，插入 "%" 字符代表0個或多個字符，插入 "\_" 代表一個字符。例如，模式%Jefferson%將會匹配“Thomas Jefferson”和“Jefferson Davis”。

這個方法返回一個CursorLoader對象。對於內容URI，則使用了Contacts.CONTENT\_URI，這個URI關聯到整個表，例子如下所示：

```
...
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
    /*
     * Makes search string into pattern and
     * stores it in the selection array
     */
    mSelectionArgs[0] = "%" + mSearchString + "%";
    // Starts the query
    return new CursorLoader(
        getActivity(),
        Contacts.CONTENT_URI,
        PROJECTION,
        SELECTION,
        mSelectionArgs,
        null
    );
}
```

## 實現onLoadFinished()方法和onLoaderReset()方法

實現onLoadFinished()方法。當Contacts Provider返回查詢結果的時候，loader框架會調用onLoadFinished()方法。在這個方法中，將查詢結果Cursor傳給SimpleCursorAdapter，這將會使用這個查詢結果自動更新ListView。

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    // Put the result Cursor in the adapter for the ListView
    mCursorAdapter.swapCursor(cursor);
}
```

當loader框架檢測到結果集Cursor包含過時的數據時，它會調用onLoaderReset()。我們需要刪除SimpleCursorAdapter對已經存在Cursor的引用。如果不這麼做的話，loader框架將不會回收Cursor對象，這將會導致內存泄漏。例如：

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // Delete the reference to the existing Cursor
    mCursorAdapter.swapCursor(null);
```

```
}
```

我們現在已經實現了一個應用的關鍵部分，即根據搜索字符串匹配聯繫人名字和將獲得的結果展示在ListView中。用戶可以點擊選擇一個聯繫人名字，這將會觸發一個監聽器，在監聽器的回調函數中，你可以使用此聯繫人的數據做進一步的處理。例如，你可以進一步獲取此聯繫人的詳情，想要知道何如獲取聯繫人詳情，請繼續學習下一課[獲取聯繫人詳情](#)。

想要瞭解更多搜索用戶界面的知識，請參考API指南[Creating a Search Interface](#)。

這一課的以下內容展示了在Contacts Provider中查找聯繫人的其他方法。

## 根據特定的數據類型匹配聯繫人

這種方法可以讓我們指定想要匹配的數據類型。根據名字去檢索是這種類型的查詢的一個具體例子。但也可以用任何與聯繫人詳情數據相關的數據類型去做查詢。例如，我們可以檢索具有特定郵政編碼聯繫人，在這種情況下，搜索字符串將會去匹配存儲在一個郵政編碼列中的數據。

為了實現這種類型的檢索，首先實現以下的代碼，正如之前的內容所展示的：

- 請求讀取聯繫人的權限
- 定義列表和列表項的佈局
- 定義顯示聯繫人列表的Fragment
- 定義全局變量
- 初始化Fragment
- 為ListView設置CursorAdapter
- 設置選擇聯繫人的監聽器
- 定義Cursor的列索引常量

儘管我們現在從不同的表中取數據，檢索列的映射順序是一樣的，所以我們可以為這個Cursor使用同樣的索引常量。

- 定義onItemClick()方法
- 初始化loader
- 實現onLoadFinished()方法和onLoaderReset()方法

為了將搜索字符串匹配特定的詳請數據類型並顯示結果，以下的步驟展示了我們需要額外添加的代碼。

### 選擇要查詢的數據類型和數據庫表

為了從特定類型的詳請數據中查詢，我們必須知道的數據類型的自定義MIME類型的值。每一個數據類型擁有唯一的 MIME 類型值，這個值在ContactsContract.CommonDataKinds的子類中被定義為常量 `CONTENT_ITEM_TYPE`，並且與實際的數據類型相關。子類的名字會表明它們的實際數據類型。例如，`email`數據的子類是 `ContactsContract.CommonDataKinds.Email`，並且`email`的自定義MIME類型是 `Email.CONTENT_ITEM_TYPE`。

在搜索中需要使用ContactsContract.Data類。同時所有需要的常量，包括數據映射、選擇字句、排序規則都是由這個類定義或繼承自此類。

### 定義查詢映射

為了定義一個查詢映射，請選擇一個或者多個定義在ContactsContract.Data表或其子類的列。Contacts Provider在返回行結果集之前，隱式的連接了ContactsContract.Data表和其他表。例如：

```
@SuppressLint("InlinedApi")
private static final String[] PROJECTION = {
    /*
     * The detail data row ID. To make a ListView work,
     * this column is required.
     */
    Data._ID,
    // The primary display name
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
        Data.DISPLAY_NAME_PRIMARY :
        Data.DISPLAY_NAME,
    // The contact's _ID, to construct a content URI
    Data.CONTACT_ID
    // The contact's LOOKUP_KEY, to construct a content URI
    Data.LOOKUP_KEY (a permanent link to the contact
};


```

## 定義查詢標準

為了根據特定的聯繫人數據類型查詢字符串，請按照以下方法構建查詢選擇子句：

- 包含搜索字符串的列名。這個名字根據數據類型所變化，所以我們需要找到與數據類型對應的 `ContactsContract.CommonDataKinds` 的子類，並從這個子類中選擇列名。例如，想要搜索email地址，需要使用 `Email.ADDRESS` 列。
- 搜索字符串本身，請在查詢選擇子句裏使用 "?" 表示。
- 列名包含自定義的 MIME 類型值。這個列名字總是 `Data.MIMETYPE`。
- 自定義 MIME 類型值的數據類型。如之前描述，這需要使用 `ContactsContract.CommonDataKinds` 子類中的 `CONTENT_ITEM_TYPE` 常量。例如，`email` 數據的 MIME 類型值是 `Email.CONTENT_ITEM_TYPE`。需要在這個常量值的開頭和結尾加上 "" (單引號)。否則，provider 會把這個值翻譯成一個變量而不是一個字符串。我們不需要為這個值提供佔位符，因為我們在使用一個常量而不是用戶提供的值。例如：

```
/*
 * Constructs search criteria from the search string
 * and email MIME type
*/
private static final String SELECTION =
    /*
     * Searches for an email address
     * that matches the search string
     */
    Email.ADDRESS + " LIKE ? " + "AND " +
    /*
     * Searches for a MIME type that matches
     * the value of the constant
     * Email.CONTENT_ITEM_TYPE. Note the
     * single quotes surrounding Email.CONTENT_ITEM_TYPE.
     */
    Data.MIMETYPE + " = '" + Email.CONTENT_ITEM_TYPE + "'";
```

下一步，定義包含選擇字符串的變量：

```
String mSearchString;
String[] mSelectionArgs = { "" };
```

## 實現`onCreateLoader()`方法

現在，我們已經詳述了想要的數據和如何找到這些數據，如何在 `onCreateLoader()` 方法中定義一個查詢。使用你的數據

映射、查詢選擇表達式和一個數組作為選擇表達式的參數，並從這個方法中返回一個新的CursorLoader對象。而內容URI需要使用Data.CONTENT\_URI，例如：

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
    // OPTIONAL: Makes search string into pattern
    mSearchString = "%" + mSearchString + "%";
    // Puts the search string into the selection criteria
    mSelectionArgs[0] = mSearchString;
    // Starts the query
    return new CursorLoader(
        getActivity(),
        Data.CONTENT_URI,
        PROJECTION,
        SELECTION,
        mSelectionArgs,
        null
    );
}
```

這段代碼片段是基於特定的聯繫人詳情數據類型的簡單反向查找。如果我們的應用關注於某一種特定的數據類型，比如說email地址，並且允許用戶獲得與此數據相關的聯繫人名字，這種形式的查詢是最好的方法。

## 根據任意類型的數據匹配聯繫人

根據任意數據類型獲取聯繫人時，如果聯繫人的數據（這些數據包括名字、email地址、郵件地址和電話號碼等等）能匹配要搜索的字符串，那麼該聯繫人信息將會被返回。這種搜索結果會比較廣泛。例如，如果搜索字符串是"Doe"，搜索任意類型的數據將會返回名字為"Jone Doe"的聯繫人，也會返回一個住在"Doe Street"的聯繫人。

為了完成這種類型的查詢，就像之前展示的那樣，首先需要實現以下代碼：

- 請求讀取聯繫人的權限
- 定義列表和列表項的佈局
- 定義顯示聯繫人列表的Fragment
- 定義全局變量
- 初始化Fragment
- 為ListView設置CursorAdapter
- 設置選擇聯繫人的監聽器
- 定義Cursor的列索引常量

對於這種形式的查詢，你需要使用與在“使用特定類型的數據匹配聯繫人”那一節中相同的表，也可以使用相同的列索引。

- 定義onItemClick()方法
- 初始化loader
- 實現onLoadFinished()方法和onLoaderReset()方法

以下的步驟展示了為了能夠根據任意的數據類型去匹配查詢字符串並顯示結果列表，我們需要添加的額外代碼。

## 去除查詢標準

不需要為mSelectionArgs定義查詢標準常量SELECTION。這些內容在這種類型的檢索不會被用到。

## 實現onCreateLoader()方法

實現`onCreateLoader()`方法，返回一個新的`CursorLoader`對象。我們不需要把搜索字符串轉化成一個搜索模式，因為`Contacts Provider`會自動做這件事。使用`Contacts.CONTENT_FILTER_URI`作為基礎查詢URI，並使用`Uri.withAppendedPath()`方法將搜索字符串添加到基礎URI中。使用這個URI會自動觸發對任意數據類型的搜索，就像以下例子所示：

```
@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
    /*
     * Appends the search string to the base URI. Always
     * encode search strings to ensure they're in proper
     * format.
     */
    Uri contentUri = Uri.withAppendedPath(
        Contacts.CONTENT_FILTER_URI,
        Uri.encode(mSearchString));
    // Starts the query
    return new CursorLoader(
        getActivity(),
        contentUri,
        PROJECTION,
        null,
        null,
        null
    );
}
```

這段代碼片段，是想要在`Contacts Provider`中建立廣泛搜索類型應用的基礎部分。這種方法對那些想要實現與通訊錄應用聯繫人列表中相似搜索功能的應用，會很有幫助。

# 獲取聯繫人詳情

編寫:spencer198711 - 原文:<http://developer.android.com/training/contacts-provider/retrieve-details.html>

這一課展示瞭如何取得一個聯繫人的詳細信息，比如email地址、電話號碼等。當使用者去獲取聯繫人信息的時候，這些信息正是他們所查找的。我們可以給他們關於一個聯繫人的所有信息，或者僅僅顯示一個特定的數據類型，比如email地址。

這一課假設你已經獲取到了一個用戶所選取的聯繫人的ContactsContract.Contacts數據項。在[獲取聯繫人名字那一課](#)展示了如何獲取聯繫人列表。

## 獲取聯繫人的所有詳細信息

爲了取得一個聯繫人的所有詳情，查找ContactsContract.Data表中包含聯繫人LOOKUP\_KEY列的任意行。因爲Contacts Provider隱式地連接了ContactsContract.Contacts表和ContactsContract.Data表，所以這個LOOKUP\_KEY列在ContactsContract.Data表中是可用的。關於LOOKUP\_KEY列，在[獲取聯繫人名字那一課](#)有詳細的描述。

Note：檢索一個聯繫人的所有信息會降低設備的性能，因爲這需要檢索ContactsContract.Data表的所有列。在使用這種方法之前，請認真考慮對性能影響。

## 請求權限

爲了能夠讀Contacts Provider，我們的應用必須擁有READ\_CONTACTS權限。爲了請求這個權限，需要在manifest文件的中添加如下子節點：

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

## 設置查詢映射

根據一行數據的數據類型，它可能會使用很多列或者只使用幾列。另外，數據會根據不同的數據類型而出現在不同的列中。爲了確保能夠獲取所有數據類型的所有可能的數據列，需要在查詢映射中添加所有列的名字。如果要把Cursor綁定到ListView，記得要獲取Data.\_ID，否則的話，界面綁定就不會起作用。同時也需要獲取Data.MIMETYPE列，這樣才能識別我們獲取到的每一行數據的數據類型。例如：

```
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  
};
```

這個查詢映射使用了ContactsContract.Data類中定義的列名字，去獲取ContactsContract.Data表中一行的所有數據列。

我們也可以使用由ContactsContract.Data或其子類定義的列常量去設置查詢映射。需要注意的是，從SYNC1到SYNC4的數據列是sync adapter同步數據所使用的，它們的值對我們沒有意義。

## 定義查詢標準

為查詢選擇子句定義一個常量，一個包含查詢選擇參數的數組，以及一個保存查詢選擇值的變量。使用Contacts.LOOKUP\_KEY列去查找這個聯繫人。例如：

```
// Defines the selection clause  
private static final String SELECTION = Data.LOOKUP_KEY + " = ?";  
// Defines the array to hold the search criteria  
private String[] mSelectionArgs = { "" };  
/*  
 * Defines a variable to contain the selection value. Once you  
 * have the Cursor from the Contacts table, and you've selected  
 * the desired row, move the row's LOOKUP_KEY value into this  
 * variable.  
 */  
private String mLookupKey;
```

在查詢選擇表達式中使用“?”佔位符，確保了搜索是由綁定生成而不是由SQL編譯生成。這種方法消除了惡意SQL注入的可能性。

## 定義排序順序

定義在查詢結果Cursor中希望的排序順序。按照Data.MIMETYPE去排序，可以讓特定數據類型的所有行排列在一起。這種形式的查詢排序參數讓所有具有email的行排在一起，讓所有具有電話的行排在一起....例如：

```
/*  
 * Defines a string that specifies a sort order of MIME type  
 */  
private static final String SORT_ORDER = Data.MIMETYPE;
```

Note：一些數據類型不使用子類型，所以不能按照子類型來排序。作為替代方法，我們不得不遍歷返回的Cursor，去判定當前行的數據類型，為那些使用子類型的數據行保存數據。當讀取完cursor後，我們可以根據子類型去排序每一個數據類型並顯示結果。

## 初始化查詢loader

永遠在後臺線程中去檢索Contacts Provider(或者其他content provider)的數據。使用Loader框架中的LoaderManager類和LoaderManager.LoaderCallbacks在後臺去做獲取數據的工作。

當我們已經準備好去獲取數據行，需要通過調用initLoader()方法去初始化loader框架。傳遞一個Integer類型的標識符給initLoader()方法，這個標識符會傳遞給LoaderManager.LoaderCallbacks方法。當在一個應用中使用多個loader時，這個標識符能夠幫助我們區分它們。

以下的代碼片段展示瞭如何初始化loader框架：

```

public class DetailsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor> {
    ...
    // Defines a constant that identifies the loader
    DETAILS_QUERY_ID = 0;
    ...
    /*
     * Invoked when the parent Activity is instantiated
     * and the Fragment's UI is ready. Put final initialization
     * steps here.
     */
    @Override
    onActivityCreated(Bundle savedInstanceState) {
        ...
        // Initializes the loader framework
        getLoaderManager().initLoader(DETAILS_QUERY_ID, null, this);
    }
}

```

## 實現onCreateLoader()方法

實現onCreateLoader()方法。loader框架會在我們調用initLoader()方法後立即調用onCreateLoader()方法。這個方法會返回一個CursorLoader對象。由於搜索的是ContactsContract.Data表，所以需要使用常量Data.CONTENT\_URI作為內容URI。例如：

```

@Override
public Loader<Cursor> onCreateLoader(int loaderId, Bundle args) {
    // Choose the proper action
    switch (loaderId) {
        case DETAILS_QUERY_ID:
            // Assigns the selection parameter
            mSelectionArgs[0] = mLookupKey;
            // Starts the query
            CursorLoader mLoader =
                new CursorLoader(
                    getActivity(),
                    Data.CONTENT_URI,
                    PROJECTION,
                    SELECTION,
                    mSelectionArgs,
                    SORT_ORDER
                );
            ...
    }
}

```

## 實現onLoadFinished()方法和onLoaderReset()方法

實現onLoadFinished()方法。當Contacts Provider返回查詢結果的時候，loader框架會調用onLoadFinished()方法。例如：

```

public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    switch (loader.getId()) {
        case DETAILS_QUERY_ID:
            /*
             * Process the resulting Cursor here.
             */
        }
        break;
    ...
}

```

當loader框架檢測到結果集Cursor所對應的數據已經發生變化的時候，會調用onLoaderReset()方法。這時，需要通過

把Cursor設置為null來移除對已經存在Cursor對象的引用。否則，loader框架就不會銷燬舊的Cursor對象，從而導致內存泄漏。例如：

```
@Override  
public void onLoaderReset(Loader<Cursor> loader) {  
    switch (loader.getId()) {  
        case DETAILS_QUERY_ID:  
            /*  
             * If you have current references to the Cursor,  
             * remove them here.  
             */  
            break;  
    }  
}
```

## 獲取聯繫人的特定類型的信息

獲取聯繫人的特定類型的信息，例如所有的email信息，跟獲取聯繫人的所有詳細信息類似。下面的內容是在[獲取聯繫人所有詳細信息](#)列出的代碼的基礎上作出的修改：

### 查詢映射

修改查詢映射使得能夠針對特定的數據類型去獲取列。同時需要修改查詢映射，來把在ContactsContract.CommonDataKinds子類中定義的列常量與數據類型對應起來。

### 查詢選擇

修改查詢選擇子句去搜索特定類型的MIMETYPE值。

### 排序順序

由於僅僅搜索一種類型的詳細數據，所以不需要將返回的Cursor按照Data.MIMETYPE進行分組。

這些修改將會在下面的小節中詳細描述。

## 設置查詢映射

使用ContactsContract.CommonDataKinds的特定類型子類所定義的列名稱常量，定義我們想要獲取的數據列。如果我們打算把Cursor綁定到ListView，確保要獲取 \_ID 列。例如，為了獲取email數據，需要定義以下數據映射：

```
private static final String[] PROJECTION =  
{  
    Email._ID,  
    Email.ADDRESS,  
    Email.TYPE,  
    Email.LABEL  
};
```

需要注意的是，這個查詢映射使用在ContactsContract.CommonDataKinds.Email類中定義的列名稱，來替代ContactsContract.Data類中定義的列名稱。使用email類型的列名稱使得代碼更具可讀性。

在查詢映射中，我們也可以使用ContactsContract.CommonDataKinds子類所定義的其他數據列。

## 定義查詢標準

根據我們想要找的特定聯繫人的LOOKUP\_KEY和聯繫人詳細信息的Data.MIMETYPE定義一個搜索表達式，去獲取數據。把MIMETYPE的值從頭到尾用單引號括住，否則的話，content provider將會把這個常量當成變量名而不是字符串。因為我們使用的是常量，而不是用戶提供的值，所以這裏不需要使用佔位符。例如：

```
/*
 * Defines the selection clause. Search for a lookup key
 * and the Email MIME type
 */
private static final String SELECTION =
    Data.LOOKUP_KEY + " = ?" +
    " AND " +
    Data.MIMETYPE + " = " +
    "!" + Email.CONTENT_ITEM_TYPE + "!";
// Defines the array to hold the search criteria
private String[] mSelectionArgs = { "" };
```

## 定義排序規則

為查詢返回的Cursor定義一個排序規則。由於是檢索特定的數據類型，刪除根據MIMETYPE來排序的部分。而如果查詢的詳細數據類型包含子類型，可以根據這個子類型去排序。例如，對於email數據，我們可以根據Email.TYPE排序：

```
private static final String SORT_ORDER = Email.TYPE + " ASC ";
```

# 使用Intent修改聯繫人信息

編寫:spencer198711 - 原文:<http://developer.android.com/training/contacts-provider/modify-data.html>

這一課介紹如何使用Intent去插入一個新的聯繫人或者修改聯繫人的數據。我們不是直接訪問Contacts Provider，而是通過Intent啓動Contacts應用去運行適當的Activity。對於這一課中描述的數據修改行為，如果你向Intent發送擴展的數據，它會自動填充進啓動的Activity頁面中。

使用Intent去插入或者更新一個聯繫人是比較推薦的修改Contacts Provider的做法。原因如下：

- 節省了我們自行開發UI和編寫代碼的時間和精力。
- 避免了由於不按照Contacts Provider的規則去修改而產生的錯誤。
- 減少應用需要申請的權限數量。因為我們的應用把修改行為委託給已經擁有寫Contacts Provider權限的Contacts 應用，所以我們的應用不需要再去申請這個權限，。

## 使用Intent插入新的聯繫人

當我們的應用接收到新的數據時，我們通常會允許用戶去插入一個新的聯繫人。例如，一個餐館評論應用可以允許用戶在評論餐館的時候，把這個餐館添加為一個聯繫人。可以使用Intent去做這個任務，使用我們擁有的儘可能多的數據去創建對應的Intent，然後發送這個Intent到Contacts應用。

使用Contacts應用去插入一個聯繫人將會向Contacts Provider中的ContactsContract.RawContacts表中插入一個原始聯繫人。必要的情況下，在創建原始聯繩人的情況下，Contacts應用將會提示用戶選擇賬戶類型和要使用的賬戶。如果聯繫人已經存在，Contacts應用也會告知用戶。用戶將會有取消插入的選項，在這種情況下不會有聯繫人被創建。想要知道更多關於原始聯繩人的信息，請參閱Contacts Provider的API指導。

### 創建一個Intent

利用Intents.Insert.ACTION創建一個新的Intent對象，並設置其MIME類型為RawContacts.CONTENT\_TYPE。例如：

```
...
// Creates a new Intent to insert a contact
Intent intent = new Intent(Intents.Insert.ACTION);
// Sets the MIME type to match the Contacts Provider
intent.setType(ContactsContract.RawContacts.CONTENT_TYPE);
```

如果我們已經獲得了此聯繫人的詳細信息，比如說電話號碼或者email地址，那麼我們可以把它們作為擴展數據添加到Intent中。對於鍵值，需要使用Intents.Insert中對應的常量。Contacts應用將會在插入界面顯示這些數據，以便用戶作進一步的數據編輯和數據添加。

```
/* Assumes EditText fields in your UI contain an email address
 * and a phone number.
 */
private EditText mEmailAddress = (EditText) findViewById(R.id.email);
private EditText mPhoneNumber = (EditText) findViewById(R.id.phone);
...
/*
 * Inserts new data into the Intent. This data is passed to the
 * contacts app's Insert screen
 */
```

```
// Inserts an email address
intent.putExtra(Intents.Insert.EMAIL, mEmailAddress.getText())
/*
 * In this example, sets the email type to be a work email.
 * You can set other email types as necessary.
 */
    .putExtra(Intents.Insert.EMAIL_TYPE, CommonDataKinds.Email.TYPE_WORK)
// Inserts a phone number
    .putExtra(Intents.Insert.PHONE, mPhoneNumber.getText())
/*
 * In this example, sets the phone type to be a work phone.
 * You can set other phone types as necessary.
 */
    .putExtra(Intents.Insert.PHONE_TYPE, Phone.TYPE_WORK);
```

一旦我們創建好Intent，調用startActivity()將其發送到Contacts應用。

```
/* Sends the Intent
 */
startActivity(intent);
```

這個調用將會打開Contacts應用的界面，並允許用戶進入一個新的聯繫人。這個聯繫人的賬戶類型和賬戶名字列在屏幕的上方。一旦用戶輸入數據並點擊確定，Contacts應用的聯繫人列表則會顯示出來。用戶可以點擊Back鍵返回到我們自己創建的應用。

## 使用Intent編輯已經存在的聯繫人

如果用戶已經選擇了一個感興趣的聯繫人，使用Intent去編輯這個已存在的聯繫人會很有用。例如，一個用來查找擁有郵政地址但是缺少郵政編碼的聯繫人的應用，可以給用戶提供查找郵政編碼的選項，然後把找到的郵政編碼添加到這個聯繫人中。

使用Intent編輯已經存在的聯繫人，同插入一個聯繫人的步驟類似。像前面介紹的[使用Intent插入新的聯繫人](#)創建一個Intent，但是需要給這個Intent添加對應聯繫人的[Contacts.CONTENT\\_LOOKUP\\_URI](#)和MIME類型[Contacts.CONTENT\\_ITEM\\_TYPE](#)。如果想要使用已經擁有的詳情信息編輯這個聯繫人，我們需要把這些數據放到Intent的擴展數據中。同時注意有些列是不能使用Intent編輯的，這些不可編輯的列在[ContactsContract.Contacts](#)摘要部分“Update”標題下有列出。

最後，發送這個Intent。Contacts應用會顯示一個編輯界面作為迴應。當用戶編輯完成並保存，Contacts應用會顯示一個聯繫人列表。當用戶點擊Back，我們自己的應用會出現。

## 創建Intent

為了能夠編輯一個聯繫人，需要調用Intent(action)去創建一個擁有ACTION\_EDIT行為的Intent。調用setDataAndType()去設置這個Intent要編輯的聯繫人的[Contacts.CONTENT\\_LOOKUP\\_URI](#)和MIME類型[Contacts.CONTENT\\_ITEM\\_TYPE](#)。因為調用setType()會重寫Intent當前的數據，所以我們必須同時設置數據和MIME類型。

為了得到聯繫人的[Contacts.CONTENT\\_LOOKUP\\_URI](#)，需要調用[Contacts.getLookupUri\(id, lookupkey\)](#)方法，該方法的參數分別是聯繫人的[Contacts.\\_ID](#)和[Contacts.LOOKUP\\_KEY](#)。

以下的代碼片段展示瞭如何創建這個Intent：

```
// The Cursor that contains the Contact row
public Cursor mCursor;
```

```
// The index of the lookup key column in the cursor
public int mLookupKeyIndex;
// The index of the contact's _ID value
public int mIdIndex;
// The lookup key from the Cursor
public String mCurrentLookupKey;
// The _ID value from the Cursor
public long mCurrentId;
// A content URI pointing to the contact
Uri mSelectedContactUri;
...
/*
 * Once the user has selected a contact to edit,
 * this gets the contact's lookup key and _ID values from the
 * cursor and creates the necessary URI.
 */
// Gets the lookup key column index
mLookupKeyIndex = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Gets the lookup key value
mCurrentLookupKey = mCursor.getString(mLookupKeyIndex);
// Gets the _ID column index
mIdIndex = mCursor.getColumnIndex(Contacts._ID);
mCurrentId = mCursor.getLong(mIdIndex);
mSelectedContactUri =
    Contacts.getLookupUri(mCurrentId, mCurrentLookupKey);
...
// Creates a new Intent to edit a contact
Intent editIntent = new Intent(Intent.ACTION_EDIT);
/*
 * Sets the contact URI to edit, and the data type that the
 * Intent must match
 */
editIntent.setDataAndType(mSelectedContactUri, Contacts.CONTENT_ITEM_TYPE);
```

## 添加導航標誌

在Android 4.0（API版本14）和更高的版本，Contacts應用中的一個問題會導致錯誤的頁面導航。我們的應用發送一個編輯聯繫人的Intent到Contacts應用，用戶編輯並保存這個聯繫人，當用戶點擊Back鍵的時候會看到聯繫人列表頁面。用戶需要點擊最近使用的應用，然後選擇我們的應用，才能返回到我們自己的應用。

要在Android 4.0.3（API版本15）及以後的版本解決此問題，需要添加 `finishActivityOnSaveCompleted` 擴展數據參數到這個Intent，並將它的值設置為true。Android 4.0之前的版本也能夠接受這個參數，但是不起作用。為了設置擴展數據，請按照以下方式去做：

```
// Sets the special extended data for navigation
editIntent.putExtra("finishActivityOnSaveCompleted", true);
```

## 添加其他的擴展數據

對Intent添加額外的擴展數據，需要調用`putExtra()`。可以為常見的聯繫人數據字段添加擴展數據，這些常見字段的key值可以從[Intents.Insert API參考文檔](#)中查到。記住`ContactsContract.Contacts`表中有些列是不能編輯的，這些列在[ContactsContract.Contacts](#)的摘要部分“Update”標題下有列出。

## 發送Intent

最後，發送我們已經構建好的Intent。例如：

```
// Sends the Intent
startActivity(editIntent);
```

## 使用Intent讓用戶去選擇是插入還是編輯聯繫人

---

我們可以通過發送帶有 `ACTION_INSERT_OR_EDIT` 行爲的Intent，讓用戶去選擇是插入聯繫人還是編輯已有的聯繫人。例如，一個email客戶端應用會允許用戶添加一個收件地址到新的聯繫人，或者僅僅作爲額外的郵件地址添加到已有的聯繫人。需要爲這個Intent設置MIME類型`Contacts.CONTENT_ITEM_TYPE`，但是不需要設置數據URI。

當我們發送這個Intent後，`Contacts`應用會展示一個聯繫人列表。用戶可以選擇是插入一個新的聯繫人還是挑選一個存在的聯繫人去編輯。任何添加到Intent中的擴展數據字段都會填充在界面上。我們可以使用任何在[Intents.Insert](#)中指定的的key值。以下的代碼片段展示瞭如何構建和發送這個Intent：

```
// Creates a new Intent to insert or edit a contact
Intent intentInsertEdit = new Intent(Intent.ACTION_INSERT_OR_EDIT);
// Sets the MIME type
intentInsertEdit.setType(Contacts.CONTENT_ITEM_TYPE);
// Add code here to insert extended data, if desired
...
// Sends the Intent with an request ID
startActivity(intentInsertEdit);
```

# 顯示聯繫人頭像

編寫:spencer198711 - 原文:<http://developer.android.com/training/contacts-provider/display-contact-badge.html>

這一課展示瞭如何在我們的應用界面上添加一個QuickContactBadge，以及如何爲它綁定數據。QuickContactBadge是一個在初始情況下顯示聯繫人縮略圖頭像的widget。儘管我們可以使用任何Bitmap作爲縮略圖頭像，但是我們通常會使用從聯繫人照片縮略圖中解碼出來的Bitmap。

這個小的圖片是一個控件，當用戶點擊它時，QuickContactBadge會展開一個包含以下內容的對話框：

- 一個大的聯繫人頭像

與這個聯繫人關聯的大的頭像，如果此人沒有設置頭像，則顯示預留的圖案。

- 應用程序圖標

根據聯繫人詳情數據，顯示每一個能夠被手機中的應用所處理的數據的圖標。例如，如果聯繫人的數據包含一個或多個email地址，就會顯示email應用的圖標。當用戶點擊這個圖標的時候，這個聯繫人所有的email地址都會顯示出來。當用戶點擊其中一個email地址時，email應用將會顯示一個界面，讓用戶爲選中的地址撰寫郵件。

QuickContactBadge視圖提供了對聯繫人數據的即時訪問，是一種與聯繫人溝通的快捷方式。用戶不用查詢一個聯繫人，查找並複製信息，然後把信息粘貼到合適的應用中。他們可以點擊QuickContactBadge，選擇他們想要的溝通方式，然後直接把信息發送給合適的應用中。

## 添加一個QuickContactBadge視圖

爲了添加一個QuickContactBadge視圖，需要在佈局文件中插入一個QuickContactBadge。例如：

```
<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>
```

## 獲取Contacts Provider的數據

爲了能在QuickContactBadge中顯示聯繫人，我們需要這個聯繫人的內容URI和顯示頭像的Bitmap。我們可以從在Contacts Provider中獲取到的數據列中生成這兩個數據。需要指定這些列作爲查詢映射去把數據加載到Cursor中。

對於Android 3.0（API版本爲11）以及以後的版本，需要在查詢映射中添加以下列：

- Contacts.\_ID
- Contacts.LOOKUP\_KEY
- Contacts.PHOTO\_THUMBNAIL\_URI

對於Android 2.3.3（API版本為10）以及之前的版本，則使用以下列：

- `Contacts._ID`
- `Contacts.LOOKUP_KEY`

這一課的剩餘部分假設你已經獲取到了包含這些以及其他你可能選擇的數據列的Cursor對象。想要學習如何獲取這些列對象的Cursor，請參閱課程[獲取聯繫人列表](#)。

## 設置聯繫人URI和縮略圖

一旦我們已經擁有了所需的數據列，那麼我們就可以為QuickContactBadge視圖綁定數據了。

### 設置聯繫人URI

為了設置聯繫人URI，需要調用`getLookupUri(id, lookupKey)`去獲取`CONTENT_LOOKUP_URI`，然後調用`assignContactUri()`去為QuickContactBadge設置對應的聯繫人。例如：

```
// The Cursor that contains contact rows
Cursor mCursor;
// The index of the _ID column in the Cursor
int mIdColumn;
// The index of the LOOKUP_KEY column in the Cursor
int mLookupKeyColumn;
// A content URI for the desired contact
Uri mContactUri;
// A handle to the QuickContactBadge view
QuickContactBadge mBadge;
...
mBadge = (QuickContactBadge) findViewById(R.id.quickbadge);
/*
 * Insert code here to move to the desired cursor row
 */
// Gets the _ID column index
mIdColumn = mCursor.getColumnIndex(Contacts._ID);
// Gets the LOOKUP_KEY index
mLookupKeyColumn = mCursor.getColumnIndex(Contacts.LOOKUP_KEY);
// Gets a content URI for the contact
mContactUri =
    Contacts.getLookupUri(
        mCursor.getLong(mIdColumn),
        mCursor.getString(mLookupKeyColumn)
    );
mBadge.assignContactUri(mContactUri);
```

當用戶點擊QuickContactBadge圖標的時候，這個聯繫人的詳細信息將會自動展現在對話框中。

### 設置聯繫人照片的縮略圖

為QuickContactBadge設置聯繫人URI並不會自動加載聯繫人的縮略圖照片。為了加載聯繫人照片，需要從聯繫人的Cursor對象的一行數據中獲取照片的URI，使用這個URI去打開包含壓縮的縮略圖文件，並把這個文件讀到Bitmap對象中。

Note : `PHOTO_THUMBNAIL_URI`這一列在Android 3.0之前的版本是不存在的。對於這些版本，我們必須從`Contacts.Photo`表中獲取照片的URI。

首先，為包含`Contacts._ID`和`Contacts.LOOKUP_KEY`的Cursor數據列設置對應的變量，這在之前已經有描述：

```

// The column in which to find the thumbnail ID
int mThumbnailColumn;
/*
 * The thumbnail URI, expressed as a String.
 * Contacts Provider stores URIs as String values.
 */
String mThumbnailUri;
...
/*
 * Gets the photo thumbnail column index if
 * platform version >= Honeycomb
 */
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    mThumbnailColumn =
        mCursor.getColumnIndex(Contacts.PHOTO_THUMBNAIL_URI);
// Otherwise, sets the thumbnail column to the _ID column
} else {
    mThumbnailColumn = mIdColumn;
}
/*
 * Assuming the current Cursor position is the contact you want,
 * gets the thumbnail ID
 */
mThumbnailUri = mCursor.getString(mThumbnailColumn);
...

```

定義一個方法，使用與這個聯繫人的照片有關的數據和目標視圖的尺寸作為參數，返回一個尺寸合適的縮略圖Bitmap對象。下面先構建一個指向這個縮略圖的URI：

```

/**
 * Load a contact photo thumbnail and return it as a Bitmap,
 * resizing the image to the provided image dimensions as needed.
 * @param photoData photo ID Prior to Honeycomb, the contact's _ID value.
 * For Honeycomb and later, the value of PHOTO_THUMBNAIL_URI.
 * @return A thumbnail Bitmap, sized to the provided width and height.
 * Returns null if the thumbnail is not found.
 */
private Bitmap loadContactPhotoThumbnail(String photoData) {
    // Creates an asset file descriptor for the thumbnail file.
    AssetFileDescriptor afd = null;
    // try-catch block for file not found
    try {
        // Creates a holder for the URI.
        Uri thumbUri;
        // If Android 3.0 or later
        if (Build.VERSION.SDK_INT
            >=
            Build.VERSION_CODES.HONEYCOMB) {
            // Sets the URI from the incoming PHOTO_THUMBNAIL_URI
            thumbUri = Uri.parse(photoData);
        } else {
            // Prior to Android 3.0, constructs a photo Uri using _ID
            /*
             * Creates a contact URI from the Contacts content URI
             * incoming photoData (_ID)
             */
            final Uri contactUri = Uri.withAppendedPath(
                Contacts.CONTENT_URI, photoData);
            /*
             * Creates a photo URI by appending the content URI of
             * Contacts.Photo.
             */
            thumbUri =
                Uri.withAppendedPath(
                    contactUri, Photo.CONTENT_DIRECTORY);
        }
    /*
     * Retrieves an AssetFileDescriptor object for the thumbnail
    }
}

```

```

    * URI
    * using ContentResolver.openAssetFileDescriptor
    */
    afd = getActivity().getContentResolver();
        openAssetFileDescriptor(thumbUri, "r");
    /*
     * Gets a file descriptor from the asset file descriptor.
     * This object can be used across processes.
     */
    FileDescriptor fileDescriptor = afd.getFileDescriptor();
    // Decode the photo file and return the result as a Bitmap
    // If the file descriptor is valid
    if (fileDescriptor != null) {
        // Decodes the bitmap
        return BitmapFactory.decodeFileDescriptor(
            fileDescriptor, null, null);
    }
    // If the file isn't found
} catch (FileNotFoundException e) {
    /*
     * Handle file not found errors
     */
}
// In all cases, close the asset file descriptor
} finally {
    if (afd != null) {
        try {
            afd.close();
        } catch (IOException e) {}
    }
}
return null;
}

```

在代碼中調用loadContactPhotoThumbnail()去獲取縮略圖Bitmap對象，使用獲取的Bitmap對象去設置QuickContactBadge頭像縮略圖。

```

...
/*
 * Decodes the thumbnail file to a Bitmap.
 */
Bitmap mThumbnail =
    loadContactPhotoThumbnail(mThumbnailUri);
/*
 * Sets the image in the QuickContactBadge
 * QuickContactBadge inherits from ImageView, so
 */
mBadge.setImageBitmap(mThumbnail);

```

## 把QuickContactBadge添加到ListView

QuickContactBadge對於一個展示聯繫人列表的ListView來說是一個非常有用的添加功能。使用QuickContactBadge去為每一個聯繫人顯示一個縮略圖，當用戶點擊這個縮略圖時，QuickContactBadge對話框將會顯示。

### 為ListView添加QuickContactBadge

首先，在列表項佈局文件中添加QuickContactBadge視圖元素。例如，如果我們想為獲取到的每一個聯繫人顯示QuickContactBadge和名字，把以下的XML內容放到對應的佈局文件中：

```

<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>
```

在以下的章節中，這個文件被稱為 `contact_item_layout.xml`。

## 設置自定義的CursorAdapter

定義一個繼承自 `CursorAdapter` 的 `adapter` 來將 `CursorAdapter` 綁定到一個包含 `QuickContactBadge` 的 `ListView` 中。這種方式允許我們在綁定數據到 `QuickContactBadge` 之前對 `Cursor` 中的數據進行處理。同時也能將多個 `Cursor` 中的列綁定到 `QuickContactBadge`。而使用普通的 `CursorAdapter` 是不能完成這些操作的。

我們定義的 `CursorAdapter` 的子類必須重寫以下方法：

- `CursorAdapter.newView()`

填充一個 `View` 對象去持有列表項佈局。在重寫這個方法的過程中，需要保存這個佈局的子 `View` 的 handles，包括 `QuickContactBadge` 的 handles。通過採用這種方法，避免了每次在填充新的佈局時都去獲取子 `View` 的 handles。

我們必須重寫這個方法以便能夠獲取每個子 `View` 對象的 handles。這種方法允許我們控制這些子 `View` 對象在 `CursorAdapter.bindView()` 方法中的綁定。

- `CursorAdapter.bindView()`

將數據從當前 `Cursor` 行綁定到列表項佈局的子 `View` 對象中。必須重寫這個方法以便能夠將聯繫人的 URI 和 縮略圖信息綁定到 `QuickContactBadge`。這個方法的默認實現僅僅允許在數據列和 `View` 之間的一對一映射。

以下的代碼片段是一個包含了自定義 `CursorAdapter` 子類的例子。

## 定義自定義的列表Adapter

定義 `CursorAdapter` 的子類包括編寫這個類的構造方法，以及重寫 `newView()` 和 `bindView()`:

```
private class ContactsAdapter extends CursorAdapter {  
    private LayoutInflater mInflater;  
    ...  
    public ContactsAdapter(Context context) {  
        super(context, null, 0);  
  
        /*  
         * Gets an inflator that can instantiate  
         * the ListView layout from the file.  
         */  
        mInflater = LayoutInflater.from(context);  
        ...  
    }  
    ...  
    /**  
     * Defines a class that hold resource IDs of each item layout  
     * row to prevent having to look them up each time data is
```

```

    * bound to a row.
    */
    private class ViewHolder {
        TextView displayname;
        QuickContactBadge quickcontact;
    }
    ...
@Override
public View newView(
        Context context,
        Cursor cursor,
        ViewGroup viewGroup) {
    /* Inflates the item layout. Stores resource IDs in a
     * in a ViewHolder class to prevent having to look
     * them up each time bindView() is called.
    */
    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);
    ...
    // Sets the display name in the layout
    holder.displayname = cursor.getString(mDisplayNameIndex);
    ...
    /*
     * Generates a contact URI for the QuickContactBadge.
    */
    final Uri contactUri = Contacts.getLookupUri(
        cursor.getLong(mIdIndex),
        cursor.getString(mLookupKeyIndex));
    holder.quickcontact.assignContactUri(contactUri);
    String photoData = cursor.getString(mPhotoDataIndex);
    /*
     * Decodes the thumbnail file to a Bitmap.
     * The method loadContactPhotoThumbnail() is defined
     * in the section "Set the Contact URI and Thumbnail"
    */
    Bitmap thumbnailBitmap =
        loadContactPhotoThumbnail(photoData);
    /*
     * Sets the image in the QuickContactBadge
     * QuickContactBadge inherits from ImageView
    */
    holder.quickcontact.setImageBitmap(thumbnailBitmap);
}

```

## 設置變量

在代碼中，設置相關變量，添加一個包括必須數據列的Cursor。

Note：以下的代碼片段使用了方法 `loadContactPhotoThumbnail()`，這個方法是在[設置聯繫人URI和縮略圖](#)那一節中定義的。

例如：

```
public class ContactsFragment extends Fragment implements
    LoaderManager.LoaderCallbacks<Cursor> {
    ...
    // Defines a ListView
    private ListView mListview;
    // Defines a ContactsAdapter
    private ContactsAdapter mAdapter;
    ...
    // Defines a Cursor to contain the retrieved data
    private Cursor mCursor;
    /*
     * Defines a projection based on platform version. This ensures
     * that you retrieve the correct columns.
     */
    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 :
            /*
             * Although it's not necessary to include the
             * column twice, this keeps the number of
             * columns the same regardless of version
             */
            Contacts_ID
        ...
    };
    /*
     * As a shortcut, defines constants for the
     * column indexes in the Cursor. The index is
     * 0-based and always matches the column order
     * in the projection.
     */
    // Column index of the _ID column
    private int mIdIndex = 0;
    // Column index of the LOOKUP_KEY column
    private int mLookupKeyIndex = 1;
    // Column index of the display name column
    private int mDisplayNameIndex = 3;
    /*
     * Column index of the photo data column.
     * It's PHOTO_THUMBNAIL_URI for Honeycomb and later,
     * and _ID for previous versions.
     */
    private int mPhotoDataIndex =
        Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB ?
            3 :
            0;
    ...
}
```

## 設置ListView

在[Fragment.onCreate\(\)](#)方法中，實例化自定義的adapter對象，獲得一個ListView的handle。

```
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
...
/*
 * Instantiates the subclass of
 * CursorAdapter
 */
ContactsAdapter mContactsAdapter =
    new ContactsAdapter(getActivity());
/*
 * Gets a handle to the ListView in the file
 * contact_list_layout.xml
 */
mListView = (ListView) findViewById(R.layout.contact_list_layout);
...
}
```

在[onActivityCreated\(\)](#)方法中，將ContactsAdapter綁定到ListView。

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    ...
    // Sets up the adapter for the ListView
    mListview.setAdapter(mAdapter);
    ...
}
```

當獲取到一個包含聯繫人數據的Cursor時（通常在[onLoadFinished\(\)](#)的時候），調用[swapCursor\(\)](#)把Cursor中的數據綁定到ListView。這將會為聯繫人列表中的每一項都顯示一個QuickContactBadge。

```
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    // When the loader has completed, swap the cursor into the adapter.
    mContactsAdapter.swapCursor(cursor);
}
```

當我們使用CursorAdapter或其子類中將Cursor中的數據綁定到ListView，並且使用了CursorLoader去加載Cursor數據時，記得要在[onLoaderReset\(\)](#)方法的實現中清理對Cursor對象的引用。例如：

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // Removes remaining reference to the previous Cursor
    mContactsAdapter.swapCursor(null);
}
```

# Android位置信息

編寫:penkzhou - 原文:<http://developer.android.com/training/location/index.html>

位置感知是移動應用一個獨特的功能。用戶去到哪裏都會帶着他們的移動設備，而將位置感知功能添加到我們的應用裏，可以讓用戶有更加真實的情境體驗。位置服務API集成在Google Play服務裏面，這便於我們將自動位置跟蹤、地理圍欄和用戶活動識別等位置感知功能添加到我們的應用當中。

我們喜歡用[Google Play services location APIs](#)勝過Android framework location APIs (`android.location`) 來給我們的應用添加位置感知功能。如果你現在正在使用Android framework location APIs，我們強烈建議你儘可能切換到[Google Play services location APIs](#)。

這個課程介紹如何使用Google Play services location APIs來獲取當前位置、週期性地更新位置以及查找地址。創建並監視地理圍欄以及探測用戶的活動。這個課程包括示例應用和代碼片段，你可以利用這些資源作為添加位置感知到你的應用的基礎。

Note：因為這個課程基於Google Play services client library，所以在使用這些示例應用和代碼段之前確保你安裝了最新版本的Google Play services client library。要想學習如何安裝最新版的client library，請參考[安裝Google Play services嚮導](#)。

## Lessons

- [獲取最後可知位置](#)

學習如何獲取Android設備的最後可知位置。通常Android設備的最後可知位置相當於用戶的當前位置。

- [接收位置更新](#)

學習如何請求和接收週期性的位置更新。

- [顯示位置地址](#)

學習如何將一個位置的經緯度轉化成一個地址（反向地理編碼）。

- [創建和監視地理圍欄](#)

學習如何將一個或多個地理區域定義成一個興趣位置集合，稱為地理圍欄。學習如何探測用戶靠近或者進入地理圍欄事件。

# 獲取最後可知位置

編寫:penkzhou - 原文:<http://developer.android.com/training/location/retrieve-current.html>

使用Google Play services location APIs，我們的應用可以請求獲得用戶設備的最後可知位置。大多數情況下，我們會對用戶的當前位置比較感興趣。而通常用戶的當前位置相當於設備的最後可知位置。

特別地，使用fused location provider來獲取設備的最後可知位置。fused location provider是Google Play services location APIs中的一個。它處理基本定位技術並提供一個簡單的API，使得我們可以指定高水平的需求，如高精度或者低功耗。同時它優化了設備的耗電情況。

這節課介紹如何通過使用fused location provider的[getLastLocation\(\)](#)方法為設備的位置構造一個單一請求。

## 安裝Google Play Services

為了訪問fused location provider，我們的應用開發工程必須包括Google Play services。通過[SDK Manager](#)下載和安裝Google Play services組件，添加相關的庫到我們的工程。更詳細的介紹，請看[Setting Up Google Play Services](#)。

## 確定應用的權限

使用位置服務的應用必須請求用戶位置權限。Android擁有兩種位置權限：[ACCESS\\_COARSE\\_LOCATION](#) 和 [ACCESS\\_FINE\\_LOCATION](#)。我們選擇的權限決定API返回的位置信息的精度。如果我們選擇了[ACCESS\\_COARSE\\_LOCATION](#)，API返回的位置信息的精確度大體相當於一個城市街區。

這節課只要求粗略的定位。在我們應用的manifest文件中，用 `uses-permission` 節點請求這個權限，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.basiclocationsample" >
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
</manifest>
```

## 連接Google Play Services

為了連接到API，我們需要創建一個Google Play services API客戶端實例。關於使用這個客戶端的更詳細的介紹，請看[Accessing Google APIs](#)。

在我們的activity的[onCreate\(\)](#)方法中，用[GoogleApiClient.Builder](#)創建一個Google API Client實例。使用這個builder添加[LocationServices](#) API。

實例應用定義了一個 `buildGoogleApiClient()` 方法，這個方法在activity的[onCreate\(\)](#)方法中被調用。`buildGoogleApiClient()` 方法包括下面的代碼。

```
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
```

```
}
```

## 獲取最後可知位置

一旦我們將Google Play services和location services API連接完成後，我們就可以獲取用戶設備的最後可知位置。當我們的應用連接到這些服務之後，我們可以用fused location provider的`getLastLocation()`方法來獲取設備的位置。調用這個方法返回的定位精確度是由我們在應用的manifest文件裏添加的權限決定的，如本文的確定應用的權限部分描述的內容一樣。

為了請求最後可知位置，調用`getLastLocation()`方法，並將我們創建的`GoogleApiClient`對象的實例傳給該方法。在Google API Client提供的`onConnected()`回調函數裏調用`getLastLocation()`方法，這個回調函數在client準備好的時候被調用。下面的示例代碼說明了請求和一個對響應簡單的處理：

```
public class MainActivity extends ActionBarActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
    @Override
    public void onConnected(Bundle connectionHint) {
        mLastLocation = LocationServices.FusedLocationApi.getLastLocation(
                mGoogleApiClient);
        if (mLastLocation != null) {
            mLatitudeText.setText(String.valueOf(mLastLocation.getLatitude()));
            mLongitudeText.setText(String.valueOf(mLastLocation.getLongitude()));
        }
    }
}
```

`getLastLocation()`方法返回一個`Location`對象。通過`Location`對象，我們可以取得地理位置的經度和緯度座標。在少數情況下，當位置不可用時，這個`Location`對象會返回null。

下一課，[獲取位置更新](#)，教你如何週期性地獲取位置信息更新。

# 獲取位置更新

編寫:penkzhou - 原文:<http://developer.android.com/training/location/receive-location-updates.html>

如果我們的應用可以週期性地跟蹤位置，那麼應用可以給用戶提供更多相關信息。例如，如果我們的應用在用戶行走或者駕車時幫助找到他們的路，或者如果我們的應用跟蹤用戶的位置，那麼它需要定期獲取設備的位置。除了地理位置之外（經度和緯度），我們可能還想為用戶提供更多的信息，例如方位（行駛的水平方向）、海拔或者設備的速度。這些信息可以在 [Location](#) 對象中獲得，我們的應用可以從 [fused location provider](#) 中得到這個對象。

當我們用 [getLastLocation\(\)](#) 獲取設備的位置時，如上一節課[獲取最後可知位置](#)介紹的一樣，一個更加直接的方法是從 fused location provider 中請求週期性的更新。作為迴應，API根據現有的位置供應源，如Wifi和GPS（Global Positioning System），用最佳位置週期地更新我們的應用。這些providers、我們請求的權限和我們在位置請求中設置的選項決定了位置的精確度。

這節課介紹如何用fused location provider的[requestLocationUpdates\(\)](#)方法來請求定期更新設備的位置。

## 連接Location Services

應用的 Location Services 由 Google Play services 和 fused location provider 提供。為了用這些服務，用 Google API Client 連接到我們的應用，然後請求位置更新。用 [GoogleApiClient](#) 進行連接的詳細步驟請見[獲取最後可知位置](#)，包括了請求當前位置。

設備的最後可知位置提供有關起點的基準信息，在開始定期更新位置信息前，保證應用擁有一個可知的位置。[獲取最後可知位置](#)介紹瞭如何通過調用 [getLastLocation\(\)](#) 獲取最後可知位置。接下來的內容假設我們的應用已經取得最後可知位置，並已將最後可知位置作為一個 [Location](#) 對象保存在全局變量 `mCurrentLocation` 中。

使用位置服務的應用必須請求位置權限。在這節課中我們需要很好的定位檢測，使得我們的應用可以從可用的位置供應源得到儘可能精確的位置數據。在我們應用的manifest文件中，用 `uses-permission` 節點請求位置權限，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.sample.locationupdates" >

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

## 設置位置請求

創建一個 [LocationRequest](#) 以保存請求 fused location provider 的參數。這些參數決定了請求精確度的水平。對於位置請求中所有可用的選項，請見 [LocationRequest](#) 類的參考文檔。這節課設置更新間隔、最快更新間隔和優先級。如下所述：

### 更新間隔

[setInterval\(\)](#) - 這個方法設置應用接收位置更新的速率（每毫秒）。注意如果另一個應用正在接收一個更快的或者更慢的更新速率，又或者根本沒有更新（例如，設備還沒有連接），那麼我們應用的位置更新速率可能會比 `setInterval()` 設置的速率更快。

### 最快更新間隔

`setFastestInterval()` - 這個方法設置應用可以處理位置更新的最快速率（每毫秒）。因為其它應用會影響到已發送出去的位置更新的速率，所以我們需要設置這個最快速率。Google Play services location APIs 發送任何應用用 `setInterval()` 請求的最快的更新速率。如果這個速率比我們的應用可以處理的速率還要快，那麼我們可能會遇到UI閃爍或者數據溢出等問題。為了避免這個問題，調用 `setFastestInterval()` 限制更新速率的上限。

## 優先級

`setPriority()` - 這個方法設置請求的優先級，為 Google Play services 位置服務提供了關於使用哪個位置源的強烈的暗示。支持下面幾個值：

- `PRIORITY_BALANCED_POWER_ACCURACY` - 這個設置請求一個城市街區範圍的位置精確度（精確度約為 100米）。這被認為是一個粗略的精確度，也可能是耗電較小的設置。對於這個設置，位置服務可能使用 WiFi 和 基站進行定位。注意，無論如何，位置供應源的選擇依賴於很多其它的因素。
- `PRIORITY_HIGH_ACCURACY` - 這個設置請求最高精度的位置信息。對於這個設置，位置服務更可能使用 GPS(Global Positioning System) 來定位。
- `PRIORITY_LOW_POWER` - 這個設置請求一個城市範圍的精確度（精確度約為 10公里）。這被認為是一個粗略的精確度，也可能是耗電較小的設置。
- `PRIORITY_NO_POWER` - 如果需要對功率消耗的影響微乎其微，但又想在可用的時候接收位置更新，那麼使用這個設置。對於這個設置，我們的應用不會觸發任何位置更新，但是會接收由其它應用觸發的位置。

下面的示例介紹創建位置請求和設置相關的參數：

```
protected void createLocationRequest() {  
    LocationRequest mLocationRequest = new LocationRequest();  
    mLocationRequest.setInterval(10000);  
    mLocationRequest.setFastestInterval(5000);  
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);  
}
```

`PRIORITY_HIGH_ACCURACY` 的優先級聯合在我們應用的 manifest 文件中定義的 `ACCESS_FINE_LOCATION` 權限和一個5000毫秒（5秒）的更新間隔。該優先級使 fused location provider 返回精確到幾英尺之內的位置更新。這個方法適用於需要實時顯示位置的地圖應用。

**性能提示：**如果我們的應用在接收一個位置更新後接入網絡或者執行持續時間長的工作，那麼將最快更新間隔調整到一個更慢的值。這個調整防止我們的應用接收不可用的更新。一旦持續時間長的工作完成，將最快更新間隔改回一個快的值。

## 請求位置更新

我們已經設置了包含應用位置更新要求的位置請求，我們可以調用 `requestLocationUpdates()` 來啓動週期性的更新。在 Google API Client 提供的 `onConnected()` 回調函數（當 client 準備好之後會調用這個回調函數）中啓動週期性更新。

根據請求的形式，fused location provider 要麼調用 `LocationListener.onLocationChanged()` 回調函數並傳遞一個 `Location` 對象，要麼發出一個將位置信息包含在擴展數據的 `PendingIntent`。更新的精確度和頻率受已請求的位置權限和在位置請求對象中設置的選項等因素影響。

這節課介紹如何使用 `LocationListener` 回調函數獲取位置更新。調用 `requestLocationUpdates()`，並傳入 `GoogleApiClient` 的實例、`LocationRequest` 對象和一個 `LocationListener`。定義一個 `startLocationUpdates()` 方法，

該方法在 `onConnected()` 回調函數被調用，如下面的示例代碼所示：

```
@Override  
public void onConnected(Bundle connectionHint) {  
    ...  
    if (mRequestingLocationUpdates) {  
        startLocationUpdates();  
    }  
}  
  
protected void startLocationUpdates() {  
    LocationServices.FusedLocationApi.requestLocationUpdates(  
        mGoogleApiClient, mLocationRequest, this);  
}
```

注意到上述的代碼片段提到一個布爾標誌位，`mRequestingLocationUpdates`，該標誌位用於判斷用戶將位置更新打開還是關閉。關於這個標誌位更詳細的介紹，請見下面的[保存 Activity](#) 的狀態的內容。

## 定義位置更新回調函數

fused location provider 調用 `LocationListener.onLocationChanged()` 回調函數。這個回調函數傳入的參數是一個含有位置經緯度的 `Location` 對象。下面的代碼介紹瞭如何實現 `LocationListener` 接口和定義方法，然後獲取位置更新的時間戳並在應用用戶界面上顯示經度、緯度和時間戳：

```
public class MainActivity extends ActionBarActivity implements  
    ConnectionCallbacks, OnConnectionFailedListener, LocationListener {  
    ...  
    @Override  
    public void onLocationChanged(Location location) {  
        mCurrentLocation = location;  
        mLastUpdateTime = DateFormat.getTimeInstance().format(new Date());  
        updateUI();  
    }  
  
    private void updateUI() {  
        mLatitudeTextView.setText(String.valueOf(mCurrentLocation.getLatitude()));  
        mLongitudeTextView.setText(String.valueOf(mCurrentLocation.getLongitude()));  
        mLastUpdateTimeTextView.setText(mLastUpdateTime);  
    }  
}
```

## 停止位置更新

我們需要考慮當 `activity` 不在焦點上時我們是否需要停止位置更新，例如，當用戶切換到另一個應用或者同一個應用的不同 `activity` 的情況。假如應用即使在後臺運行時也不需要收集用戶數據，將會有利於降低功耗。這節課會介紹如何在 `activity` 的 `onPause()` 方法裏停止位置更新。

為了停止位置更新，調用 `removeLocationUpdates()`，並傳入 `GoogleApiClient` 對象的實例和一個 `LocationListener`，如下面的示例代碼所示：

```
@Override  
protected void onPause() {  
    super.onPause();  
    stopLocationUpdates();  
}  
  
protected void stopLocationUpdates() {
```

```
        LocationServices.FusedLocationApi.removeLocationUpdates(
            mGoogleApiClient, this);
    }
```

使用一個布爾值，`mRequestingLocationUpdates`，來判斷當前位置更新是否打開。在 `activity` 的 `onResume()` 方法裏，檢查當前的位置更新是否起作用。如果位置更新不起作用，那麼激活它：

```
@Override
public void onResume() {
    super.onResume();
    if (mGoogleApiClient.isConnected() && !mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
```

## 保存 Activity 的狀態

一個設備配置的變動，如旋轉屏幕或者改變語言，可以導致當前的 `activity` 崩潰。我們的應用必須保存任何在重新創建 `activity` 時需要用到的信息。一種方法是通過一個保存在 `Bundle` 對象的實例狀態來解決這個問題。

下面的示例代碼介紹瞭如何用 `activity` 的 `onSaveInstanceState()` 回調函數來保存實例狀態：

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putBoolean(REQUESTING_LOCATION_UPDATES_KEY,
        mRequestingLocationUpdates);
    savedInstanceState.putParcelable(LOCATION_KEY, mCurrentLocation);
    savedInstanceState.putString(LAST_UPDATED_TIME_STRING_KEY, mLastUpdateTime);
    super.onSaveInstanceState(savedInstanceState);
}
```

定義一個 `updateValuesFromBundle()` 方法來恢復保存在 `activity` 的上一個實例的值（如果這些值可用的話）。在 `onCreate()` 中調用這個方法。如下所示：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    updateValuesFromBundle(savedInstanceState);
}

private void updateValuesFromBundle(Bundle savedInstanceState) {
    if (savedInstanceState != null) {
        // Update the value of mRequestingLocationUpdates from the Bundle, and
        // make sure that the Start Updates and Stop Updates buttons are
        // correctly enabled or disabled.
        if (savedInstanceState.keySet().contains(REQUESTING_LOCATION_UPDATES_KEY)) {
            mRequestingLocationUpdates = savedInstanceState.getBoolean(
                REQUESTING_LOCATION_UPDATES_KEY);
            setButtonsEnabledState();
        }

        // Update the value of mCurrentLocation from the Bundle and update the
        // UI to show the correct latitude and longitude.
        if (savedInstanceState.keySet().contains(LOCATION_KEY)) {
            // Since LOCATION_KEY was found in the Bundle, we can be sure that
            // mCurrentLocationis not null.
            mCurrentLocation = savedInstanceState.getParcelable(LOCATION_KEY);
        }

        // Update the value of mLastUpdateTime from the Bundle and update the UI.
        if (savedInstanceState.keySet().contains(LAST_UPDATED_TIME_STRING_KEY)) {
```

```
        mLastUpdateTime = savedInstanceState.getString(
            LAST_UPDATED_TIME_STRING_KEY);
    }
    updateUI();
}
}
```

更多關於保存實例狀態的內容，請看 [Android Activity](#) 類的參考文檔。

Note：為了可以更加持久地存儲，我們可以將用戶的偏好設定保存在應用的 [SharedPreferences](#) 中。在 `activity` 的 `onPause()` 方法中設置偏好設定，在 `onResume()` 中獲取這些設定。更多關於偏好設定的內容，請見[保存到 Reference](#)。

下一節課，[顯示位置地址](#)，介紹如何顯示指定位置的街道地址。

# 顯示位置地址

編寫:penkzhou - 原文:<http://developer.android.com/training/location/display-address.html>

獲取最後可知位置和獲取位置更新課程描述瞭如何以一個Location對象的形式獲取用戶的位置信息，這個位置信息包括了經緯度。儘管經緯度對計算地理距離和在地圖上顯示位置很有用，但是更多情況下位置的地址更有用。例如，如果我們想讓用戶知道他們在哪裏，那麼一個街道地址比地理座標（經度/緯度）更加有意義。

使用 Android 框架位置 APIs 的 Geocoder 類，我們可以將地址轉換成相應的地理座標。這個過程叫做地理編碼。或者，我們可以將地理位置轉換成相應的地址。這種地址查找功能叫做反向地理編碼。

這節課介紹瞭如何用 `getFromLocation()` 方法將地理位置轉換成地址。這個方法返回與制定經緯度相對應的估計的街道地址。

## 獲取地理位置

設備的最後可知位置對於地址查找功能是很有用的基礎。獲取最後可知位置介紹瞭如何通過調用 fused location provider 提供的 `getLastLocation()` 方法找到設備的最後可知位置。

為了訪問 fused location provider，我們需要創建一個 Google Play services API client 的實例。關於如何連接 client，請見[連接 Google Play Services](#)。

為了讓 fused location provider 得到一個準確的街道地址，在應用的 manifest 文件添加位置權限 `ACCESS_FINE_LOCATION`，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.locationupdates" >

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

## 定義一個 Intent 服務來取得地址

Geocoder 類的 `getFromLocation()` 方法接收一個經度和緯度，返回一個地址列表。這個方法是同步的，可能會花很長時間來完成它的工作，所以我們不應該在應用的主線程和 UI 線程裏調用這個方法。

IntentService 類提供了一種結構使一個任務在後臺線程運行。使用這個類，我們可以在不影響 UI 韻應速度的情況下處理一個長時間運行的操作。注意到，`AsyncTask` 類也可以執行後臺操作，但是它被設計用於短時間運行的操作。在 `activity` 重新創建時（例如當設備旋轉時），`AsyncTask` 不應該保存 UI 的引用。相反，當 `activity` 重建時，不需要取消 `IntentService`。

定義一個繼承 `IntentService` 的類 `FetchAddressIntentService`。這個類是地址查找服務。這個 Intent 服務在一個工作線程上異步地處理一個 intent，並在它離開這個工作時自動停止。Intent 外加的數據提供了服務需要的數據，包括一個用於轉換成地址的 Location 對象和一個用於處理地址查找結果的 ResultReceiver 對象。這個服務用一個 Geocoder 來獲取位置的地址，並且將結果發送給 ResultReceiver。

### 在應用的 manifest 文件中定義 Intent 服務

在 manifest 文件中添加一個節點以定義 intent 服務：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.locationaddress" >
    <application
        ...
        <service
            android:name=".FetchAddressIntentService"
            android:exported="false"/>
    </application>
    ...
</manifest>
```

Note : manifest 文件裏的 `<service>` 節點不需要包含一個 intent filter，這是因為我們的主 `activity` 通過指定 intent 用到的類的名字來創建一個隱式的 intent。

## 創建一個 Geocoder

將一個地理位置傳換成地址的過程叫做反向地理編碼。通過實現 `FetchAddressIntentService` 類的 `onHandleIntent()` 來執行 intent 服務的主要工作，即反向地理編碼請求。創建一個 `Geocoder` 對象來處理反向地理編碼。

一個區域設置代表一個特定的地理上的或者語言上的區域。`Locale` 對象用於調整信息的呈現方式，例如數字或者日期，來適應區域設置表示的區域的約定。傳一個 `Locale` 對象到 `Geocoder` 對象，確保地址結果為用戶的地理區域作出了本地化。

```
@Override
protected void onHandleIntent(Intent intent) {
    Geocoder geocoder = new Geocoder(this, Locale.getDefault());
    ...
}
```

## 獲取街道地址數據

下一步是從 `geocoder` 獲取街道地址，處理可能出現的錯誤，和將結果返回給請求地址的 `activity`。我們需要兩個分別代表成功和失敗的數字常量來報告地理編碼過程的結果。定義一個 `constants` 類來包含這些值，如下所示：

```
public final class Constants {
    public static final int SUCCESS_RESULT = 0;
    public static final int FAILURE_RESULT = 1;
    public static final String PACKAGE_NAME =
        "com.google.android.gms.location.sample.locationaddress";
    public static final String RECEIVER = PACKAGE_NAME + ".RECEIVER";
    public static final String RESULT_DATA_KEY = PACKAGE_NAME +
        ".RESULT_DATA_KEY";
    public static final String LOCATION_DATA_EXTRA = PACKAGE_NAME +
        ".LOCATION_DATA_EXTRA";
}
```

為了獲取與地理位置相對應的街道地址，調用 `getFromLocation()`，傳入位置對象的經度和緯度，以及我們想要返回的地址的最大數量。在這種情況下，我們只需要一個地址。`geocoder` 返回一個地址數組。如果沒有找到指定位置的地址，那麼它會返回空的列表。如果沒有可用的後臺地理編碼服務，`geocoder` 會返回 null。

如下面代碼介紹來檢查下述這些錯誤。如果出現錯誤，就將相應的錯誤信息傳給變量 `errorMessage`，從而將錯誤信息發送給發出請求的 `activity`：

- No location data provided - Intent 的附加數據沒有包含反向地理編碼需要用到的 `Location` 對象。

- Invalid latitude or longitude used - [Location](#) 對象提供的緯度和/或者經度無效。
- No geocoder available - 由於網絡錯誤或者 IO 異常，導致後臺地理編碼服務不可用。
- Sorry, no address found - geocoder 找不到指定緯度/經度對應的地址。

使用 [Address](#) 類中的 [getAddressLine\(\)](#) 方法來獲得地址對象的個別行。然後將這些行加入一個地址 fragment 列表當中。其中，這個地址 fragment 列表準備好返回到發出地址請求的 [activity](#)。

為了將結果返回給發出地址請求的 [activity](#)，需要調用 [deliverResultToReceiver\(\)](#) 方法（定義於下面的[把地址返回給請求端](#)）。結果由之前提到的成功/失敗數字代碼和一個字符串組成。在反向地理編碼成功的情況下，這個字符串包含着地址。在失敗的情況下，這個字符串包含錯誤的信息。如下所示：

```

@Override
protected void onHandleIntent(Intent intent) {
    String errorMessage = "";

    // Get the location passed to this service through an extra.
    Location location = intent.getParcelableExtra(
        Constants.LOCATION_DATA_EXTRA);

    ...

    List<Address> addresses = null;

    try {
        addresses = geocoder.getFromLocation(
            location.getLatitude(),
            location.getLongitude(),
            // In this sample, get just a single address.
            1);
    } catch (IOException ioException) {
        // Catch network or other I/O problems.
        errorMessage = getString(R.string.service_not_available);
        Log.e(TAG, errorMessage, ioException);
    } catch (IllegalArgumentException illegalArgumentException) {
        // Catch invalid latitude or longitude values.
        errorMessage = getString(R.string.invalid_lat_long_used);
        Log.e(TAG, errorMessage + ". " +
            "Latitude = " + location.getLatitude() +
            ", Longitude = " +
            location.getLongitude(), illegalArgumentException);
    }

    // Handle case where no address was found.
    if (addresses == null || addresses.size() == 0) {
        if (errorMessage.isEmpty()) {
            errorMessage = getString(R.string.no_address_found);
            Log.e(TAG, errorMessage);
        }
        deliverResultToReceiver(Constants.FAILURE_RESULT, errorMessage);
    } else {
        Address address = addresses.get(0);
        ArrayList<String> addressFragments = new ArrayList<String>();

        // Fetch the address lines using getAddressLine,
        // join them, and send them to the thread.
        for(int i = 0; i < address.getMaxAddressLineIndex(); i++) {
            addressFragments.add(address.getAddressLine(i));
        }
        Log.i(TAG, getString(R.string.address_found));
        deliverResultToReceiver(Constants.SUCCESS_RESULT,
            TextUtils.join(System.getProperty("line.separator"),
                addressFragments));
    }
}

```

## 把地址返回給請求端

Intent 服務最後要做的事情是將地址返回給啓動服務的 `activity` 裏的 `ResultReceiver`。這個 `ResultReceiver` 類允許我們發送一個帶有結果的數字代碼和一個包含結果數據的消息。這個數字代碼說明了地理編碼請求是成功還是失敗。在反向地理編碼成功的情況下，這個消息包含着地址。在失敗的情況下，這個消息包含一些描述失敗原因的文本。

我們已經可以從 `geocoder` 取得地址，捕獲到可能出現的錯誤，調用 `deliverResultToReceiver()` 方法。現在我們需要定義 `deliverResultToReceiver()` 方法來將結果代碼和消息包發送給結果接收端。

對於結果代碼，使用已經傳給 `deliverResultToReceiver()` 方法的 `resultCode` 參數的值。對於消息包的結構，連接 `Constants` 類的 `RESULT_DATA_KEY` 常量（定義與 [獲取街道地址數據](#)）和傳給 `deliverResultToReceiver()` 方法的 `message` 參數的值。如下所示：

```
public class FetchAddressIntentService extends IntentService {
    protected ResultReceiver mReceiver;
    ...
    private void deliverResultToReceiver(int resultCode, String message) {
        Bundle bundle = new Bundle();
        bundle.putString(Constants.RESULT_DATA_KEY, message);
        mReceiver.send(resultCode, bundle);
    }
}
```

## 啓動 Intent 服務

上節課定義的 `intent` 服務在後臺運行，同時，該服務負責提取與指定地理位置相對應的地址。當我們啓動服務，Android 框架會實例化並啓動服務（如果該服務沒有運行），並且如果需要的話，創建一個進程。如果服務正在運行，那麼讓它保持運行狀態。因為服務繼承於 `IntentService`，所以當所有 `intent` 都被處理完之後，該服務會自動停止。

在我們應用的主 `activity` 中啓動服務，並且創建一個 `Intent` 來把數據傳給服務。我們需要創建一個顯式的 `intent`，這是因為我們只想我們的服務響應該 `intent`。詳細請見 [Intent Types](#)。

為了創建一個顯式的 `intent`，需要為服務指定要用到的類名：`FetchAddressIntentService.class`。在 `intent` 附加數據中傳入兩個信息：

- 一個用於處理地址查找結果的 `ResultReceiver`。
- 一個包含想要轉換成地址的緯度和經度的 `Location` 對象。

下面的代碼介紹瞭如何啓動 `intent` 服務：

```
public class MainActivity extends ActionBarActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {

    protected Location mLastLocation;
    private AddressResultReceiver mResultReceiver;
    ...

    protected void startIntentService() {
        Intent intent = new Intent(this, FetchAddressIntentService.class);
        intent.putExtra(Constants.RECEIVER, mResultReceiver);
        intent.putExtra(Constants.LOCATION_DATA_EXTRA, mLastLocation);
        startService(intent);
    }
}
```

當用戶請求查找地理地址時，調用上述的 `startIntentService()` 方法。例如，用戶可能會在我們應用的 UI 上麪點擊提

取地址按鈕。在啓動 intent 服務之前，我們需要檢查是否已經連接到 Google Play services。下面的代碼片段介紹在一個按鈕 handler 中調用 `startIntentService()` 方法。

```
public void fetchAddressButtonHandler(View view) {
    // Only start the service to fetch the address if GoogleApiClient is
    // connected.
    if (mGoogleApiClient.isConnected() && mLastLocation != null) {
        startIntentService();
    }
    // If GoogleApiClient isn't connected, process the user's request by
    // setting mAddressRequested to true. Later, when GoogleApiClient connects,
    // launch the service to fetch the address. As far as the user is
    // concerned, pressing the Fetch Address button
    // immediately kicks off the process of getting the address.
    mAddressRequested = true;
    updateUIWidgets();
}
```

如果用戶點擊了應用 UI 上面的提取地址按鈕，那麼我們必須在 Google Play services 連接穩定之後啓動 intent 服務。下面的代碼片段介紹了調用 Google API Client 提供的 `onConnected()` 回調函數中的 `startIntentService()` 方法。

```
public class MainActivity extends ActionBarActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
    @Override
    public void onConnected(Bundle connectionHint) {
        // Gets the best and most recent location currently available,
        // which may be null in rare cases when a location is not available.
        mLastLocation = LocationServices.FusedLocationApi.getLastLocation(
            mGoogleApiClient);

        if (mLastLocation != null) {
            // Determine whether a Geocoder is available.
            if (!Geocoder.isPresent()) {
                Toast.makeText(this, R.string.no_geocoder_available,
                    Toast.LENGTH_LONG).show();
                return;
            }

            if (mAddressRequested) {
                startIntentService();
            }
        }
    }
}
```

## 獲取地理編碼結果

Intent 服務已經處理完地理編碼請求，並用 `ResultReceiver` 將結果返回給發出請求的 `activity`。在發出請求的 `activity` 裏，定義一個繼承於 `ResultReceiver` 的 `AddressResultReceiver`，用於處理在 `FetchAddressIntentService` 中的響應。

結果包含一個數字代碼（`resultCode`）和一個包含結果數據（`resultData`）的消息。如果反向地理編碼成功的話，`resultData` 會包含地址。如果失敗，`resultData` 包含描述失敗原因的文本。關於錯誤信息更詳細的內容，請見[把地址返回給請求端](#)

重寫 `onReceiveResult()` 方法來處理發送給接收端的結果，如下所示：

```
public class MainActivity extends ActionBarActivity implements
    ConnectionCallbacks, OnConnectionFailedListener {
    ...
}
```

```
class AddressResultReceiver extends ResultReceiver {
    public AddressResultReceiver(Handler handler) {
        super(handler);
    }

    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {

        // Display the address string
        // or an error message sent from the intent service.
        mAddressOutput = resultData.getString(Constants.RESULT_DATA_KEY);
        displayAddressOutput();

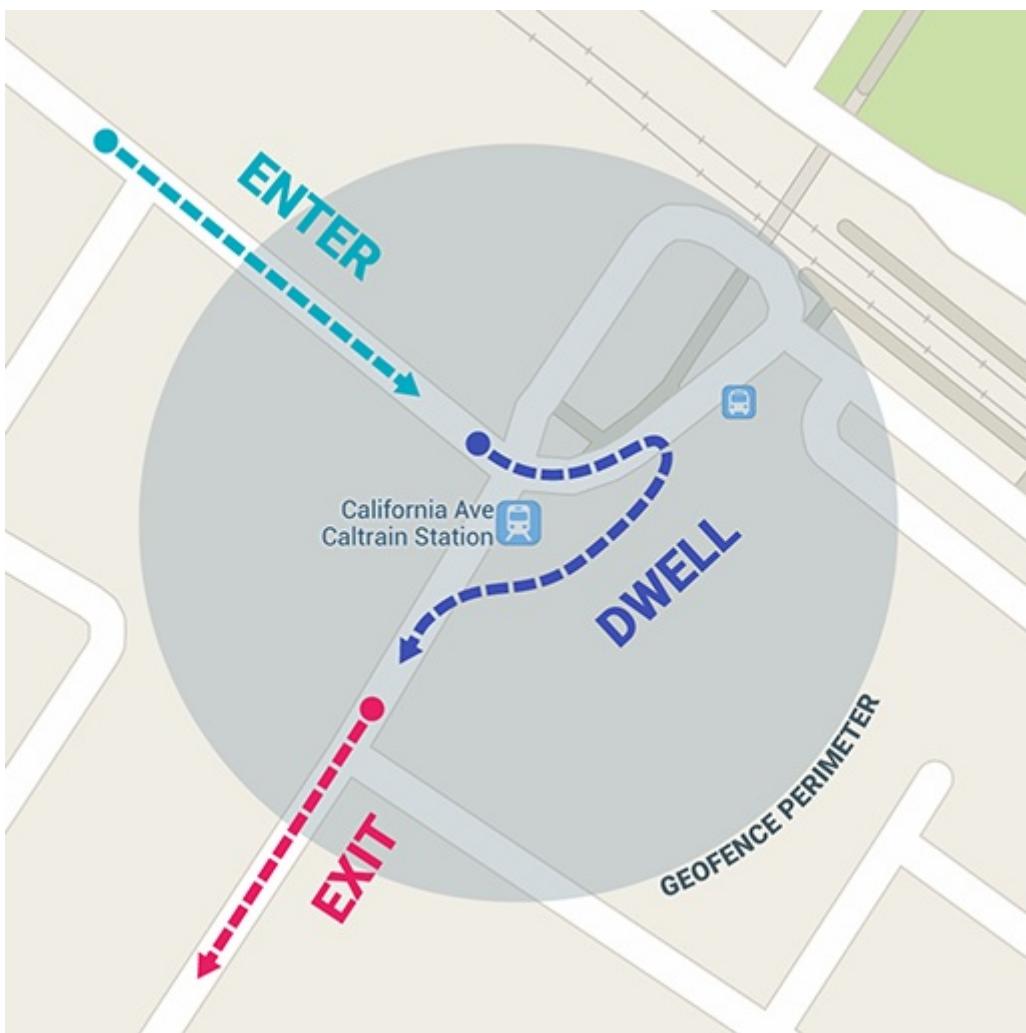
        // Show a toast message if an address was found.
        if (resultCode == Constants.SUCCESS_RESULT) {
            showToast(getString(R.string.address_found));
        }
    }
}
```

# 創建和監視地理圍欄

編寫:penkzhou - 原文:<http://developer.android.com/training/location/geofencing.html>

地理圍欄將用戶當前位置感知和附件地點特徵感知相結合。為了標示一個感興趣的位置，我們需要指定這個位置的經緯度。為了調整位置的鄰近度，需要添加一個半徑。經緯度和半徑定義一個地理圍欄，即在感興趣的位置創建一個圓形區域或者圍欄。

我們可以有多個活動的地理圍欄（限制是一個設備用戶100個）。對於每個地理圍欄，我們可以讓 Location Services 發出進入和離開事件，或者我們可以在觸發一個事件之前，指定在某個地理圍欄區域等待一段時間或者停留。通過指定一個以毫秒為單位的截止時間，我們可以限制任何一個地理圍欄的持續時間。當地理圍欄失效後，Location Services 會自動刪除這個地理圍欄。



這節課介紹如何添加和刪除地理圍欄，和用 IntentService 監聽地理位置變化。

## 設置地理圍欄監視

請求地理圍欄監視的第一步就是設置必要的權限。在使用地理圍欄時，我們必須設置 ACCESS\_FINE\_LOCATION 權限。在應用的 manifest 文件中添加如下子節點即可：

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

如果想要用 [IntentService](#) 監聽地理位置變化，那麼還需要添加一個節點來指定服務名字。這個節點必須是 的子節點：

```
<application
    android:allowBackup="true">
    ...
    <service android:name=".GeofenceTransitionsIntentService"/>
</application>
```

為了訪問位置 API，我們需要創建一個 Google Play services API client 的實例。想要學習如何連接 client，請見[連接 Google Play Services](#)。

## 創建和添加地理圍欄

我們的應用需要用位置 API 的 builder 類來創建地理圍欄，用 convenience 類來添加地理圍欄。另外，我們可以定義一個 [PendingIntent](#)（將在這節課介紹）來處理當地理位置發生遷移時，Location Services 發出的 intent。

### 創建地理圍欄對象

首先，用 [Geofence.Builder](#) 創建一個地理圍欄，設置想要的半徑，持續時間，和地理圍欄遷移的類型。例如，填充一個叫做 `mGeofenceList` 的 list 對象：

```
mGeofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
        Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```

這個例子從一個固定的文件中獲取數據。在實際情況下，應用可能會根據用戶的位置動態地創建地理圍欄。

### 指定地理圍欄和初始化觸發器

下面的代碼用到 [GeofencingRequest](#) 類。該類嵌套了 [GeofencingRequestBuilder](#) 類來需要監視的地理圍欄和設置如何觸發地理圍欄事件：

```
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(mGeofenceList);
    return builder.build();
}
```

這個例子介紹了兩個地理圍欄觸發器。當設備進入一個地理圍欄時， [GEOFENCE\\_TRANSITION\\_ENTER](#) 轉移會觸

發。當設備離開一個地理圍欄時，`GEOFENCE_TRANSITION_EXIT` 轉移會觸發。如果設備已經在地理圍欄裏面，那麼指定 `INITIAL_TRIGGER_ENTER` 來通知位置服務觸發 `GEOFENCE_TRANSITION_ENTER`。

在很多情況下，使用 `INITIAL_TRIGGER_DWELL` 可能會更好。僅僅當由於到達地理圍欄中已定義好的持續時間，而導致用戶停止時，`INITIAL_TRIGGER_DWELL` 纔會觸發事件。這個方法可以減少當設備短暫地進入和離開地理圍欄時，由大量的通知造成的“垃圾警告信息”。另一種獲取最好的地理圍欄結果的策略是設置最小半徑為100米。這有助於估計典型的 Wifi 網絡的位置精確度，也有利於降低設備的功耗。

## 為地理圍欄轉移定義Intent

從 Location Services 發送來的Intent能夠觸發各種應用內的動作，但是不能用它來打開一個 `Activity` 或者 `Fragment`，這是因為應用內的組件只能在響應用戶動作時纔可見。大多數情況下，處理這一類 Intent 最好使用 `IntentService`。一個 `IntentService` 可以推送一個通知，可以進行長時間的後臺作業，可以將 intent 發送給其他的 services，還可以發送一個廣播 intent。下面的代碼展示瞭如何定義一個 `PendingIntent` 來啓動一個 `IntentService`:

```
public class MainActivity extends FragmentActivity {  
    ...  
  
    private PendingIntent getGeofencePendingIntent() {  
        // Reuse the PendingIntent if we already have it.  
        if (mGeofencePendingIntent != null) {  
            return mGeofencePendingIntent;  
        }  
        Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);  
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when  
        // calling addGeofences() and removeGeofences().  
        return PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);  
    }  
}
```

## 添加地理圍欄

使用 `GeoencingApi.addGeofences()` 方法來添加地理圍欄。為該方法提供 Google API client，`GeofencingRequest` 對象和 `PendingIntent`。下面的代碼，在 `onResult()` 中處理結果，假設主 `activity` 實現 `ResultCallback`。

```
public class MainActivity extends FragmentActivity {  
    ...  
  
    LocationServices.GeofencingApi.addGeofences(  
        mGoogleApiClient,  
        getGeofencingRequest(),  
        getGeofencePendingIntent()  
    ).setResultCallback(this);  
}
```

## 處理地理圍欄轉移

當 Location Services 探測到用戶進入或者離開一個地理圍欄，它會發送一個包含在 `PendingIntent` 的 Intent，這個 `PendingIntent` 就是在添加地理圍欄時被我們包括在請求當中。這個 Intent 被一個類似 `GeofenceTransitionsIntentService` 的 service 接收，這個 service 從 intent 得到地理圍欄事件，決定地理圍欄轉移的類型，和決定觸發哪個已定義的地理圍欄。然後它會發出一個通知。

下面的代碼介紹瞭如何定義一個 `IntentService`。這個 `IntentService` 在地理圍欄轉移出現時，會推送一個通知。當用戶點擊這個通知，那麼應用的主 `activity` 會出現：

```
public class GeofenceTransitionsIntentService extends IntentService {  
    ...
```

```

protected void onHandleIntent(Intent intent) {
    GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
    if (geofencingEvent.hasError()) {
        String errorMessage = GeofenceErrorMessages.getErrorString(this,
            geofencingEvent.getErrorCode());
        Log.e(TAG, errorMessage);
        return;
    }

    // Get the transition type.
    int geofenceTransition = geofencingEvent.getGeofenceTransition();

    // Test that the reported transition was of interest.
    if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
        geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

        // Get the geofences that were triggered. A single event can trigger
        // multiple geofences.
        List<TriggeringGeofence> triggeringGeofences =
            geofencingEvent.getTriggeringGeofences();

        // Get the transition details as a String.
        String geofenceTransitionDetails = getGeofenceTransitionDetails(
            this,
            geofenceTransition,
            triggeringGeofences
        );

        // Send notification and log the transition details.
        sendNotification(geofenceTransitionDetails);
        Log.i(TAG, geofenceTransitionDetails);
    } else {
        // Log the error.
        Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
            geofenceTransition));
    }
}

```

在通過 `PendingIntent` 檢測轉移事件之後，這個 `IntentService` 獲取地理圍欄轉移類型和測試一個事件是不是應用用來觸發通知的——要麼是 `GEOFENCE_TRANSITION_ENTER`，要麼是 `GEOFENCE_TRANSITION_EXIT`。然後，這個 service 會發出一個通知並且記錄轉移的詳細信息。

## 停止地理圍欄監視

當不再需要監視地理圍欄或者想要節省設備的電池電量和 CPU 週期時，需要停止地理圍欄監視。我們可以在用於添加和刪除地理圍欄的主 `activity` 裏停止地理圍欄監視；刪除地理圍欄會導致它馬上停止。API 要麼通過 request IDs，要麼通過刪除與指定 `PendingIntent` 相關的地理圍欄來刪除地理圍欄。

下面的代碼通過 `PendingIntent` 刪除地理圍欄，當設備進入或者離開之前已經添加的地理圍欄時，停止所有通知：

```

LocationServices.GeofencingApi.removeGeofences(
    mGoogleApiClient,
    // This is the same pending intent that was used in addGeofences().
    getGeofencePendingIntent()
).setResultCallback(this); // Result processed in onResult().
}

```

你可以將地理圍欄同其他位置感知的特性結合起來，比如週期性的位置更新。像要瞭解更多的信息，請看本章的其它課程。

# Android可穿戴應用

---

編寫:kesenhoo - 原文: <http://developer.android.com/training/building-wearables.html>

這些課程將教會你如何在手持應用上構建notification，並且使得這些notification能夠自動同步到可穿戴設備上。同樣也會教你如何創建直接運行在可穿戴設備上的應用。

有下面幾個課程：

## 賦予Notification可穿戴的特性

學習如何構建運行在手持設備的上得notification並且使得他們能夠同步到可穿戴上設備時有良好的體驗。

## 創建可穿戴應用

學習如何構建直接運行在可穿戴設備上的應用。

## 創建自定義的UI

學習如何為可穿戴應用創建自定義的界面。

## 發送與同步數據

學習如何在手持設備與可穿戴設備之間同步數據。

# 為Notification賦加可穿戴特性

編寫:wangyachen - 原文: <http://developer.android.com/training/wearables/notifications/index.html>

當一部Android手持設備（手機或平板）與Android可穿戴設備連接後，手持設備能夠自動的與可穿戴設備共享Notification。在可穿戴設備上，每個Notification都是以一張新卡片的形式出現在context stream中。

與此同時，為了給予用戶以最佳的體驗，開發者應當為自己創建的Notification增加一些具備可穿戴特性的功能。下面的課程將指導你如何實現同時支持手持設備和可穿戴設備的Notification。



圖1. 同時展示在手持設備和可穿戴設備的Notification

## 分集課程

- [創建一個Notification](#)

學習如何應用Android support library創建具備可穿戴特性的Notification。

- [在Notification中接收語音輸入](#)

學習在可穿戴式設備上的Notification中接收來自用戶的語音輸入，同時添加一個action，並且將錄入的消息傳遞給手持設備。

- [為Notification添加顯示界面](#)

學習如何為Notification創建附加的頁面，使得用戶在向左滑動時能看到更多的信息。

- [以stack的方式顯示Notification](#)

學習如何以stack的形式，堆砌地顯示那些從app中發出的，較為類似的Notification，使得用戶能夠看到每一個Notification是獨立顯示，而不是將各種卡片放入同一個卡片流中。

# 創建Notification

編寫:wangyachen - 原文: <http://developer.android.com/training/wearables/notifications/creating.html>

為了創建一個手持設備上的並且也能同時發送給可穿戴設備的Notification，需要使用 `NotificationCompat.Builder`。當你使用這個類創建Notification之後，如何正確展示的工作就交由系統去完成，無論他們出現在手持式設備上還是可穿戴設備上。

注意: Notification如果使用`RemoteViews`，便能夠自定義layout，並且可穿戴設備上只顯示文本和icon。但是，通過創建一個運行在可穿戴設備上的應用，開發者能夠使用自定義的 card layouts [創建自定義Notifications](#)。

## Import關鍵類

為了引入關鍵的包，在你的 `build.gradle` 文件中加入如下內容：

```
compile "com.android.support:support-v4:20.0.+"
```

現在你的項目能夠訪問關鍵的包，從support library中引入關鍵的類：

```
import android.support.v4.app.NotificationCompat;
import android.support.v4.app.NotificationManagerCompat;
import android.support.v4.app.NotificationCompat.WearableExtender;
```

## 通過Notification Builder創建Notification

`v4 support library`能夠讓開發者使用最新的特性去創建 Notification，諸如放置action button或採用large icon，而且兼容Android1.6 ( API level4 ) 及以上。

為了通過support library創建一個Notification，你需要創建一個`NotificationCompat.Builder`的實例，然後通過`notify()`去激活。例如：

```
int notificationId = 001;
// Build intent for notification content
Intent viewIntent = new Intent(this, ViewEventActivity.class);
viewIntent.putExtra(EXTRA_EVENT_ID, eventId);
PendingIntent viewPendingIntent =
    PendingIntent.getActivity(this, 0, viewIntent, 0);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setContentTitle(eventTitle)
        .setContentText(eventLocation)
        .setContentIntent(viewPendingIntent);

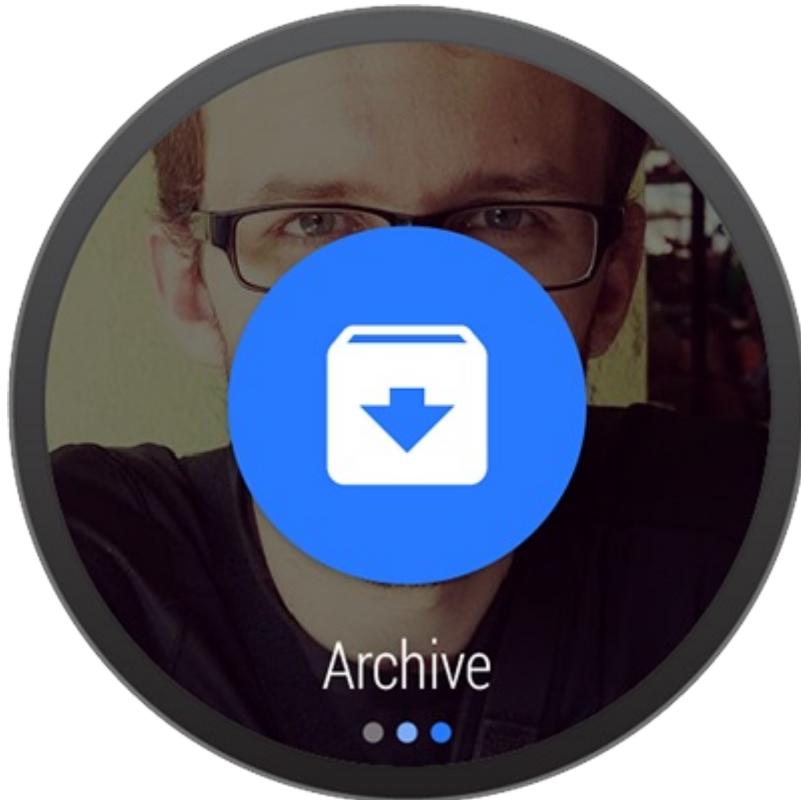
// Get an instance of the NotificationManager service
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);

// Build the notification and issues it with notification manager.
notificationManager.notify(notificationId, notificationBuilder.build());
```

當該Notification出現在手持設備上時，用戶能夠通過觸摸Notification來觸發之前通過`setContentIntent()`設置的`PendingIntent`。當該Notification出現在可穿戴設備上時，用戶能夠通過向左滑動該Notification顯示Open的action，點擊這個action能夠觸發響應的Intent，作用在手持設備上。

## 添加Action按鈕

除了通過`setContentIntent()`定義的主要的action之外，你還可以通過傳遞一個`PendingIntent`給`addAction()`的參數，從而添加更多的action。



例如，下面的代碼展示了創建一個同之前相仿的Notification，只不過在最末添加了一個在地圖上查看事件位置的action。

```
// Build an intent for an action to view a map
Intent mapIntent = new Intent(Intent.ACTION_VIEW);
Uri geoUri = Uri.parse("geo:0,0?q=" + Uri.encode(location));
mapIntent.setData(geoUri);
PendingIntent mapPendingIntent =
    PendingIntent.getActivity(this, 0, mapIntent, 0);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setContentTitle(eventTitle)
        .setContentText(eventLocation)
        .setContentIntent(viewPendingIntent)
        .addAction(R.drawable.ic_map,
            getString(R.string.map), mapPendingIntent);
```

在手持設備上，action表現為在Notification上附加的一個額外按鈕。而在可穿戴設備上，action表現為Notification左滑後出現的大按鈕。當用戶點擊action時，能夠觸發手持設備上對應的intent。

提示：如果你的Notification包含了一個"Reply"的action(例如短信類app)，你可以通過支持直接從Android可穿戴

設備返回的語音輸入，來加強該功能的使用。更多信息，詳見[Receiving Voice Input from a Notification](#)。

## 可穿戴式獨有的 Actions

如果開發者想要可穿戴式設備上的action與手持式設備不一樣的話，可以使用[WearableExtender.addAction\(\)](#)，一旦你通過這種方式添加了action，可穿戴式設備便不會顯示任何其他通過[NotificationCompat.Builder.addAction\(\)](#)添加的action。這是因為，只有通過[WearableExtender.addAction\(\)](#)添加的action才能只在可穿戴設備上顯示且不在手持式設備上顯示。

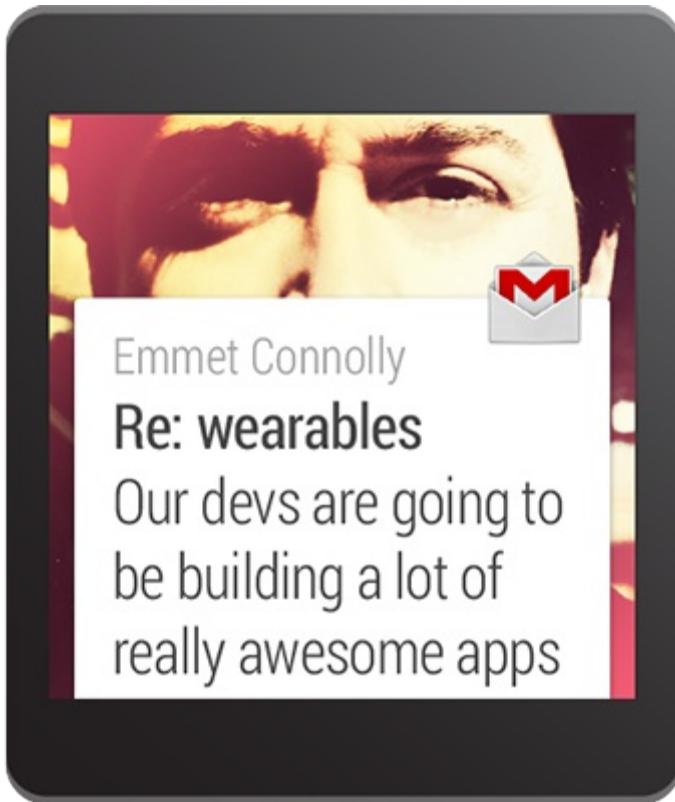
```
// Create an intent for the reply action
Intent actionIntent = new Intent(this, ActionActivity.class);
PendingIntent actionPendingIntent =
    PendingIntent.getActivity(this, 0, actionIntent,
        PendingIntent.FLAG_UPDATE_CURRENT);

// Create the action
NotificationCompat.Action action =
    new NotificationCompat.Action.Builder(R.drawable.ic_action,
        getString(R.string.label, actionPendingIntent))
    .build();

// Build the notification and add the action via WearableExtender
Notification notification =
    new NotificationCompat.Builder(mContext)
        .setSmallIcon(R.drawable.ic_message)
        .setContentTitle(getString(R.string.title))
        .setContentText(getString(R.string.content))
        .extend(new WearableExtender().addAction(action))
        .build();
```

## 添加Big View

開發者可以在Notification中通過添加某種"big view"的style來插入擴展文本。在手持式設備上，用戶能夠通過擴展的Notification看見big view的內容。在可穿戴式設備上，big view內容是默認可見的。



可以通過對[NotificationCompat.Builder](#)的對象調用[setStyle\(\)](#)，設置參數為[BigTextStyle](#)或[InboxStyle](#)的實例。

比如，下面的代碼為事件的Notification添加了一個[NotificationCompat.BigTextStyle](#)的實例，目的是為了包含完整的事件描述(這能夠包含比[setContentText\(\)](#)提供的空間所能容納的字數更多的文字)。

```
// Specify the 'big view' content to display the long
// event description that may not fit the normal content text.
BigTextStyle bigStyle = new NotificationCompat.BigTextStyle();
bigStyle.bigText(eventDescription);

NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_event)
        .setLargeIcon(BitmapFactory.decodeResource(
            getResources(), R.drawable.notif_background))
        .setContentTitle(eventTitle)
        .setContentText(eventLocation)
        .setContentIntent(viewPendingIntent)
        .addAction(R.drawable.ic_map,
            getString(R.string.map), mapPendingIntent)
        ..setStyle(bigStyle);
```

要注意的是，開發者可以通過[setLargeIcon\(\)](#)方法為任何Notification添加一張較大的背景圖片。更多關於大圖片在Notification上的設計，詳見[Design Principles of Android Wear](#)。

## 為Notification添加可穿戴式特性

如果你需要為Notification添加一些可穿戴式的特性設置，比如制定額外的內容頁，或者讓用戶通過語音輸入一些文字，那麼你可以使用[NotificationCompat.WearableExtender](#)來制定這些設置。請使用如下的API：

1. 創建一個[WearableExtender](#)的實例，設置可穿戴獨有的Notification特性。
2. 創建一個[NotificationCompat.Builder](#)的實例，就像本課程先前所說的，設置需要的Notification屬性。

3. 調用[build\(\)](#)去build一個Notification。

例如，以下代碼調用[setHintHideIcon\(\)](#)把app的icon從Notification的卡片上remove掉。

```
// Create a WearableExtender to add functionality for wearables
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender()
    .setHintHideIcon(true);

// Create a NotificationCompat.Builder to build a standard notification
// then extend it with the WearableExtender
Notification notif = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender)
    .setContentText(subject)
    .setSmallIcon(R.drawable.new_mail);
    .extend(wearableExtender)
    .build();
```

這個[setHintHideIcon\(\)](#)方法是[NotificationCompat.WearableExtender](#)提供的Notification的一個接口。

如果開發者需要稍後去讀取可穿戴特性設置的內容項，可以使用相應的get方法，該例子通過調用[getHintHideIcon\(\)](#)去獲取當前Notification是否隱藏了icon。

```
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender(notif);
boolean hintHideIcon = wearableExtender.getHintHideIcon();
```

## 傳遞Notification

如果開發者想要傳遞自己的Notification，請使用[NotificationManagerCompat](#)的API代替[NotificationManager](#)：

```
// Get an instance of the NotificationManager service
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(mContext);

// Issue the notification with notification manager.
notificationManager.notify(notificationId, notif);
```

如果開發者使用了framework中的[NotificationManager](#)，那麼[NotificationCompat.WearableExtender](#)中的一些特性就會失效，所以，請確保使用[NotificationManagerCompat](#)。

```
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender(notif);
boolean hintHideIcon = wearableExtender.getHintHideIcon();
```

[NotificationCompat.WearableExtender](#)的API允許開發者為Notification、stack Notification等添加額外的內容頁。下面的課程將學習這些特性。

下一課：[在Notifcation中接收語音輸入](#)

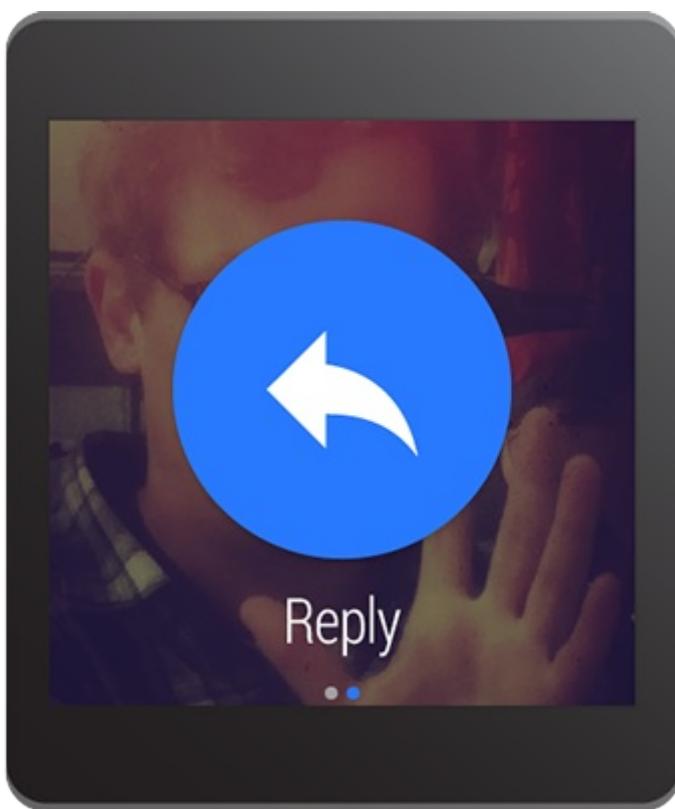
# 在Notification中接收語音輸入

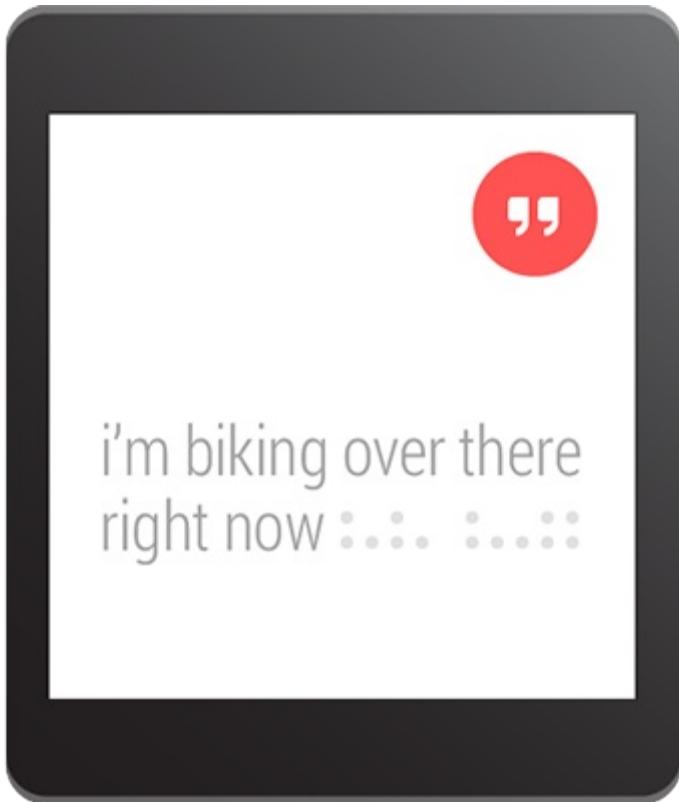
編寫:wangyachen - 原文:<http://developer.android.com/training/wearables/notifications/voice-input.html>

如果手持式設備上的Notification包含了一個輸入文本的action，比如回覆郵件，那麼，這個action正常情況下應該會調起一個activity讓用戶進行輸入。但是，當這個action出現在可穿戴式設備上時，是沒有鍵盤可以讓用戶進行輸入的，所以開發者應該讓用戶指定一個反饋或者通過RemoteInput預先設定好文本信息。

當用戶通過語音或者選擇可見的消息進行回覆時，系統會將文本的反饋信息與開發者指定的Notification中的action中的Intent進行綁定，並且將該intent發送給手持設備中的app。

注意: Android模擬器並不支持語音輸入。如果使用可穿戴式設備的模擬器的話，可以打開AVD設置中的Hardware keyboard present，實現用打字代替語音。





## 定義語音輸入

為了創建一個支持語音輸入的action，需要創建一個[RemoteInput.Builder](#)的實例，將其加到Notification的action中。這個類的構造函數接受一個String類型的參數，該參數的含義是系統用來作為語音輸入的key，這個key可以用來在手持設備中檢索出所需要的那一次語音輸入的內容。

舉個例子，下面展示瞭如何創建一個[RemoteInput](#)對象，並且提供了一個用戶label用於提示語音輸入。

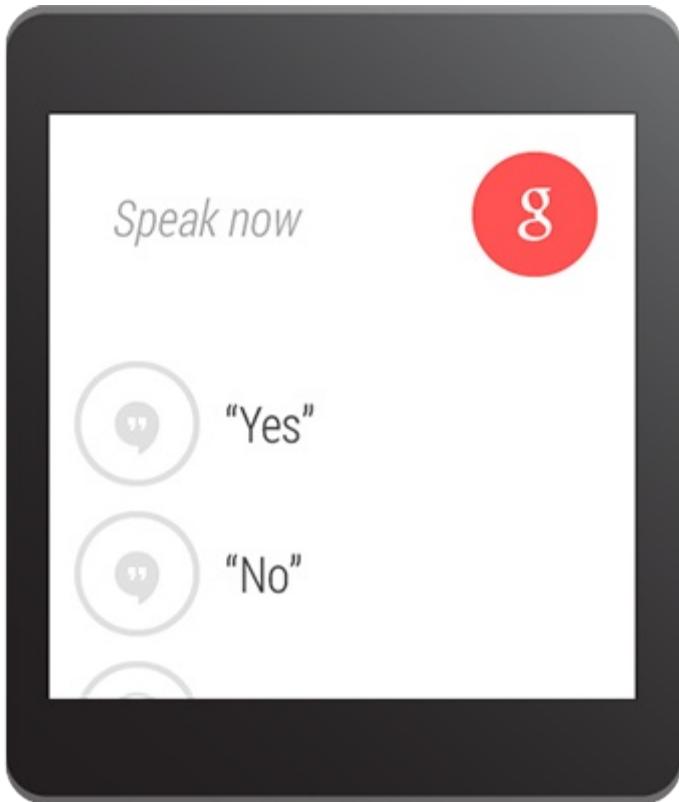
```
// Key for the string that's delivered in the action's intent
private static final String EXTRA_VOICE_REPLY = "extra_voice_reply";

String replyLabel = getResources().getString(R.string.reply_label);

RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)
    .setLabel(replyLabel)
    .build();
```

## 添加預先設定的文本反饋

除了要打開語音輸入支持之外，開發者還可以提供多達5條的文本反饋，這樣用戶可以直接進行選擇實現快速回覆。該功能可通過調用[setChoices\(\)](#)並傳遞一個String數組實現。



舉個例子，可以用resource數組的方式定義這些反饋。

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="reply_choices">
        <item>Yes</item>
        <item>No</item>
        <item>Maybe</item>
    </string-array>
</resources>
```

然後，獲得該數組，並將其添加到RemoteInput中：

```
public static final EXTRA_VOICE_REPLY = "extra_voice_reply";
...
String replyLabel = getResources().getString(R.string.reply_label);
String[] replyChoices = getResources().getStringArray(R.array.reply_choices);

RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)
    .setLabel(replyLabel)
    .setChoices(replyChoices)
    .build();
```

## 添加語音輸入作為Notification的action

為了實現設置語音輸入，可以把RemoteInput對象通過addRemoteInput())設置到一個action中。然後你可以將這個action應用到Notification中，例如：

```

// Create an intent for the reply action
Intent replyIntent = new Intent(this, ReplyActivity.class);
PendingIntent replyPendingIntent =
    PendingIntent.getActivity(this, 0, replyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT);

// Create the reply action and add the remote input
NotificationCompat.Action action =
    new NotificationCompat.Action.Builder(R.drawable.ic_reply_icon,
        getString(R.string.label, replyPendingIntent))
        .addRemoteInput(remoteInput)
        .build();

// Build the notification and add the action via WearableExtender
Notification notification =
    new NotificationCompat.Builder(mContext)
        .setSmallIcon(R.drawable.ic_message)
        .setContentTitle(getString(R.string.title))
        .setContentText(getString(R.string.content))
        .extend(new WearableExtender().addAction(action))
        .build();

// Issue the notification
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(mContext);
notificationManager.notify(notificationId, notification);

```

當程序發出這個Notification的時候，用戶在可穿戴設備上左滑便可以看到reply的按鈕。

## 接受語音輸入作為String值

通過調用`getResultsFromIntent()`方法，將返回值放在"Reply"的action指定的intent中，開發者便可以在回覆的action的intent中指定的activity裏，接收到用戶轉錄後的消息。

該方法返回的是包含了文本反饋的Bundle，接下來你可以通過查詢Bundle中的內容來獲得這條反饋。

**注意:**請不要使用`Intent.getExtras()`來獲取語音輸入的結果，因為語音輸入的內容是存儲在`ClipData`中的。`getResultsFromIntent()`提供了一條很方便的途徑來接收字符串類型的語音信息，並且不需要經過`ClipData`自身的調用。

下面的代碼展示了一個接收intent，並且返回語音反饋信息的方法，該方法是依據之前例子中的`EXTRA_VOICE_REPLY`作為key進行檢索。

```

/**
 * Obtain the intent that started this activity by calling
 * Activity.getIntent() and pass it into this method to
 * get the associated voice input string.
 */

private CharSequence getMessageText(Intent intent) {
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);
    if (remoteInput != null) {
        return remoteInput.getCharSequence(EXTRA_VOICE_REPLY);
    }
}
return null;
}

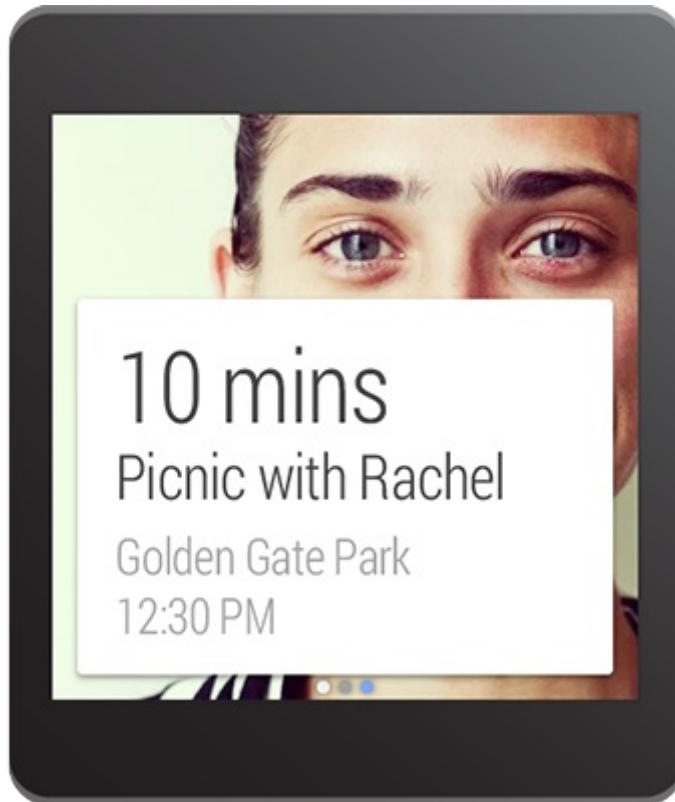
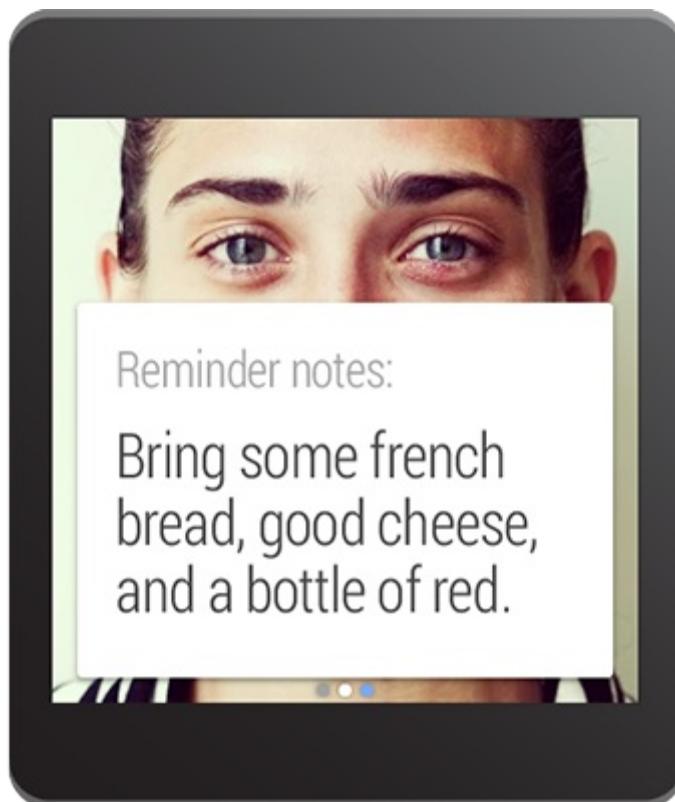
```

下一課：[為Notification添加顯示頁面](#)

# 為Notification添加顯示頁面

編寫:wangyachen - 原文:<http://developer.android.com/training/wearables/notifications/pages.html>

當開發者想要在不需要用戶在他們的手機上打開app的情況下，還可以允許表達更多的信息，那麼開發者可以在可穿戴設備上的Notification中添加一個或更多的頁面。



為了創建一個多頁的Notification，開發者需要：

1. 通過NotificationCompat.Builder創建主Notification（首頁），以開發者想要的方式使其出現在手持設備上。
2. 通過NotificationCompat.Builder為可穿戴設備添加更多的頁面。
3. 通過addPage())方法為主Notification應用這些添加的頁面，或者通過addPages())添加一個以Collection組織的多個頁面。

舉個例子，以下代碼為Notification添加了第二個頁面：

```
// Create builder for the main notification
NotificationCompat.Builder notificationBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.new_message)
        .setContentTitle("Page 1")
        .setContentText("Short message")
        .setContentIntent(viewPendingIntent);

// Create a big text style for the second page
BigTextStyle secondPageStyle = new NotificationCompat.BigTextStyle();
secondPageStyle.setBigContentTitle("Page 2")
    .bigText("A lot of text...");

// Create second page notification
Notification secondPageNotification =
    new NotificationCompat.Builder(this)
        .setStyle(secondPageStyle)
        .build();

// Add second page with wearable extender and extend the main notification
Notification twoPageNotification =
    new WearableExtender()
        .addPage(secondPageNotification)
        .extend(notificationBuilder)
        .build();

// Issue the notification
notificationManager =
    NotificationManagerCompat.from(this);
notificationManager.notify(notificationId, twoPageNotification);
```

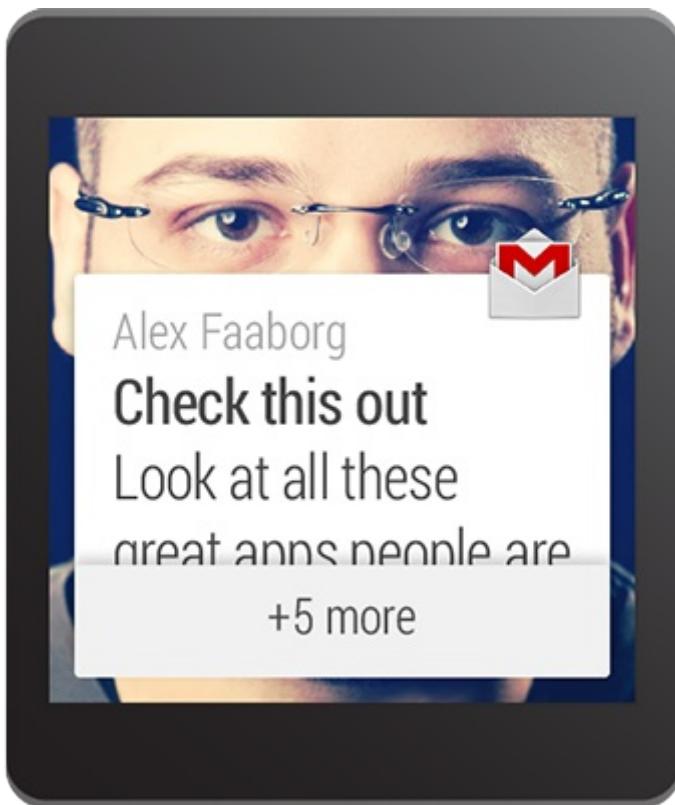
下一課：[以Stack的方式顯示Notifications](#)

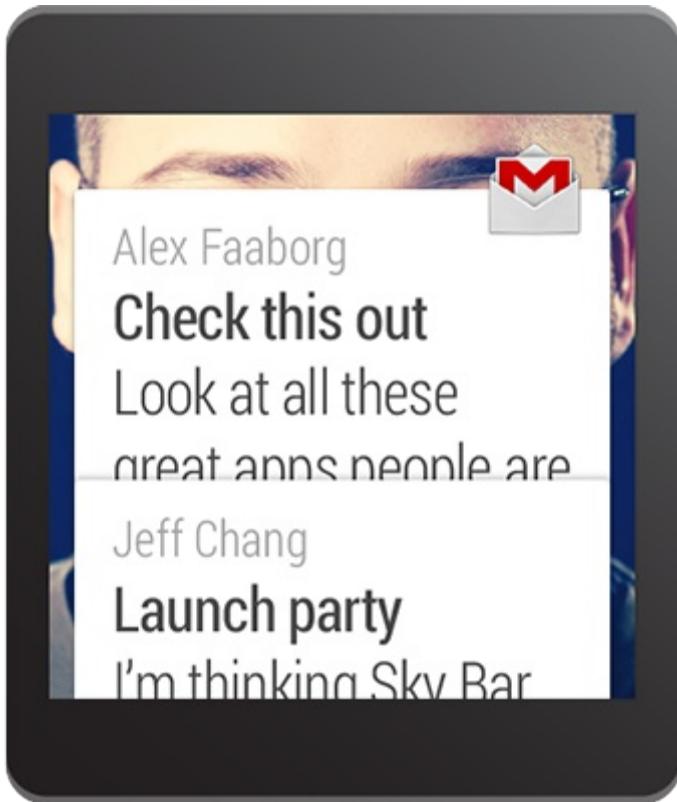
# 以Stack的方式顯示Notifications

編寫:wangyachen - 原文: <http://developer.android.com/training/wearables/notifications/stacks.html>

當為手持式設備創建Notification時，開發者應該將多個相似的Notification合併成一個概括式的Notification。例如，如果app創建了一系列接收短信的Notification，開發者不應該把它們都展示出來，當多於一條短信被接收的時候，用一條Notification提供總結性信息比如"2條新消息"。

儘管如此，一個概括式的Notification在可穿戴設備上並不是很有用處，因為用戶不能在可穿戴設備上閱讀每條消息的詳細內容(他們必須在手持式設備上打開相應的app才能看到更多信息)。所以對可穿戴設備而言，開發者應該將所有的Notification都集中起來，以stack的形式進行展示。這個stack展示的時候就像一張卡片，用戶可以在上面以擴展的方式分別看到其他的Notification。通過新方法`setGroup()`能夠實現該功能，同時，也能保持手持式設備上顯示為一條概括式的Notification。





## 將每個Notification添加到一個羣組中

為了創建一個stack，可以對每個想要放入該stack的Notification調用`setGroup()`，並且指定group key。然後調用`notify()`將其發送至可穿戴設備上。

```
final static String GROUP_KEY_EMAILS = "group_key_emails";

// Build the notification, setting the group appropriately
Notification notif = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender1)
    .setContentText(subject1)
    .setSmallIcon(R.drawable.new_mail);
    .setGroup(GROUP_KEY_EMAILS)
    .build();

// Issue the notification
NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);
notificationManager.notify(notificationId1, notif);
```

稍後，當開發者創建另一個Notification的時候，指定同樣的group key。當在調用`notify()`的時候，這個Notification就會出現在之前那個Notification的同一個stack中，而非新建一張卡片。

```
Notification notif2 = new NotificationCompat.Builder(mContext)
    .setContentTitle("New mail from " + sender2)
    .setContentText(subject2)
    .setSmallIcon(R.drawable.new_mail);
    .setGroup(GROUP_KEY_EMAILS)
    .build();

notificationManager.notify(notificationId2, notif2);
```

在默認的情況下，Notification的排列順序由開發者添加的先後順序決定，最近的Notification會被放置在最頂端。你可以通過[setSortKey\(\)](#)來修改Notification的排順序。

## 添加概括式Notification

在手持設備上提供一個概括式的Notification是很重要的。因此除了要將每條單獨的Notification放置在同一個stack group中，還需要添加一個概括式的Notification，並對其調用[setGroupSummary\(\)](#)即可實現。

該Notification並不會出現在可穿戴設備上的stack中，只會出現在手持式設備上。



```
Bitmap largeIcon = BitmapFactory.decodeResource(getResources(),
    R.drawable.ic_large_icon);

// Create an InboxStyle notification
Notification summaryNotification = new NotificationCompat.Builder(mContext)
    .setContentTitle("2 new messages")
    .setSmallIcon(R.drawable.ic_small_icon)
    .setLargeIcon(largeIcon)
    ..setStyle(new NotificationCompat.InboxStyle()
        .addLine("Alex Faaborg Check this out")
        .addLine("Jeff Chang Launch Party")
        .setBigContentTitle("2 new messages")
        .setSummaryText("johndoe@gmail.com"))
    .setGroup(GROUP_KEY_EMAILS)
    .setGroupSummary(true)
    .build();

notificationManager.notify(notificationId3, summaryNotification);
```

該Notification使用了[NotificationCompat.InboxStyle](#)，這個style能夠讓開發者很輕鬆地創建郵件或者短信app的Notifications。開發者可以對概括式Notification使用這個style，或者[NotificationCompat](#)中定義的其他style，或者不使用任何style也可以。

提示:如果想要和上面截圖中一樣的設計文本，請參考[Styling with HTML markup](#)和[Styling with Spannables](#)。

概括式Notification能夠在不顯示在可穿戴設備上的前提下做到影響可穿戴設備上的Notification。當開發者創建一個概括式Notification時，可以利用[NotificationCompat.WearableExtender](#)，調用[setBackground\(\)](#)或者[addAction\(\)](#)為可穿戴設備上的整個stack設置一個背景圖片或者一個action。以下代碼展示瞭如何為整個stack設置背景：

```
Bitmap background = BitmapFactory.decodeResource(getResources(),
    R.drawable.ic_background);
```

```
NotificationCompat.WearableExtender wearableExtender =  
    new NotificationCompat.WearableExtender()  
    .setBackground(background);  
  
// Create an InboxStyle notification  
Notification summaryNotificationWithBackground =  
    new NotificationCompat.Builder(mContext)  
    .setContentTitle("2 new messages")  
    ...  
    .extend(wearableExtender)  
    .setGroup(GROUP_KEY_EMAILS)  
    .setGroupSummary(true)  
    .build();
```

[下一課：創建可穿戴的應用](#)

# 創建可穿戴的應用

編寫:kesenhoo - 原文:<http://developer.android.com/training/wearables/apps/index.html>

可穿戴應用直接運行在穿戴設備上，應用可以直接訪問例如傳感器與GPU這樣的硬件。這些應用和一般的Android應用的基礎部分是一致的，只是在設計與可用性還有一些特殊功能上有比較大差異。手持設備與可穿戴設備上的應用主要有下面的一些差異：

- 系統會強制執行超時機制。如果你顯示了一個Activity，用戶並沒有進行操作，設備會進入睡眠狀態。當設備喚醒時，穿戴設備會顯示主界面而不是你剛纔的activity。如果你想要持續的顯示一些東西，請使用notification來替代。
- 相比起手持設備的應用，可穿戴應用的界面相對更小，功能也相對更少。他僅僅包含了那些對於可穿戴有意義的功能，這些功能通常是手持設備的一個子集。通常來說，你應該儘可能的把運行操作搬到手持設備上，然後發送操作結果到可穿戴設備。
- 用戶不能直接給可穿戴設備安裝應用。你需要給手持設備的應用綁定一個可穿戴設備的應用。當用戶安裝手持設備的應用時，系統會自動安裝可穿戴應用。然而，為了開發便利，你還是可以直接安裝應用到可穿戴設備。
- 可穿戴應用可以使用大多數的標準Android APIs，除了下面的以外：
  - android.webkit
  - android.print
  - android.app.backup
  - android.appwidget
  - android.hardware.usb

在使用某個API之前，你可以通過執行hasSystemFeature() 來判斷功能是否可用。

Note: 我們推薦使用Android Studio來開發Android Wear的應用，因為它提供了建立工程，添加庫依賴，打包程序等等在ADT上沒有的功能。下面的培訓課程的前提是假設你已經在使用Android Studio了。

## Lessons

- [創建並執行可穿戴應用\(Creating and Running a Wearable App\)](#)

學習如何創建一個包含了可穿戴與手持應用的Android Studio工程。學習如何在設備或者模擬器上執行程序。

- [創建自定義的佈局\(Creating Custom Layouts\)](#)

學習如何為notification與activiyt，創建並顯示一個自定義的佈局

- [添加語言能力\(Adding Voice Capabilities\)](#)

學習如何使用語音指令啓動一個activity，學習如何啓動系統語音識別應用來獲取用戶的語音輸入。

- [打包可穿戴應用\(Packaging Wearable Apps\)](#)

學習如何把可穿戴應用打包到手持應用上。這使得系統能夠在安裝Google Play上的手持應用時自動安裝可穿戴應用。

- [通過藍牙進行調試\(Debugging over Bluetooth\)](#)

學習如何通過藍牙而不是USB來調試你的可穿戴應用。



# 創建並執行可穿戴應用

編寫:kesenhoo - 原文:<http://developer.android.com/training/wearables/apps/creating.html>

可穿戴應用可以直接運行在可穿戴的設備上。擁有訪問類似傳感器的硬件權限，還有操作activity，services等權限。

你無法直接發佈可穿戴應用到Google Play商城，需要利用手持應用來達到目的。因為可穿戴的設備不支持Google Play商城，所以當用戶下載手持設備應用的時候，，會自動安裝可穿戴應用到可穿戴設備上。手持應用還可以用來處理一些複雜繁重的任務，網絡指令，或者其他任務，最好發送操作結果返回給可穿戴設備。

這節課會介紹如何創建一個包含了手持應用與可穿戴應用的工程。

## 搭建Android Wear模擬器或者真機設備。

我們推薦在真機上進行開發，這樣可以更好的評估用戶體驗。然而，模擬器可以使得你在不同類型的設備屏幕上進行模擬，這對測試來說更加有用。

### 搭建Android Wear虛擬設備

建立Android Wear虛擬設備需要下面幾個步驟：

1. 點擊Tools > Android > AVD Manager.
2. 點擊Create....
3. 填寫下面幾項詳細的設置，其餘選項保留默認：
  - AVD Name - AVD的名字
  - Device - Android Wear圓形還是方形
  - Target - Android 4.4W - API Level 20
  - CPU/ABI - Android Wear ARM (armeabi-v7a)
  - Keyboard - 選擇Hardware keyboard present
  - Skin - 圓形還是方形取決於選擇的設備類型
  - Snapshot - 不勾選 selected
  - Use Host GPU - 勾選，為了支持自定義的activity能夠顯示可穿戴的notification。
4. 點擊OK.
5. 啓動模擬器：
  - 選擇你剛纔創建的虛擬設備
  - 點擊Start...，然後選擇Launch.
  - 等待模擬器初始化直到顯示Android Wear的主界面。
6. 匹配你的手持和模擬器：
  - 在你的手持設備上，從Google Play安裝 Android Wear 應用(這是一個由Google公司寫的用來匹配的應用)
  - 通過USB連接你的手持設備到你的電腦。
  - 切換AVD的接口到手持設備(這個步驟需要每次連接都執行)

```
adb -d forward tcp:5601 tcp:5601
```

- 啓動手持設備上的 Android Wear 應用，並連接到模擬器。
- 點擊右上角的菜單，選擇Demo Cards。
- 你選擇的卡片呈現在模擬器上會類似一個Notification。

## 搭建Android Wear真機

建立Android Wear真機，需要下面幾個步驟：

- 在你的手持設備的Google Play上安裝 Android Wear 應用。
- 按照應用的命令指示與你的可穿戴設備進行匹對。如果你有做建立notification的操作，這個步驟剛好可以測試這一功能。
- 保持 Android Wear 應用在手機上的打開狀態。
- 通過USB連接可穿戴設備到電腦上，這樣你能夠直接安裝應用到可穿戴設備上。在可穿戴設備與 Android Wear 應用上會顯示一個消息提示，是否允許進行調試。
- 在 Android Wear 應用上，總是選擇允許連接。

Android Studio上的Tool的窗口可以顯示可穿戴設備的日誌。當你執行 `adb devices` 命令的時候，也可以看到wearable的存在。

## 創建Wear項目

在開始開發之前，需要創建一個項目包含可穿戴應用與手持應用這兩個模塊。在Android Studio中，點擊File > New Project 然後按照[創建項目的指引](#)進行操作。如果你按照安裝嚮導操作，需要輸入下面的信息：

1. 在確認項目的窗口，輸入你的應用的名稱與包名。
2. 在應用參數選擇窗口：
  - 勾選Phone 與 Tablet 並選擇API 8: Android 2.2 (Froyo) 作為Minimum SDK.
  - 勾選可穿戴並選擇API 20: Android 4.4 (KitKat Wear) 作為Minimum SDK.
3. 在第一個添加activity的窗口，選擇為Mobile模塊添加一個空白的activity。
4. 在第二個添加activity的窗口，選擇為Wear模塊添加一個空白的activity。

當安裝嚮導完成後，Andriod Studio創建了一個包含Mobile與Wear兩個模塊的項目。你可以在這2個模塊中各自創建activity，service，layout等等。在手持應用裏面，需要承擔大部分繁重的任務，例如網絡請求，密集計算任務或者是需要大量用戶交互的任務。待這些任務完成之後，再通常把任務結果通過notification發送給可穿戴設備上，或者是通過同步機制發送數據給可穿戴設備。

Note: 可穿戴模塊包含了一個"Hello World"的activity，它是使用 WatchViewStub 的佈局。WatchViewStub是可穿戴support library中的一個UI組件。

## 安裝可穿戴應用

在開發過程中，你可以像安裝手持應用一樣直接安裝可穿戴應用。可以使用 `adb install` 命令也可以使用Android Studio上面的Play按鈕。

當需要把應用發佈給用戶的時候，你需要把可穿戴應用打包到手持應用中。當用戶從Google Play安裝手持應用時，連接上得可穿戴設備會自動收到可穿戴應用。

Note: 如果你給應用簽名是Debug Key，是無法完成自動安裝可穿戴應用的。請參考[打包可穿戴應用](#)獲取更多信息，學習如何正確的打包。

為了安裝"Hello World"應用到可穿戴設備，在Android Studiod的Run/Debug的下拉選項中選中Wear模塊，點擊Play按鈕即可。在可穿戴設備上會顯示activity並打印"Hello world!"

## include需要的libraries

項目安裝嚮導會自動把合適的模塊依賴添加到對應的build.gradle文件中。然而，這些依賴並不是必須得，請閱讀下面描述判斷你是否需要這些依賴。

- Notifications

The [Android v4 support library](#) (or v13)能夠支持運行在手持應用的notification也能夠在可穿戴設備上顯示。

對於只顯示在可穿戴設備上得notification(這意味着，他們是由直接執行在可穿戴設備上得app進行處理的)，你可以把Wear模塊僅僅使用標準APIs (API Level 20) 並且把Mobile模塊的依賴support library移除。

- Wearable Data Layer

可穿戴與手持設備之間進行同步與發送數據需要使用Wearable Data Layer APIs, 你需要最新版本的[Google Play Services](#)。如果你不需要這些APIs，可以從這兩個模塊中把這部分的依賴移除。

- Wearable UI support library

這是一個非官方正式的library，它包含了為可穿戴設備設計的UI組件。我們鼓勵你在你的應用中使用他們。因為這些組件是最佳實踐的例證。但是他們可能隨時發生變化。然而，如果library有更新，你的應用並不會發送崩潰，因為那些代碼已經編譯到你的應用中了。為了獲取更新包中新的功能，你只需要更新鏈接到新的版本並相應的更新你的應用就好了。這個library只是在你需要創建可穿戴應用時纔會使用到。

# 創建自定義的佈局

編寫: kesenhoo - 原文: <http://developer.android.com/training/wearables/apps/layouts.html>

為可穿戴設備創建佈局是和手持設備是一樣的。但是 不要期望通過搬遷手持應用的功能與UI到可穿戴上會有一個好的用戶體驗。僅僅在有需要的時候，你才應該創建自定義的佈局。請參考可穿戴設備的[design guidelines](#)學習如何設計一個優秀的可穿戴應用。

## 創建自定義Notification

通常來說，你應該在手持應用上創建好notification，然後讓它自動同步到可穿戴設備上。這使得你只需要創建一次notification，然後可以在不同類型的設備(不僅僅是可穿戴設備，也包含車載設備與電視)上進行顯示，免去為不同設備進行重新設計。

如果標準的notification風格無法滿足你的需求(例如[NotificationCompat.BigTextStyle](#) 或者 [NotificationCompat.InboxStyle](#))，你可以使用activity，顯示一個自定義的佈局來達到目的。在可穿戴設備上你只可以創建並處理自定義的notification，同時系統無法為這些notification同步到手持設備上。

Note: 當在可穿戴設備上創建自定義的notification時，你可以使用API Level 20上標準的APIs，不需要使用Support Library。

為了創建自定義的notification，步驟如下：

1. 創建佈局並設置這個佈局為需要顯示的activity的content view:

```
public void onCreate(Bundle bundle){  
    ...  
    setContentView(R.layout.notification_activity);  
}
```

2. 為了使得activity能夠顯示在可穿戴設備上，需要在manifest文件中為activity定義必須的屬性。你需要把activity聲明為exportable，embeddable以及擁有一個空的task affinity。我們也推薦把activity的主題設置為 Theme.DeviceDefault.Light 。例如：

```
<activity android:name="com.example.MyDisplayActivity"  
        android:exported="true"  
        android:allowEmbedded="true"  
        android:taskAffinity=""  
        android:theme="@android:style/Theme.DeviceDefault.Light" />
```

3. 為activity創建PendingIntent，例如：

```
Intent notificationIntent = new Intent(this, NotificationActivity.class);  
PendingIntent notificationPendingIntent = PendingIntent.getActivity(this, 0, notificationIntent,  
                    PendingIntent.FLAG_UPDATE_CURRENT);
```

4. 創建Notification並執行setDisplayIntent()方法，參數是前面創建的PendingIntent。當用戶查看這個notification時，系統使用這個PendingIntent來啓動activity。

5. 觸發notification使用notify() 的方法。

Note: 當notification呈現在主頁時，系統會根據notification的語義，使用一個標準的模板來呈現它。這個模板

可以在所有的錶盤上進行顯示。當用戶往上滑動notification時，將會看到為這個notification準備的自定義的activity。

## 使用Wearable UI庫創建佈局

當你使用Android Studio的引導功能創建一個Wearable應用的時候，會自動包含一個非官方的UI庫文件。你也可以通過給build.gradle文件添加下面的依賴聲明把庫文件添加到項目：

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.google.android.support:wearable:+'  
    compile 'com.google.android.gms:play-services-wearable:+'  
}
```

這個庫文件幫助你建立你設計的UI。下面是一些主要的類：

- BoxInsetLayout - 一個能夠感知屏幕的形狀並把子控件居中擺放的FrameLayout。
- CardFragment - 一個能夠可拉伸，垂直可滑動卡片的fragment。
- CircledImageView - 一個圓形的image view。
- ConfirmationActivity - 一個在用戶完成一個操作之後用來顯示確認動畫的activity。 \*DismissOverlayView - 一個用來實現長按消失的View。
- ViewPager - 一個可以橫向與縱向滑動的局部控制器。你需要提供一個PagerAdapter用來生成顯示頁面的數據。
- PagerAdapter - 一個提供給ViewPager顯示頁面的適配器。
- FragmentPagerAdapter - 一個為每個頁面提供單獨的fragment的適配器。
- WatchViewStub - 一個可以根據屏幕的形狀生成特定佈局的類。
- WearableListView - 一個針對可穿戴設備優化過後的ListView。它會垂直的顯示列表內容，並在用戶停止滑動時自動顯示最靠近的item。

[點擊下載完整的API說明文檔](#) 這個文檔會詳細的介紹每一個UI組件。

# 添加語音能力

編寫: kesenhoo - 原文: <http://developer.android.com/training/wearables/apps/voice.html>

語音指令是可穿戴體驗的一個重要的部分。這使得用戶可以釋放雙手，快速發出指令。穿戴提供了2種類型的語音操作：

- 系統提供的

這些語音指令都是基於任務的，並且內置在Wear的平臺內。你在activity中過濾你想要接收的指令。例如包含"Take a note"或者"Set an alarm"的指令。

- 應用提供的

這些語音指令都是基於應用的，你需要像聲明一個Launcher Icon一樣定義這些指令。用戶通過說"Start XXX"來使用那些語音指令，然後會啓動你指定啓動的activity。

## 聲明系統提供的語音指令

Android Wear平臺基於用戶的操作提供了一些語音指令，例如"Take a note"或者"Set an alarm"。用戶發出想要做的操作指令，讓系統尋找應該啓動最合適的activity。

當用戶說出語音指令時，你的應用能夠被過濾出來啓動一個activity。如果你想要啓動一個service在後臺執行任務，需要顯示一個activity呈現作為線索。當你想要廢棄這個可見的線索時，需要確保執行了finish()。

例如，對於"Take a note"的指令，定義一個MyNoteActivity來接收這個指令：

```
<activity android:name="MyNoteActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="com.google.android.voicesearch.SEARCH" />
    </intent-filter>
</activity>
```

下面列出了Wear平臺支持的語音指令：

Name	Example Phrases	Intent
Call a car/taxi	"OK Google, get me a taxi"  "OK Google, call me a car"	Action  <code>com.google.android.gms.actions.RESERVE_TAXI_RESERVATION</code>
Take a note	"OK Google, take a note"  "OK Google, note to self"	Action  <code>android.intent.action.SEND</code>  Category  <code>com.google.android.voicesearch.SESSION_NOTE</code>  Extras  <code>android.content.Intent.EXTRA_TEXT</code> - a string with note body
Set alarm	"OK Google, set an alarm for 8 AM"  "OK Google, wake me up at 6 tomorrow"	Action  <code>android.intent.action.SET_ALARM</code>  Extras  <code>android.provider.AlarmClock.EXTRA_HOUR</code> - an integer with the hour of the alarm.  <code>android.provider.AlarmClock.EXTRA_MINUTES</code> - an integer with the minute of the alarm  (these 2 extras are optional, either none or both are provided)

Set timer	"Ok Google, set a timer for 10 minutes"	Action <code>android.provider.AlarmClock.ACTION_SET_TIMER</code>  Extras <code>android.provider.AlarmClock.EXTRA_LENGTH</code> - an integer in the range of 1 to 86400 (number of seconds in 24 hours) representing the length of the timer
Start/Stop a bike ride	"OK Google, start cycling"  "OK Google, start my bike ride"  "OK Google, stop cycling"	Action <code>vnd.google.fitness.TRACK</code>  Mime Type <code>vnd.google.fitness.activity/biking</code>  Extras <code>actionStatus</code> - a string with the value <code>ActiveActionStatus</code> when starting and <code>CompletedActionStatus</code> when stopping.
Start/Stop a run	"OK Google, track my run"  "OK Google, start running"  "OK Google, stop running"	Action <code>vnd.google.fitness.TRACK</code>  MimeType <code>vnd.google.fitness.activity/running</code>  Extras <code>actionStatus</code> - a string with the value <code>ActiveActionStatus</code> when starting and <code>CompletedActionStatus</code> when stopping

Start/Stop a workout	"OK Google, start a workout"  "OK Google, track my workout"  "OK Google, stop workout"	Action vnd.google.fitness.TRACK  MimeType vnd.google.fitness.activity/other  Extras actionStatus - a string with the value ActiveActionStatus when starting and CompletedActionStatus when stopping
Show heart rate	"OK Google, what's my heart rate?"  "OK Google, what's my bpm?"	Action vnd.google.fitness.VIEW  MimeType vnd.google.fitness.data_type/com.google.heart_rate.bpm
Show step count	"OK Google, how many steps have I taken?"  "OK Google, what's my step count?"	Action vnd.google.fitness.VIEW  MimeType vnd.google.fitness.data_type/com.google.step_count.cumulative

關於註冊intent與獲取intent extra的信息，請參考[Common intents](#).

## 聲明應用提供的語音指令

如果系統提供的語音指令無法滿足你的需求，你可以使用"Start MyActivityName"的指令來直接啓動你的應用。

註冊一個"Start"指令和註冊手持應用上得Launcher是一樣的。

在"Start"指令的後面需要指定的文字，這個文字需要註冊在[activity](#)的label屬性上。例如，下面的設置能夠識別"Start MyRunningApp"的語音指令並啓動StartRunActivity.

```
<application>
    <activity android:name="StartRunActivity" android:label="MyRunningApp">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

# 獲取輸入的自由語音

除了使用語音指令來啓動activity之外，你也可以執行系統內置的語言識別activity來獲取用戶的語音輸入。這對於獲取用戶的輸入信息非常有幫助，例如執行搜索或者發送一個消息。

在你的應用中，`startActivityForResult()`使用 `ACTION_RECOGNIZE_SPEECH` 啓動系統語音識別應用。在`onActivityResult()`中處理返回的結果：

```
private static final int SPEECH_REQUEST_CODE = 0;

// Create an intent that can start the Speech Recognizer activity
private void displaySpeechRecognizer() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    // Start the activity, the intent will be populated with the speech text
    startActivityForResult(intent, SPEECH_REQUEST_CODE);
}

// This callback is invoked when the Speech Recognizer returns.
// This is where you process the intent and extract the speech text from the intent.
@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if (requestCode == SPEECH_REQUEST && resultCode == RESULT_OK) {
        List<String> results = data.getStringArrayListExtra(
            RecognizerIntent.EXTRA_RESULTS);
        String spokenText = results.get(0);
        // Do something with spokenText
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

# 打包可穿戴應用

編寫: kesenhoo - 原文: <http://developer.android.com/training/wearables/apps/packaging.html>

當發佈應用給用戶之前，你必須把可穿戴應用打包到手持應用內。因為不能直接在可穿戴設備上瀏覽並安裝應用。如果打包正確，當用戶下載手持應用時，系統會自動下發可穿戴應用到匹對的可穿戴設備上。

Note: 如果開發時簽名用的是debug key，這個特性是無法正常工作的。在開發時，需要使用 `adb install` 的命令或者Android Studio來安裝可穿戴應用。

## 使用Android Studio打包

在Android Studio中打包可穿戴應用有下面幾個步驟：

1. 在手持應用的**build.gradle**文件中把可穿戴應用聲明為依賴：

```
dependencies {  
    compile 'com.google.android.gms:play-services:5.0.+@aar'  
    compile 'com.android.support:support-v4:20.0.+'  
    wearApp project(':wearable')  
}
```

2. 點擊Build > Generate Signed APK... 安裝屏幕上的指示來制定你的release key併為你的app進行簽名。Android Studio導出簽名好的手持應用，他內置了可穿戴應用。或者，你可以在可穿戴應用與手持應用的**build.gradle**文件裏面建立一個簽名規則。為了能夠正常自動推送可穿戴應用，這兩個應用都必須簽名。

```
android {  
    ...  
    signingConfigs {  
        release {  
            keyAlias 'myAlias'  
            keyPassword 'myPw'  
            storeFile file('path/to/release.keystore')  
            storePassword 'myPw'  
        }  
    }  
    buildTypes {  
        release {  
            ...  
            signingConfig signingConfigs.release  
        }  
    }  
    ...  
}
```

通過點擊Android Studio右邊的Gradle按鈕來建立手持應用，並執行**assembleRelease**任務。這個任務放在Project name > Handheld module name > **assembleRelease**.

Note: 這個例子中把密碼寫在了Gradle文件中，這應該不是期待的寫法。請參考[Configure signing settings](#)學習如何為密碼創建環境變量。

## 分別為可穿戴應用與手持應用進行簽名

如果你的Build任務需要為可穿戴應用與手持應用籤不同的Key，你可以像下面一樣在手持應用的**build.gradle**文件中聲明規則。

```
dependencies {  
    ...  
    wearApp files('/path/to/wearable_app.apk')  
}
```

你可以爲手持應用手動進行籤任何形式的Key (可以是Android Studio Build > Generate Signed APK...的方式，也可以是Gradle signingConfig規則的方式)

## 手動打包

如果你使用的是其他IDE，你仍然可以把可穿戴應用打包到手持應用中。

1. 把簽好名的可穿戴應用放到手持應用的 res/raw 目錄下。我們把這個應用作爲 wearable\_app.apk 。
2. 創建 res/xml/wearable\_app\_desc.xml 文件，裏麪包含可穿戴設備的版本信息與路徑。例如：

```
<wearableApp package="wearable.app.package.name">  
<versionCode>1</versionCode>  
<versionName>1.0</versionName>  
<rawPathResId>wearable_app</rawPathResId>  
</wearableApp>
```

- package, versionCode, 與 versionName需要和可穿戴應用的AndroidManifest.xml裏面的信息一致。`rawPathResId` 是一個static的變量表示APK的名稱。。
3. 添加 meta-data 標簽到你的手持應用的 `<application>` 標簽下，指明引用wearable\_app\_desc.xml文件

```
<meta-data android:name="com.google.android.wearable.beta.app"  
          android:resource="@xml/wearable_app_desc"/>
```

4. 構建並簽名手持應用。

## 關閉資源壓縮

許多構建工具會自動壓縮放在res/raw目錄下的文件。因爲可穿戴APK已經被壓縮過了，那些壓縮工作再次壓縮會導致應用無法正常安裝。

這樣的話，安裝失敗。在手持應用上，`PackageUpdateService` 會輸出如下的錯誤日誌："this file cannot be opened as a file descriptor; it is probably compressed."

Android Studio 默認不會壓縮你的APK，如果你使用另外一個構建流程，需要確保不會發生重複壓縮可穿戴應用的事情。

# 通過藍牙進行調試

編寫: kesenhoo - 原文: <http://developer.android.com/training/wearables/apps/bt-debugging.html>

你可以通過藍牙來調試你的可穿戴應用，通過藍牙把調試數據輸出到手持設備上，手持設備是有連接到開發電腦上的。

## 搭建好設備用來調試

- 開啓手持設備的USB調試：
  - 打開設置應用並滑動到底部。
  - 如果在設置裏面沒有開發者選項，點擊關於手機，滑動到底部，點擊build number 7次。
  - 返回並點擊開發者選項。
  - 開啓USB調試。
- 開啓可穿戴設備的藍牙調試：
  - 點擊主界面2次，來到Wear菜單界面。
  - 滑動到底部，點擊設置。
  - 滑動到底部，如果沒有開發者選項，點擊Build Number 7次。
  - 點擊開發者選項。
  - 開啓藍牙調試。

## 建立調試會話

1. 在手持設備上，打開 Android Wear 這個伴侶應用。
2. 點擊右上角的菜單，選擇設置。
3. 開啓藍牙調試。你將會在選項下面看到一個小的狀態信息：

```
Host: disconnected  
Target: connected
```

4. 通過USB連接手持設備到你的電腦上，並執行下面的命令：

```
adb forward tcp:4444 localabstract:/adb-hub; adb connect localhost:4444
```

**Note:** 你可以使用任何可用的端口。

在 Android Wear 伴侶應用上，你將會看到狀態變為：

```
Host: connected  
Target: connected
```

## 調試你的應用

當運行adb devices的命令時，你的可穿戴設備是作為localhost:4444的。執行任何的adb命令，需要使用下面的格式：

```
adb -s localhost:4444 <command>
```

如果沒有任何其他的設備通過TCP/IP連接到手持設備，你可以使用下面的簡短命令：

```
adb -e <command>
```

例如：

```
adb -e logcat  
adb -e shell  
adb -e bugreport
```

# 為可穿戴設備創建自定義UI

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/index.html>

可穿戴apps的用戶界面明顯的不同於手持設備。可穿戴設備的Apps應該參考Android 可穿戴設計規範和實現推薦的UI patterns, 並且在App中保證統一的用戶體驗以適合可穿戴設備。

這個課程將教你如何為可穿戴設備創建自定義UI和自定義notifications在所有Android可穿戴設備上看上去不錯,從使用這些UI patterns :

- 卡片
- 倒計時確認
- 長按忽略
- 二維選擇器
- 多選列表

可穿戴UI庫是Android SDK的Google Repository中的一部分，其中提供的類可以幫助你實現這些patterns和創建layouts工作在圓形和方形的Android可穿戴設備。

Note:我們推薦使用Android Studio做Android Wear開發,它提供工程初始配置,庫包含和方便的打包,這些在ADT中是不可用的。這系列教程假定你是正在使用Android Studio的。

## Lessons

- [定義 Layouts](#)

學習如何創建在圓形和方形Android Wear設備上看起來不錯的layouts。

- [創建卡片](#)

學習如何創建自定義layouts的卡片

- [創建列表](#)

學習如何創建為可穿戴設備優化的列表

- [創建二維選擇器](#)

學習如何實現二維選擇器UI模式以導航各頁數據

- [顯示確認器](#)

學習如何在用戶完成操作時顯示確認動畫

- [退出全屏Activities](#)

學習如何實現長按忽略UI模式以退出全屏activities

# 定義Layouts

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/layouts.html>

可穿戴設備使用與手持Android設備同樣的佈局技術，但需要有具體的約束來設計。不要以一個手持app的角度開發功能和UI以期待提供一個好的體驗。關於如何設計優秀的可穿戴apps的更多信息，請閱讀[Android Wear Design Guidelines](#)。

當你為Android Wear apps創建layouts時，你需要同時考慮方形和圓形屏幕的設備。在圓形Android Wear設備上所有放置在靠近屏幕邊角的內容會被剪裁掉，所以為方形屏幕設計的layouts不能在圓形設備上很好的工作。對這類問題是示範請查看這個視屏[Full Screen Apps for Android Wear](#)。

舉個例子，figure 1展示了下面的layout在圓形和方形屏幕上看起來是怎樣的：

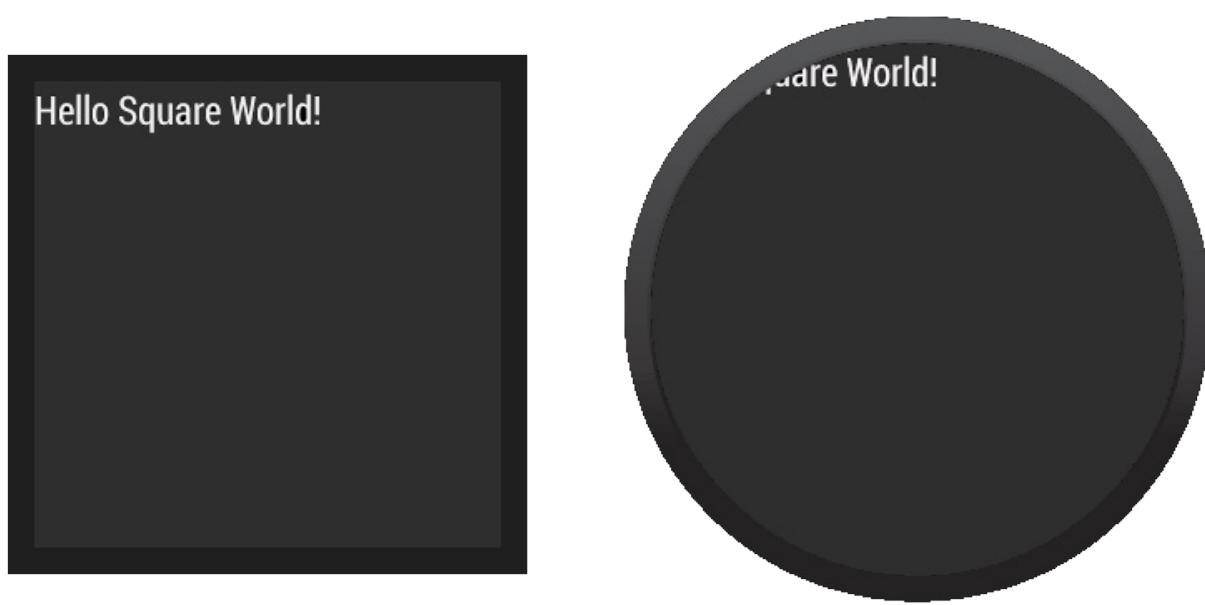


Figure 1. 為方形屏幕設計的layouts不能在圓形設備上很好工作的示範

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

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_square" />
</LinearLayout>
```

text沒有正確的顯示在圓形屏幕上。

Wearable UI Library為這個問題提供了兩種不同的解決方案：

- 為圓形和方形屏幕定義不同的layouts。你的app會在運行時檢查設備屏幕形狀後inflates正確的layout。

- 用一個包含在library裏面的特殊layout同時適配方形和圓形設備。這個layout會在不同形狀的設備屏幕窗口中插入間隔。

當你希望你的app在不同形狀的屏幕上看起來不同時，你可以典型的使用第一種方案。當你希望用一個相似的layout在兩種屏幕上且在圓形屏幕上沒有視圖被邊緣剪裁時，你可以使用第二種方案。

## 添加Wearable UI庫

Android Studio會在你使用工程嚮導時includes你在wear module中的Wearable UI庫。為了編譯你的工程和這個庫，確保 Extras > Google Repository 包已經被安裝在Android SDK manager裏、以下的wear module依賴被包含在你的build.gradle文件中。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.google.android.support:wearable:+'
    compile 'com.google.android.gms:play-services-wearable:+'
}
```

要實現以下的佈局方法‘com.google.android.support:wearable’依賴是必須的。

瀏覽[API reference documentation](#)查看Wearable UI類庫。

## 為方形和圓形屏幕指定不同的Layouts

在Wearable UI庫中的WatchViewStub類允許你為方形和圓形屏幕指定不同的layouts。這個類會在運行時檢查屏幕形狀後inflates符合的layout。

在你的app中使用這個類以應對不用的屏幕形狀：

- 在你的activity's layout以WatchViewStub為主元素。
- 為方形屏幕指定一個layout解釋文件使用rectLayout屬性。
- 為圓形屏幕指定一個layout解釋文件使用roundLayout屬性。

定義你的activity's layout類似於：

```
<android.support.wearable.view.WatchViewStub
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/watch_view_stub"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:rectLayout="@layout/rect_activity_wear"
    app:roundLayout="@layout/round_activity_wear">
</android.support.wearable.view.WatchViewStub>
```

在你的activity中inflate這個layout：

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

然後為方形和圓形屏幕創建不同的layout描述文件，在這個例子中，你需要創建文件  
res/layout/rect\_activity\_wear.xml 和 res/layout/round\_activity\_wear.xml。像創建手持apps的layouts一樣定義這些layouts，但同時考慮可穿戴設備的限制。系統會在運行時以屏幕形狀inflates適合的layout。

## 取得layout views

你為方形或圓形屏幕定義的layouts在WatchViewStub檢查完屏幕形狀之前不會被inflated。所以你的app不能立即取得它們的views。為了取得這些views，你需要在你的activity中設置一個listener，當屏幕適配的layout被inflated時會通知這個listener：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wear);

    WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub);
    stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener() {
        @Override public void onLayoutInflated(WatchViewStub stub) {
            // Now you can access your views
            TextView tv = (TextView) stub.findViewById(R.id.text);
            ...
        }
    });
}
```

## 使用形狀感知的Layout

包含在你的Wearable UI庫中的 BoxInsetLayout 繼承自 [FrameLayout](#)允許你定義一個同時適配方形和圓形屏幕的layout。這個類適用於需要根據屏幕形狀插入間隔的情況並允許你簡單的對齊views在屏幕邊緣或中心。

figure 2中，在 BoxInsetLayout 裏的灰色方形區域會在圓形屏幕裏應用所需的窗口間隔後自動放置child views。為了顯示在這個區域內，子views需要具體聲明附加屬性 layout\_box 為這些值：

- 一個top, bottom, left, right的複合屬性。比如 "left|top" 說明子view的左和上邊緣如figure 2。
- all 說明子view的內容在灰色方形內如figure 2。



Figure 2. 在圓形屏幕上的窗口間隔

在方形屏幕上，窗口間隔為0、layout\_box 屬性會被忽略。

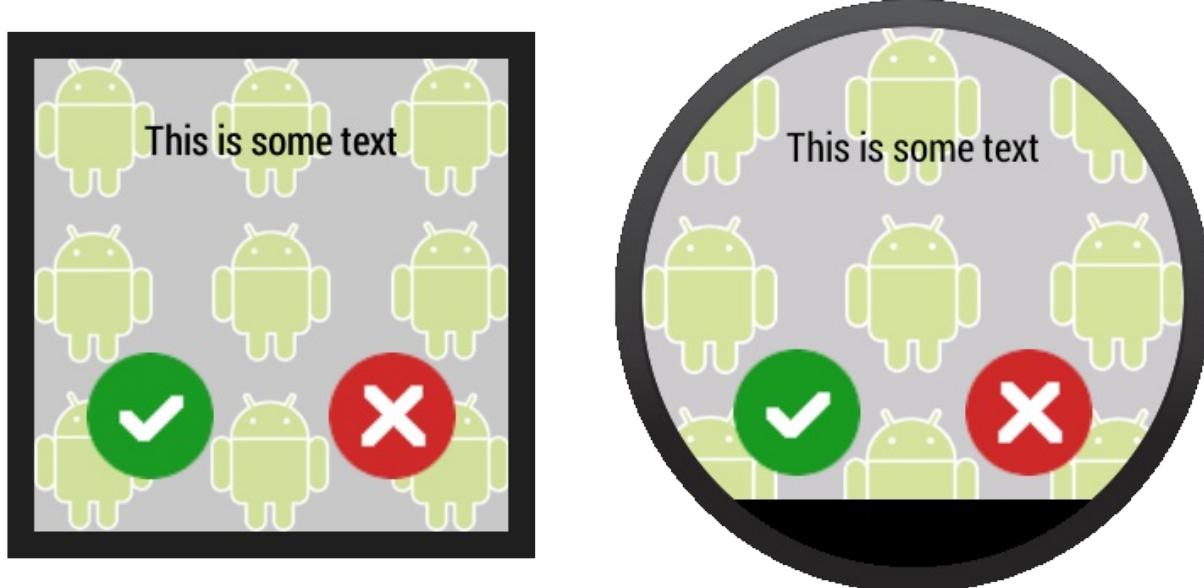


Figure 3. 同一個layout定義工作在方形和圓形屏幕上

這個layout在figure 3中展示了在圓形和方形屏幕上使用 BoxInsetLayout :

```
<android.support.wearable.view.BoxInsetLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    * android:background="@drawable/robot_background"  
    android:layout_height="match_parent"  
    android:layout_width="match_parent"
```

```
*     android:padding="15dp">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    *
    *     android:padding="5dp"
    *     app:layout_box="all">

        <TextView
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:text="@string/sometext"
            android:textColor="@color/black" />

        <ImageButton
            android:background="@null"
            android:layout_gravity="bottom|left"
            android:layout_height="50dp"
            android:layout_width="50dp"
            android:src="@drawable/ok" />

        <ImageButton
            android:background="@null"
            android:layout_gravity="bottom|right"
            android:layout_height="50dp"
            android:layout_width="50dp"
            android:src="@drawable/cancel" />
    </FrameLayout>
</android.support.wearable.view.BoxInsetLayout>
```

注意layout中這些被加\*的部分

- android:padding = "15dp"

這行指定了 BoxInsetLayout 元素的padding。因為在圓形設備商窗口間隔大於 15dp，這個padding只工作在方形屏幕上。

- android:padding = "5dp"

這行指定 FrameLayout 內部的元素padding。這個padding同時生效在方形和圓形屏幕上。在方形屏幕上總的padding是20dp(15+5)、在圓形屏幕上是5dp。

- app:layout\_box = "all"

這行聲明 FrameLayout 和它的子views都被放在有窗口間隔的圓形屏幕裏。這行在方形屏幕上沒有任何效果。

# 創建Cards

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/cards.html>

卡片以一致的外觀在不同的apps上為用戶呈現當前信息。這個章節為你演示瞭如何在你的Android Wear apps中創建卡片。

Wearable UI Library提供了為穿戴設備特別設計的卡片實現。這個庫包含了 CardFrame 類，它在卡片風格的框架中包裹views，且提供一個白色的背景，圓角，和光投射陰影。CardFrame 只能包裹一個直接的子view，這通常是一個layout manager，你可以向它添加其他views以定製卡片內容。

你有兩種方法向你的app添加cards：

- 直接使用或繼承 CardFragment 類。
- 在你的layout內，在一個 CardScrollView 中添加一個card。

Note: 這個課程為你展示了如何在Android Wear activities中添加cards。Android可穿戴設備上的notifications同樣以卡片顯示。更多信息請查看 [Adding Wearable Features to Notifications](#)

## 創建Card Fragment

CardFragment 類提供一個默認card layout含有一個title一個description和一個icon，如果你需要的card layout看起來和默認figure 1一樣，使用這個方法向你的app添加cards。

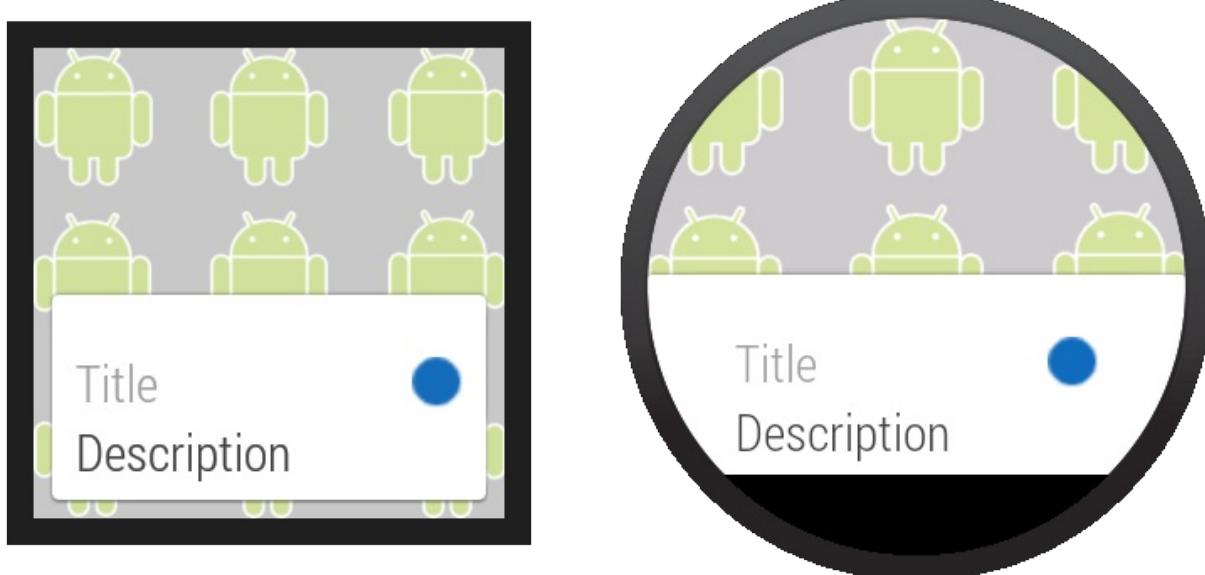


Figure 1. 默認的 CardFragment layout.

為了添加一個 CardFragment 到你的app：

- 在你的layout為包含內容卡片的節點分配一個ID
- 在你的activity中創建一個 CardFragment 實例
- 使用fragment manager添加 CardFragment 實例到這個容器

下面的示例代碼顯示了Figure 1中的屏幕顯示代碼：

```
<android.support.wearable.view.BoxInsetLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:background="@drawable/robot_background"  
    android:layout_height="match_parent"  
    android:layout_width="match_parent">  
  
    <FrameLayout  
        android:id="@+id/frame_layout"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        app:layout_box="bottom">  
  
    </FrameLayout>  
</android.support.wearable.view.BoxInsetLayout>
```

下面的代碼添加CardFragment 實例到Figure 1的activity中：

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_wear_activity2);  
  
    FragmentManager fragmentManager = getFragmentManager();  
    FragmentTransaction fragmentTransaction  
        = fragmentManager.beginTransaction();  
    CardFragment cardFragment  
        = CardFragment.create(getString(R.string.cftitle),getString(R.string.cfdesc),R.drawable.p);  
    fragmentTransaction.add(R.id.frame_layout, cardFragment);  
    fragmentTransaction.commit();  
}
```

為了創建使用自定義layout的card，繼承這個類然後override onCreateView方法。

## 添加一個CardFrame到你的Layout

你也可以直接添加一個card到你的layout描述，類似於figure 2。當你希望為layout描述文件中的card自定義一個layout時使用這個方法。

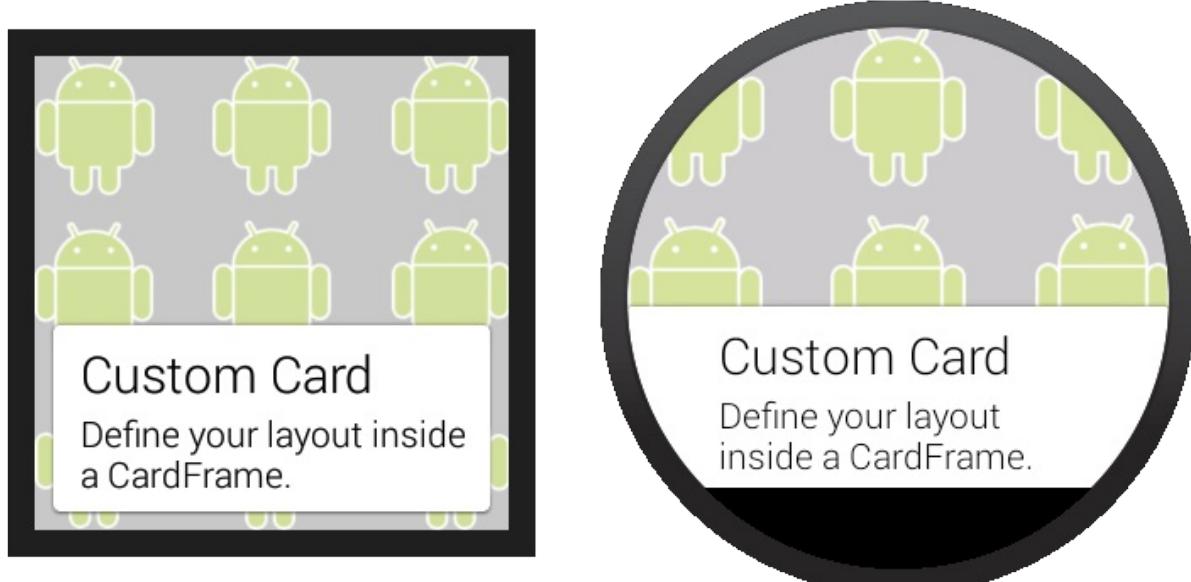


Figure 2. 添加一個 CardFrame 到你的layout.

下面的layout代碼例子示範了一個含有兩個節點的垂直linear layout。你可以創建更加複雜的layouts以適合你需要的app。

```
<android.support.wearable.view.BoxInsetLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@drawable/robot_background"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <android.support.wearable.view.CardScrollView
        android:id="@+id/card_scroll_view"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        app:layout_box="bottom">

        <android.support.wearable.view.CardFrame
            android:layout_height="wrap_content"
            android:layout_width="fill_parent">

            <LinearLayout
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:orientation="vertical"
                android:paddingLeft="5dp">
                <TextView
                    android:fontFamily="sans-serif-light"
                    android:layout_height="wrap_content"
                    android:layout_width="match_parent"
                    android:text="@string/custom_card"
                    android:textColor="@color/black"
                    android:textSize="20sp"/>
                <TextView
                    android:fontFamily="sans-serif-light"
                    android:layout_height="wrap_content"
                    android:layout_width="match_parent"
                    android:text="@string/description"
                    android:textColor="@color/black"
                    android:textSize="14sp"/>
            </LinearLayout>
        </android.support.wearable.view.CardFrame>
    </android.support.wearable.view.CardScrollView>
</android.support.wearable.view.BoxInsetLayout>
```

這個例子上的 CardScrollView 節點讓你可以配置gravity，當包含的內容小於容器時。這個例子為card對齊屏幕底部：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wear_activity2);

    CardScrollView cardScrollView =
        (CardScrollView) findViewById(R.id.card_scroll_view);
    cardScrollView.setCardGravity(Gravity.BOTTOM);
}
```

CardScrollView 檢查屏幕形狀後以不同的顯示方式顯示卡片在圓形或方形設備上，它使用更寬的側邊緣在圓形屏幕上。不管怎樣，在 BoxInsetLayout 中放置 CardScrollView 節點然後使用layout\_box="bottom"屬性對圓形屏幕上的卡片對齊底部並且沒有內容被剪裁是有用的。

# 創建Lists

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/lists.html>

Lists在可穿戴設備上簡單的從一組選項中選擇一個項目，這個課程為你演示瞭如何在Android Wear apps中創建lists。

Wearable UI庫包含了 WearableListView 類，用於實現為可穿戴設備優化的list。

Note: Android SDK 中的 Notifications 例子示範瞭如何在你的apps中使用 WearableListView。這個例子的位置在 android-sdk/samples/android-20/wearable/Notifications 目錄。

為了在你的Android Wear apps中創建list:

1. 添加 WearableListView 元素到你的activity的layout描述中。
2. 為你的list items創建一個自定義layout實現。
3. 為你的list items使用這個實現創建一個layout描述。
4. 創建一個adapter以填充list。
5. 為 WearableListView 元素指定這個adapter。

這些步驟的詳細描述在下面的章節中。



Figure 3: A list view on Android Wear.

## 添加List View

下面的layout使用 BoxInsetLayout 添加了一個list view到activity中，所以這個列表可以正確的現實在圓形和方形兩種設備上：

```

<android.support.wearable.view.BoxInsetLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@drawable/robot_background"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <FrameLayout
        android:id="@+id/frame_layout"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        app:layout_box="left|bottom|right">

        <android.support.wearable.view.WearableListView
            android:id="@+id/wearable_list"
            android:layout_height="match_parent"
            android:layout_width="match_parent">
        </android.support.wearable.view.WearableListView>
    </FrameLayout>
</android.support.wearable.view.BoxInsetLayout>

```

## 爲List Items創建一個Layout實現

在許多例子中，每個 list item 都由一個icon和一個描述組成。Android SDK中的Notifications 例子實現了一個自定義 layout：繼承`LinearLayout`以合併兩元素在每個list item。這個layout也實現了`WearableListView.OnCenterProximityListener`interface以實現在用戶滾動 WearableListView 時改變item的icon顏色和漸隱文字。

```

public class WearableListItemLayout extends LinearLayout
    implements WearableListView.OnCenterProximityListener {

    private ImageView mCircle;
    private TextView mName;

    private final float mFadedTextAlpha;
    private final int mFadedCircleColor;
    private final int mChosenCircleColor;

    public WearableListItemLayout(Context context) {
        this(context, null);
    }

    public WearableListItemLayout(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public WearableListItemLayout(Context context, AttributeSet attrs,
                                  int defStyle) {
        super(context, attrs, defStyle);

        mFadedTextAlpha = getResources()
            .getInteger(R.integer.action_text_faded_alpha) / 100f;
        mFadedCircleColor = getResources().getColor(R.color.grey);
        mChosenCircleColor = getResources().getColor(R.color.blue);
    }

    // Get references to the icon and text in the item layout definition
    @Override
    protected void onFinishInflate() {
        super.onFinishInflate();
        // These are defined in the layout file for list items
        // (see next section)
        mCircle = (ImageView) findViewById(R.id.circle);
        mName = (TextView) findViewById(R.id.name);
    }
}

```

```

@Override
public void onCenterPosition(boolean animate) {
    mName.setAlpha(1f);
    ((GradientDrawable) mCircle.getDrawable()).setColor(mChosenCircleColor);
}

@Override
public void onNonCenterPosition(boolean animate) {
    ((GradientDrawable) mCircle.getDrawable()).setColor(mFadedCircleColor);
    mName.setAlpha(mFadedTextAlpha);
}
}

```

你也可以創建animator對象以放大中間的item的icon。你可以使用WearableListView.OnCenterProximityListener interface的 onCenterPosition() 和 onNonCenterPosition()回調方法來管理你的animators。更多關於animators的信息請查看[Animating with ObjectAnimator](#)

## 為Items創建Layout解釋

在你為list items 實現自定義layout之後，你需要提供一個layout解釋文件以具體說明list item中的組件參數。下面的layout解釋使用先前的自定義layout實現然後定義icon和text view的ID以匹配layout實現類。

res/layout/list\_item.xml

```

<com.example.android.support.wearable.notifications.WearableListItemLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center_vertical"
    android:layout_width="match_parent"
    android:layout_height="80dp">
    <ImageView
        android:id="@+id/circle"
        android:layout_height="20dp"
        android:layout_margin="16dp"
        android:layout_width="20dp"
        android:src="@drawable/wl_circle"/>
    <TextView
        android:id="@+id/name"
        android:gravity="center_vertical|left"
        android:layout_width="wrap_content"
        android:layout_marginRight="16dp"
        android:layout_height="match_parent"
        android:fontFamily="sans-serif-condensed-light"
        android:lineSpacingExtra="-4sp"
        android:textColor="@color/text_color"
        android:textSize="16sp"/>
</com.example.android.support.wearable.notifications.WearableListItemLayout>

```

## 創建Adapter以填充List

Adapter用內容填充 WearableListView。下面的adapter基於strings數組簡單的填充了List：

```

private static final class Adapter extends WearableListView.Adapter {
    private String[] mDataset;
    private final Context mContext;
    private final LayoutInflater mInflater;

    // Provide a suitable constructor (depends on the kind of dataset)
    public Adapter(Context context, String[] dataset) {
        mContext = context;
        mInflater = LayoutInflater.from(context);
        mDataset = dataset;
    }
}

```

```

}

// Provide a reference to the type of views you're using
public static class ItemViewHolder extends WearableListView.ViewHolder {
    private TextView textView;
    public ItemViewHolder(View itemView) {
        super(itemView);
        // find the text view within the custom item's layout
        textView = (TextView) itemView.findViewById(R.id.name);
    }
}

// Create new views for list items
// (invoked by the WearableListView's layout manager)
@Override
public WearableListView.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                       int viewType) {
    // Inflate our custom layout for list items
    return new ItemViewHolder(mInflater.inflate(R.layout.list_item, null));
}

// Replace the contents of a list item
// Instead of creating new views, the list tries to recycle existing ones
// (invoked by the WearableListView's layout manager)
@Override
public void onBindViewHolder(WearableListView.ViewHolder holder,
                           int position) {
    // retrieve the text view
    ItemViewHolder itemHolder = (ItemViewHolder) holder;
    TextView view = itemHolder.textView;
    // replace text contents
    view.setText(mDataset[position]);
    // replace list item's metadata
    holder.itemView.setTag(position);
}

// Return the size of your dataset
// (invoked by the WearableListView's layout manager)
@Override
public int getItemCount() {
    return mDataset.length;
}
}

```

## 連接Adapter和設置Click Listener

在你的[activity](#)中，從你的layout中取得 WearableListView 元素的引用，分配一個adapter實例以填充list，然後設置一個click listener以完成動作當用戶選擇了一個特定的list item。

```

public class WearActivity extends Activity
    implements WearableListView.ClickListener {

    // Sample dataset for the list
    String[] elements = { "List Item 1", "List Item 2", ... };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_list_activity);

        // Get the list component from the layout of the activity
        WearableListView listView =
            (WearableListView) findViewById(R.id.wearable_list);

        // Assign an adapter to the list
        listView.setAdapter(new Adapter(this, elements));

        // Set a click listener
    }
}

```

```
    listView.setOnClickListener(this);
}

// WearableListView click listener
@Override
public void onClick(WearableListView.ViewHolder v) {
    Integer tag = (Integer) v.itemView.getTag();
    // use this data to complete some action ...
}

@Override
public void onTopEmptyRegionClick() {
}

}
```

# 創建2D-Picker

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/2d-picker.html>

Android Wear中的2D Picker模式允許用戶像換頁一樣從一組項目中導航和選擇。

要實現這個模式，你需要添加一個 GridViewPager 元素到你的activity的layout中然後實現一個繼承 FragmentGridPagerAdapter類的adapter以提供一組頁面。

Note: Android SDK中的GridViewPager例子示範瞭如何在你的apps中使用 GridViewPager layout。這個例子的位置在 android-sdk/samples/android-20/wearable/GridViewPager目錄中。

## 添加Page Grid

像下面一樣添加一個 GridViewPager 元素到你的layout描述文件：

```
<android.support.wearable.view.GridViewPager  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/pager"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

你可以使用任何定義Layouts技術以保證你的2D picker可以工作在圓形和方形兩種設備上。

## 實現Page Adapter

Page Adapter提供一組頁面以填充 GridViewPager 部件。要實現這個adapter，你需要繼承Wearable UI Library中的 FragmentGridPagerAdapter 類。

舉個例子□，Android SDK內的 GridViewPager例子中包含了下面的adapter實現將提供一組靜態cards和自定義背景圖片：

```
public class SampleGridPagerAdapter extends FragmentGridPagerAdapter {  
  
    private final Context mContext;  
  
    public SampleGridPagerAdapter(Context ctx, FragmentManager fm) {  
        super(fm);  
        mContext = ctx;  
    }  
  
    static final int[] BG_IMAGES = new int[] {  
        R.drawable.debug_background_1, ...  
        R.drawable.debug_background_5  
    };  
  
    // A simple container for static data in each page  
    private static class Page {  
        // static resources  
        int titleRes;  
        int textRes;  
        int iconRes;  
        ...  
    }  
}
```

```
// Create a static set of pages in a 2D array
private final Page[][] PAGES = { ... };

// Override methods in FragmentGridPagerAdapter
...
}
```

picker調用 `getFragment` 和 `getBackground`來取得內容以顯示到每個grid的位置中。

```
// Obtain the UI fragment at the specified position
@Override
public Fragment getFragment(int row, int col) {
    Page page = PAGES[row][col];
    String title =
        page.titleRes != 0 ? mContext.getString(page.titleRes) : null;
    String text =
        page.textRes != 0 ? mContext.getString(page.textRes) : null;
    CardFragment fragment = CardFragment.create(title, text, page.iconRes);

    // Advanced settings (card gravity, card expansion/scrolling)
    fragment.setCardGravity(page.cardGravity);
    fragment.setExpansionEnabled(page.expansionEnabled);
    fragment.setExpansionDirection(page.expansionDirection);
    fragment.setExpansionFactor(page.expansionFactor);
    return fragment;
}

// Obtain the background image for the page at the specified position
@Override
public ImageReference getBackground(int row, int column) {
    return ImageReference.forDrawable(BG_IMAGES[row % BG_IMAGES.length]);
}
```

`getRowCount`方法告訴picker有多少行內容是可獲得的，然後 `getColumnCount`方法告訴picker每行中有多少列內容是可獲得的。

```
// Obtain the number of pages (vertical)
@Override
public int getRowCount() {
    return PAGES.length;
}

// Obtain the number of pages (horizontal)
@Override
public int getColumnCount(int rowNum) {
    return PAGES[rowNum].length;
}
```

adapter是實現細節依賴於你的特定的某組pages。由adapter提供每個頁面類型的 Fragment。在這個例子中，每個 page是一個使用默認card layouts的 CardFragment 實例。然而，你可以混合不同類型的pages在同一個2D picker，比如cards，action icons，和自定義layouts，由你的情況決定。

不是所有行都需要有同樣數量的pages，注意這個例子中的每行有不同的列數。你也可以用一個 GridViewPager 組件實現只有一行或一列的1D picker。

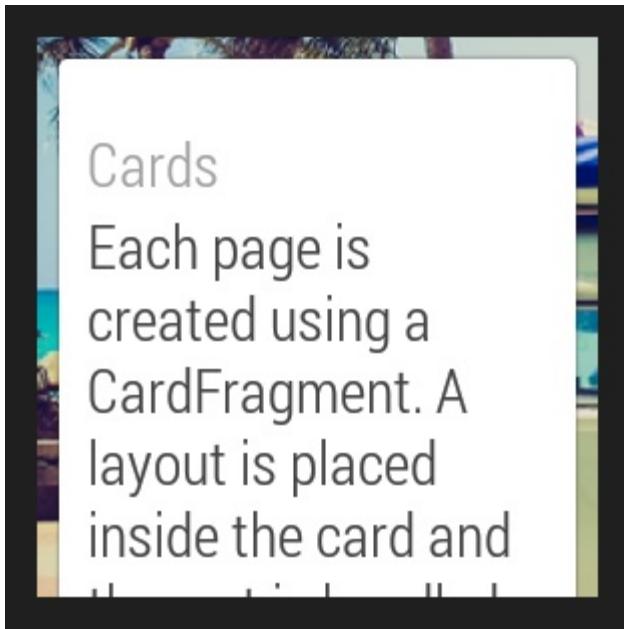


Figure 1:GridViewPager例子

GridViewPager 提供了滾動支持當cards內容超出設備屏幕。這種例子配置了每張card需要被擴展，所以用戶可以滾動卡片的內容。當用戶到達到滾動卡片的底部，向同樣方向滑動將顯示grid中的下個page（當下一page可用時）。

你可以使用 `getBackground()` 方法自定義每頁page的背景。當用戶劃過page，GridViewPager 自動使用視差滾動和淡出效果在不同的背景之間。

## 分配adapter實例給page grid

在你的[activity](#)中，分配一個你的adapter實現實例給 GridViewPager 組件：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        final ViewPager pager = (ViewPager) findViewById(R.id.pager);
        pager.setAdapter(new SampleGridPagerAdapter(this, getSupportFragmentManager()));
    }
}
```

# 顯示Confirmation

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/confirm.html>

Android Wear apps中的**Confirmations**通常全屏或是相比於手持app佔更大的部分。這樣確保用戶可以一眼看到確認器(confirmations)且有一個足夠大的觸摸區域用於取消一個操作。

Wearable UI Library幫助你在你的Android Wear apps中顯示確認器(confirmation)動畫和定時器：

## Confirmation timers

- 自動confirmation定時器為用戶顯示一個包含動畫的定時器，讓用戶可以取消他們最近的操作。

## Confirmation animations

- Confirmation animations 為用戶一個在操作完成時的完成的視覺反饋。

下面的章節為你演示瞭如何實現這些樣式。

## 使用自動 Confirmation 定時器

自動 Confirmation 定時器讓用戶只需要取消操作。當用戶做一個操作，你的app顯示帶有定時動畫的一個cancel按鈕，用戶可以在定時結束前選擇取消操作。如果用戶選擇取消操作貨定時結束你的app會得到一個通知。

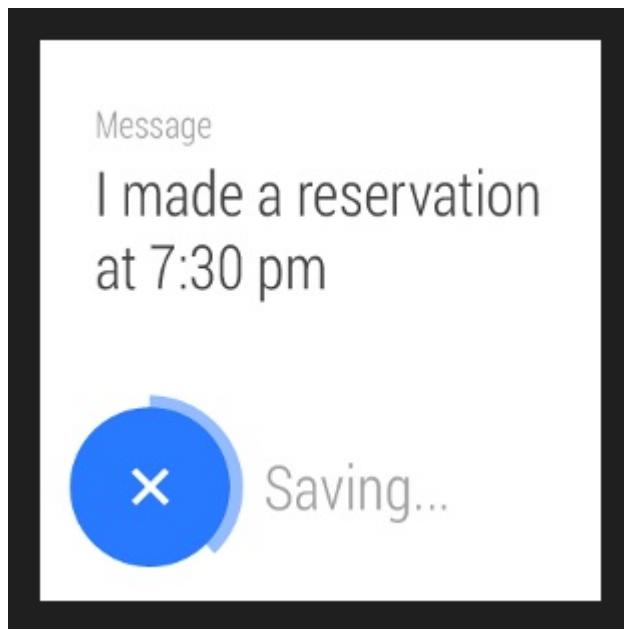


Figure 1: A confirmation timer.

為了在用戶完成操作時顯示一個confirmation timer：

- 添加 `DelayedConfirmationView` 元素到你的layout中。
- 在你的`activity`中實現 `DelayedConfirmationListener` 接口。
- 當用戶完成一個操作時，設置定時器的定時時間然後啓動它。

像下面這樣添加 `DelayedConfirmationView` 元素到你的layout中：

```
<android.support.wearable.view.DelayedConfirmationView  
    android:id="@+id/delayed_confirm"  
    android:layout_width="40dp"  
    android:layout_height="40dp"  
    android:src="@drawable/cancel_circle"  
    app:circle_border_color="@color/lightblue"  
    app:circle_border_width="4dp"  
    app:circle_radius="16dp">  
</android.support.wearable.view.DelayedConfirmationView>
```

你可以在layout解釋裏分配 android:src 中的drawable資源顯示在圓中然後直接設置圓的屬性。

為了獲得定時結束貨用戶點擊按鈕的通知，你需要在[activity](#)中實現相應的接口模塊：

```
public class WearActivity extends Activity implements DelayedConfirmationView.DelayedConfirmationListener {
```

```
    private DelayedConfirmationView mDelayedView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_wear_activity);  
  
        mDelayedView =  
            (DelayedConfirmationView) findViewById(R.id.delayed_confirm);  
        mDelayedView.setListener(this);  
    }  
  
    @Override  
    public void onTimerFinished(View view) {  
        // User didn't cancel, perform the action  
    }  
  
    @Override  
    public void onTimerSelected(View view) {  
        // User canceled, abort the action  
    }
```

```
}
```

為了開始定時器，添加下面的代碼到你的[activity](#)中處理用戶選擇了某個操作的位置中：

```
// Two seconds to cancel the action  
mDelayedView.setTotalTimeMs(2000);  
// Start the timer  
mDelayedView.start();
```

## 顯示 Confirmation 動畫

為了顯示confirmation動畫當用戶在你的app中完成一個操作，創建一個從其他activities啓動ConfirmationActivity 的intent。你可以具體制定一種動畫到 intent extra 的 EXTRA\_ANIMATION\_TYPE：

- SUCCESS\_ANIMATION
- FAILURE\_ANIMATION
- OPEN\_ON\_PHONE\_ANIMATION

你也可以添加一條消息出現在 confirmation icon下面。

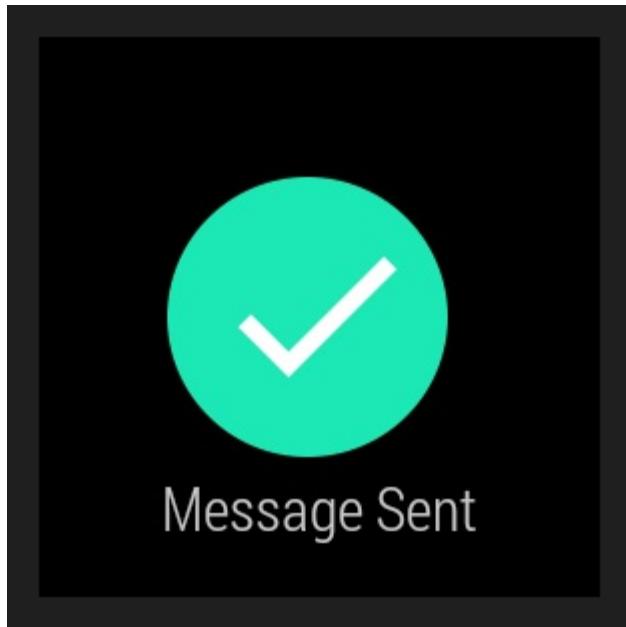


Figure 2: A confirmation animation.

要在你的app中使用 ConfirmationActivity，首先在你的manifest文件：

```
<manifest>
    <application>
        ...
        <activity
            android:name="android.support.wearable.activity.ConfirmationActivity">
        </activity>
    </application>
</manifest>
```

當用戶操作獲得結果，使用intent啓動activity:

```
Intent intent = new Intent(this, ConfirmationActivity.class);
intent.putExtra(ConfirmationActivity.EXTRA_ANIMATION_TYPE,
               ConfirmationActivity.SUCCESS_ANIMATION);
intent.putExtra(ConfirmationActivity.EXTRA_MESSAGE,
               getString(R.string.msg_sent));
startActivity(intent);
```

當confirmation動畫顯示結束。ConfirmationActivity finish然後你的activity resumes。

# 退出全屏的Activity

編寫: roya 原文:<https://developer.android.com/training/wearables/ui/exit.html>

默認情況下，用戶可以從左到右划動退出Android Wear activities。如果app含有水平水滾的內容，用戶首先滾動至內容邊緣然後再次從左到右滑動可以退出app。

對於更加沉浸式的體驗，比如在app中可以滾動地圖到任何位置，你可以在你的app中禁用滑動退出手勢，然而，如果你禁用了這個，你必須使用Wearable UI庫中的 DismissOverlayView 類實現long-press-to-dismiss UI讓用戶退出你的app。你當然也需要在用戶第一次運行你的app的時候提醒用戶可以長按退出app。

更多關於退出Android Wear activities的設計指南，請查看[Breaking out of the card](#)。

## 禁用Swipe-To-Dismis手勢

如果你的app要干涉滑動退出手勢的用戶交互模型，你在你的app中可以禁用它。為了在你的app中禁用滑動退出手勢，繼承默認的theme然後設置 android:windowSwipeToDismiss 屬性為 false：

```
<style name="AppTheme" parent="Theme.DeviceDefault">
    <item name="android:windowSwipeToDismiss">false</item>
</style>
```

如果你禁用了這個手勢，你需要實現長按退出UI元素以讓用戶可以退出你的app，類似下面章節的描述：

## 實現長按忽略元素

要在你的activity中使用 DismissOverlayView 類，添加這個元素到你的layout解釋文件，讓它全屏且覆蓋在所有其他view上，例子：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <!-- other views go here -->

    <android.support.wearable.view.DismissOverlayView
        android:id="@+id/dismiss_overlay"
        android:layout_height="match_parent"
        android:layout_width="match_parent"/>
<FrameLayout>
```

在你的activity中，取得 DismissOverlayView 元素然後設置一些提示文字。這些文字會在用戶第一次運行你的app時提醒用戶可以使用長按手勢退出app。接着用一個 GestureDetector 探測長按動作：

```
public class WearActivity extends Activity {

    private DismissOverlayView mDismissOverlay;
    private GestureDetector mDetector;

    public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.wear_activity);

// Obtain the DismissOverlayView element
mDismissOverlay = (DismissOverlayView) findViewById(R.id.dismiss_overlay);
mDismissOverlay.setIntroText(R.string.long_press_intro);
mDismissOverlay.showIntroIfNecessary();

// Configure a gesture detector
mDetector = new GestureDetector(this, new SimpleOnGestureListener() {
    public void onLongPress(MotionEvent ev) {
        mDismissOverlay.show();
    }
});

// Capture long presses
@Override
public boolean onTouchEvent(MotionEvent ev) {
    return mDetector.onTouchEvent(ev) || super.onTouchEvent(ev);
}
}
```

當系統發現長按動作， DismissOverlayView 會顯示一個退出按鈕，當用戶點擊它，你的[activity](#)會被終止。

# 發送並同步數據

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/index.html>

可穿戴數據層API(The Wearable Data Layer API)，Google Play services 的一部分，為你的手持與可穿戴應用提供了一個交流通道。此API包括一系列的數據對象，其可由系統通過網絡和能告知你應用數據層重要事件的監聽器發送並同步：

## Data Items

數據元提供了手持設備與可穿戴設備間的自動同步的數據儲存。

## Messages

MessageApi 類可以發送“自動跟蹤”命令消息，比如，從可穿戴設備上控制手持設備的媒體播放器，或在可穿戴設備上啓動一個來自手持設備的intent。當手持設備與可穿戴設備成功連接時，系統會發送該消息，否則，會發送一個錯誤。

## Asset

資源對象是為了發送如圖像這樣的二進制數據。將資源附加到數據元，系統會自動負責傳遞，並通過緩存大的資源來避免重複傳送以保護藍牙帶寬。

## WearableListenerService (for services)

拓展的 WearableListenerService 能夠監聽一個service中重要的數據層事件。系統控制WearableListenerService的生命週期，並當需要發送數據元或消息時，將其與service綁定，否則解除綁定。

## DataListener (for foreground activities)

在一個前臺activity中實現DataListener能夠監聽重要的數據通道事件。只有當用戶頻繁地使用應用時，用此代替 WearableListenerService來監聽改變。

Warning: 因為這些Api是為手持設備與可穿戴設備間通信設計，所以你只能使用這些Api來建立這些設備間的通信。例如，不能試着打開底層sockets來創建通信通道。

# Lessons

- [Accessing the Wearable Data Layer\(訪問可穿戴數據層\)](#)

這節課向你展示瞭如何創建一個客戶端訪問數據層Api。

- [Syncing Data Items\(同步數據單元\)](#)

數據元是存儲在一個複製而來的數據倉庫中的對象，該倉庫可自動由手持設備同步到可穿戴設備。

- [Transferring Assets\(傳輸資源\)](#)

資源是典型地用來傳輸圖像和媒體二進制數據。

- [Sending and Receiving Messages\(發送與接收消息\)](#)

消息被設計為自動跟蹤的消息，可以在手持與可穿戴設備間來回傳送。

- [Handling Data Layer Events\(處理數據層的事件\)](#)

獲知數據層的變化與事件。

# 訪問可穿戴數據層

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/accessing.html>

調用數據層API，需創建一個 [GoogleApiClient](#) 實例，所有 Google Play services APIs的主要入口點。

[GoogleApiClient](#) 提供了一個易於創建客戶端實例的builder。最簡單的[GoogleApiClient](#)如下：

Note: 目前，此小client僅足以能啓動。但是，更多創建GoogleApiClient，實現回調方法和處理錯誤等內容，詳見 [Accessing Google Play services APIs](#)。

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(new ConnectionCallbacks() {
        @Override
        public void onConnected(Bundle connectionHint) {
            Log.d(TAG, "onConnected: " + connectionHint);
            // 現在你可以使用Data Layer API 了
        }
        @Override
        public void onConnectionSuspended(int cause) {
            Log.d(TAG, "onConnectionSuspended: " + cause);
        }
    })
    .addOnConnectionFailedListener(new OnConnectionFailedListener() {
        @Override
        public void onConnectionFailed(ConnectionResult result) {
            Log.d(TAG, "onConnectionFailed: " + result);
        }
    })
    // 請求只訪問 Wearable API
    .addApi(Wearable.API)
    .build();
```

Important:為了避免在並沒有安裝 [Android Wear app](#) 的設備上，客戶端連接失敗，請使用獨立的[GoogleApiClient](#) 實例來僅訪問 [Wearable API](#)。更多的信息，請參見 [Access the Wearable API](#).

在你使用數據層API之前，通過調用[connect\(\)](#)方法進行連接，如 [Start a Connection](#)中所述。當系統為你的客戶端調用了[onConnected\(\)](#) 方法，你就可以使用數據層API了。

# 同步數據單元

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/data-items.html>

DataItem是指系統用於同步手持設備與可穿戴設備間數據的接口。一個DataItem通常包括以下幾點：

- Pyload- 一個字節數組，你可以用來設置任何數據，讓你的object序列化和反序列化。Pyload的大小限制在100k之內。
- Path- 唯一且以前斜線開頭的string (如：" /path/to/data")。

通常不直接實現DataItem，而是：

1. 創建一個PutdataRequest對象，指明一個string path 以唯一確定該 item。
2. 調用setData()方法設置Pyload。
3. 調用DataApi.putDataItem()方法，請求系統創建數據元。
4. 當請求的時候，系統會返回正確實現DataItem接口的對象。

然而，我們建議使用Data Map來顯示裝在一個易用的類似Bundle接口中的數據元，而用不是setData()來處理原始字節。

## 用 Data Map 同步數據

使用DataMap類，將數據元處理為 Android Bundle的形式，因此對象的序列化和反序列化就會完成，你就可以以 key-value 對的形式操縱數據。

如何使用：

1. 創建一個 PutDataMapRequest對象，設置數據元的path。

Note: path 字符串對數據元是唯一確定的，這樣能夠使你從另一連接端訪問。Path須以前斜線開始。如果你想在應用中使用分層數據，就要創建一個適合數據結構的路徑方案。
2. 調用PutDataMapRequest.getDataMap())獲取一個你可以使用的data map 對象。
3. 使用put...()方法，如：putString(),為data map設置數據。
4. 調用PutDataMapRequest.asPutDataRequest())獲得PutDataRequest對象。
5. 調用 DataApi.putDataItem() 請求系統創建數據元

Note: 如果手機和可穿戴設備沒有連接，數據會緩衝並在重新建立連接時同步。

接下的例子中的increaseCounter()方法展示瞭如何創建一個data map，並設置數據：

```
public class MainActivity extends Activity implements
    DataApi.DataListener,
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

    private static final String COUNT_KEY = "com.example.key.count";

    private GoogleApiClient mGoogleApiClient;
    private int count = 0;

    ...

    // 創建一個 data map 並設置數據
```

```

private void increaseCounter() {
    PutDataMapRequest putDataMapReq = PutDataMapRequest.create("/count");
    putDataMapReq.getDataMap().putInt(COUNT_KEY, count++);
    PutDataRequest putDataReq = putDataMapReq.asPutDataRequest();
    PendingResult<DataApi.DataItemResult> pendingResult =
        Wearable.DataApi.putDataItem(mGoogleApiClient, putDataReq);
}

...
}

```

有關控制 `PendingResult` 對象的更多信息，請參見 [Wait for the Status of Data Layer Calls](#)。

## 監聽數據元事件

如果一端的數據層的數據發生改變，想要在另一端被告知此改變，你可以通過實現一個數據元事件的監聽器來完成。

下面例子的代碼片段能夠通知你的app，當定義在上一個例子中的counter的值發生改變時。

```

public class MainActivity extends Activity implements
    DataApi.DataListener,
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

    private static final String COUNT_KEY = "com.example.key.count";

    private GoogleApiClient mGoogleApiClient;
    private int count = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();
    }

    @Override
    protected void onResume() {
        super.onStart();
        mGoogleApiClient.connect();
    }

    @Override
    public void onConnected(Bundle bundle) {
        Wearable.DataApi.addDataListener(mGoogleApiClient, this);
    }

    @Override
    protected void onPause() {
        super.onPause();
        Wearable.DataApi.removeDataListener(mGoogleApiClient, this);
        mGoogleApiClient.disconnect();
    }

    @Override
    public void onDataChanged(DataEventBuffer dataEvents) {
        for (DataEvent event : dataEvents) {
            if (event.getType() == DataEvent.TYPE_CHANGED) {
                // DataItem 變更了
                DataItem item = event.getDataItem();
                if (item.getUri().getPath().compareTo("/count") == 0) {
                    DataMap dataMap = DataMapItem.fromDataItem(item).getDataMap();
                    updateCount(dataMap.getInt(COUNT_KEY));
                }
            }
        }
    }
}

```

```
        }
    } else if (event.getType() == DataEvent.TYPE_DELETED) {
        // DataItem 削除了
    }
}

// 我們的更新 count 的方法
private void updateCount(int c) { ... }

...
}
```

這個activity是實現了 [DataItem.DataListener](#) 接口，並在onConnected()方法中增加自身成為數據元事件的監聽者，和在onPause()方法中移除監聽。

你也可以用一個service實現監聽，請見 [Listening for Data Layer Events](#)。

# 傳輸資源

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/assets.html>

為了通過藍牙發送大量的二進制數據，比如圖片，要將一個Asset附加到數據元上，並放入複製而來的數據庫中。

Assets 能夠自動地處理數據緩存以避免重複發送，保護藍牙帶寬。一般的模式是：手持設備下載圖像，並將它壓縮到適合在可穿戴設備上顯示的大小，並以Asset傳給可穿戴設備。下面的例子演示此模式。

Note: 儘管數據元的大小限制在100KB,但資源可以任意大。然而，傳輸大量資源會多方面地影響用戶體驗，因此，當你要傳輸大量資源時，要測試你的應用以保證它表現良好。

## 傳輸資源

在Asset類中使用creat..()方法創建資源。下面，我們將一個bitmap轉化為字節流，然後調用creatFromBytes())方法創建資源。

```
private static Asset createAssetFromBitmap(Bitmap bitmap) {
    final ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, byteStream);
    return Asset.createFromBytes(byteStream.toByteArray());
}
```

創建資源後，使用DataMap或者PutDataRepuest類中的putAsset()方法將其附加到數據元上，然後用putDataItem()方法將數據元放入數據庫。

### 使用 PutDataRequest

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);
Asset asset = createAssetFromBitmap(bitmap);
PutDataRequest request = PutDataRequest.create("/image");
request.putAsset("profileImage", asset);
Wearable.DataApi.putDataItem(mGoogleApiClient, request);
```

### 使用 PutDataMapRequest

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);
Asset asset = createAssetFromBitmap(bitmap);
PutDataMapRequest dataMap = PutDataMapRequest.create("/image");
dataMap.getDataMap().putAsset("profileImage", asset)
PutDataRequest request = dataMap.asPutDataRequest();
PendingResult<DataApi.DataItemResult> pendingResult = Wearable.DataApi
    .putDataItem(mGoogleApiClient, request);
```

## 接收資源

創建資源後，你可能需要在另一連接端讀取。以下是如何實現回調以發現資源變化和提取Asset對象。

```
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
    for (DataEvent event : dataEvents) {
        if (event.getType() == DataEvent.TYPE_CHANGED &&
            event.getDataItem().getUri().getPath().equals("/image")) {
            DataMapItem dataMapItem = DataMapItem.fromDataItem(event.getDataItem());
            Asset profileAsset = dataMapItem.getDataMap().getAsset("profileImage");
            Bitmap bitmap = loadBitmapFromAsset(profileAsset);
            // Do something with the bitmap
        }
    }
}

public Bitmap loadBitmapFromAsset(Asset asset) {
    if (asset == null) {
        throw new IllegalArgumentException("Asset must be non-null");
    }
    ConnectionResult result =
        mGoogleApiClient.blockingConnect(TIMEOUT_MS, TimeUnit.MILLISECONDS);
    if (!result.isSuccess()) {
        return null;
    }
    // convert asset into a file descriptor and block until it's ready
    InputStream assetInputStream = Wearable.DataApi.getFdForAsset(
        mGoogleApiClient, asset).await().getInputStream();
    mGoogleApiClient.disconnect();

    if (assetInputStream == null) {
        Log.w(TAG, "Requested an unknown Asset.");
        return null;
    }
    // decode the stream into a bitmap
    return BitmapFactory.decodeStream(assetInputStream);
}
```

# 發送與接收消息

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/messages.html>

使用MessageApi發送消息，要附加以下幾項：

- 任一payload(可選);
- 唯一確定的message's action 的 path。

不像數據元，Messages(消息)在手持和可穿戴應用之間沒有同步。Messages是單向交流機制，這有利於遠程進程調用(RPC)，比如：發送消息到可穿戴設備以開啓activity。

## 發送消息

下面的例子展示如何發送消息到另一連接端開啓一個activity。調用是同步的，當收到消息或請求超時時發生阻塞。

Note: 閱讀 [Communicate with Google Play Services](#) 瞭解更多關於異步和同步調用，以及何時使用哪個。

```
GoogleApiClient mGoogleApiClient;
public static final String START_ACTIVITY_PATH = "/start/MainActivity";
...

private void sendStartActivityMessage(String nodeId) {
    Wearable.MessageApi.sendMessage(
        mGoogleApiClient, nodeId, START_ACTIVITY_PATH, new byte[0]).setResultCallback(
        new ResultCallback<SendMessageResult>() {
            @Override
            public void onResult(SendMessageResult sendMessageResult) {
                if (!sendMessageResult.getStatus().isSuccess()) {
                    Log.e(TAG, "Failed to send message with status code: "
                        + sendMessageResult.getStatus().getStatusCode());
                }
            }
        });
}
```

這是一個簡單的方法，來獲得一列你可能發送消息給它們的連接點：

```
private Collection<String> getNodes() {
    HashSet<String> results = new HashSet<String>();
    NodeApi.GetConnectedNodesResult nodes =
        Wearable.NodeApi.getConnectedNodes(mGoogleApiClient).await();
    for (Node node : nodes.getNodes()) {
        results.add(node.getId());
    }
    return results;
}
```

## 接收消息

為了在收到消息時被提醒，你可以實現 [MessageListener](#)接口來提供消息事件的監聽，你需要在 [MessageApi.addListener\(\)](#)方法中註冊監聽。這個例子展示你可以通過檢查 上例中發送消息時使用到的 START\_ACTIVITY\_PATH的狀態，若是true,特定的activity就會啓動。

```
@Override  
public void onMessageReceived(MessageEvent messageEvent) {  
    if (messageEvent.getPath().equals(START_ACTIVITY_PATH)) {  
        Intent startIntent = new Intent(this, MainActivity.class);  
        startIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
        startActivity(startIntent);  
    }  
}
```

這僅是實現更多細節的一小段，如何在service 或 `activity` 實現完整的監聽，請參見 [Listening for Data Layer Events](#)。

# 處理數據層的事件

編寫:wly2014 - 原文: <http://developer.android.com/training/wearables/data-layer/events.html>

當做出數據層上的調用時，你可以得到它完成後的調用狀態，也可以用監聽器監聽到調用最終實現的改變。

## 等待數據層調用狀態

注意到，調用數據層API，有時會返回PendingResult，如 `putDataItem()`。PendingResult一被創建，操作就會在後臺排列等候。之後你若無動作，這些操作最終會默默完成。然而，通常要處理操作完成後的結果，PendingResult能夠讓你同步或異步地等待結果。

### 異步等待

若你的代碼運行在主UI線程上，不使阻塞調用數據層API。你可以增加一個PendingResult對象回調來運行異步調用，將在操作完成時觸發。

```
pendingResult.setResultCallback(new ResultCallback<DataItemResult>() {
    @Override
    public void onResult(final DataItemResult result) {
        if(result.getStatus().isSuccess()) {
            Log.d(TAG, "Data item set: " + result.getDataItem().getUri());
        }
    }
});
```

### 同步等待

如果你的代碼是運行在後臺服務的一個單獨的處理線程上（`WearableListenerService`的情況），則適合調用阻塞。在這種情況下，你可以用PendingResult對象調用`await()`，它將阻塞至請求完成，並返回一個Result對象。

```
DataItemResult result = pendingResult.await();
if(result.getStatus().isSuccess()) {
    Log.d(TAG, "Data item set: " + result.getDataItem().getUri());
}
```

## 監聽數據層事件

因為數據層在手持和可穿戴設備間同步並發送數據，所以通常要監聽重要事件，例如創建數據元，接受消息，或當可穿戴設備和手機連接時。

對於監聽數據層事件，有兩種選擇：

- 創建一個繼承自`WearableListenerService`的service。
- 創建一個實現`DataApi.DataListener`接口的activity。

通過這兩種選擇，你覆寫任何你關心的數據事件回調方法來處理您的實現。

## 使用 WearableListenerService

典型地，在你的手持設備和可穿戴設備上都創建該service實例。如果你不關心其中一個應用中的數據事件，就不需要在相應的應用中實現此service。

例如，您可以在一個手持設備應用程序上操作數據元對象，可穿戴設備的應用監聽這些更新並更新UI。而可穿戴不更新任何數據元，所以手持設備的應用不監聽任何可穿戴式設備應用的數據事件。

你可以用 WearableListenerService 監聽如下事件：

- `onDataChanged()` - 當數據元對象創建，更改，刪除時調用。一連接端的事件將觸發兩端的回調方法。
- `onMessageReceived()` - 消息從一連接端發出時在另一連接端觸發此回調方法。
- `onPeerConnected()` 和 `onPeerDisconnected()` - 當與手持或可穿戴設備連接或斷開時調用。一連接端連接狀態的改變會在兩端觸發此回調方法。

創建WearableListenerService：

1. 創建一個繼承自WearableListenerService的類。
2. 監聽你關心的事件，比如`onDataChanged()`。
3. 在Android manifest中聲明一個intent filter，通知系統你的WearableListenerService。這樣允許系統需要時綁定你的service。

下例展示如何實現一個簡單的WearableListenerService：

```
public class DataLayerListenerService extends WearableListenerService {

    private static final String TAG = "DataLayerSample";
    private static final String START_ACTIVITY_PATH = "/start-activity";
    private static final String DATA_ITEM_RECEIVED_PATH = "/data-item-received";

    @Override
    public void onDataChanged(DataEventBuffer dataEvents) {
        if (Log.isLoggable(TAG, Log.DEBUG)) {
            Log.d(TAG, "onDataChanged: " + dataEvents);
        }
        final List events = FreezableUtils
            .freezeIterable(dataEvents);

        GoogleApiClient googleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .build();

        ConnectionResult connectionResult =
            googleApiClient.blockingConnect(30, TimeUnit.SECONDS);

        if (!connectionResult.isSuccess()) {
            Log.e(TAG, "Failed to connect to GoogleApiClient.");
            return;
        }

        // Loop through the events and send a message
        // to the node that created the data item.
        for (DataEvent event : events) {
            Uri uri = event.getDataItem().getUri();

            // Get the node id from the host value of the URI
            String nodeId = uri.getHost();
            // Set the data of the message to be the bytes of the URI
            byte[] payload = uri.toString().getBytes();

            // Send the RPC
            Wearable.MessageApi.sendMessage(googleApiClient, nodeId,
                DATA_ITEM_RECEIVED_PATH, payload);
        }
    }
}
```

```
    }
}
```

這是Android mainfest中相應的intent filter：

```
<service android:name=".DataLayerListenerService">
<intent-filter>
    <action android:name="com.google.android.gms.wearable.BIND_LISTENER" />
</intent-filter>
</service>
```

## 數據層回調權限

為了在數據層事件上向你的程序提供回調方法，Google Play services 綁定到你的WearableListenerService，通過IPC調用回調方法。這樣的結果是，你的回調方法繼承了調用進程的權限。

如果你想在一個回調中執行權限操作，安全檢查會失敗，因為你的回調是以調用進程的身份運行，而不是應用程序進程的身份運行。

為瞭解決這個問題，在進入IPC後使用[clearCallingIdentity\(\)](#)重置身份，當你完成權限操作後，使用[restoreCallingIdentity\(\)](#)恢復身份：

```
long token = Binder.clearCallingIdentity();
try {
    performOperationRequiringPermissions();
} finally {
    Binder.restoreCallingIdentity(token);
}
```

## 使用監聽者 Activity

如果你的應用只關心當用戶與應用交互時產生的數據層事件，並且不需要一個長時間運行的service來處理每一次數據的改變，那麼你可以在一個activity中通過實現如下一個或多個接口監聽事件：

- [DataApi.DataListener](#)
- [MessageApi.MessageListener](#)
- [NodeApi.NodeListener](#)

如何創建一個activity監聽數據事件：

- 實現所需的接口。
- 在onCreate(Bundle)中創建 GoogleApiClient實例。
- 在onStart()中調用connect() 將客戶端連接到 Google Play services。
- 當連接到Google Play services後，系統調用 onConnected()。這裏你調用[DataApi.addListener\(\)](#), [MessageApi.addListener\(\)](#), 或 [NodeApi.addListener\(\)](#)以告知Google Play services，你的activity要監聽數據層事件。
- 在 onStop()中，用 [DataApi.removeListener\(\)](#), [MessageApi.removeListener\(\)](#)或[NodeApi.removeListener\(\)](#) 註銷監聽。
- 基於你實現的接口繼而實現 [onDataChanged\(\)](#), [onMessageReceived\(\)](#), [onPeerConnected\(\)](#)和 [onPeerDisconnected\(\)](#)。

這是實現DataApi.DataListener的例子：

```
public class MainActivity extends Activity implements
    DataApi.DataListener, ConnectionCallbacks, OnConnectionFailedListener {

    private GoogleApiClient mGoogleApiClient;

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

        setContentView(R.layout.main);
        mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();
    }

    @Override
    protected void onStart() {
        super.onStart();
        if (!mResolvingError) {
            mGoogleApiClient.connect();
        }
    }

    @Override
    public void onConnected(Bundle connectionHint) {
        if (Log.isLoggable(TAG, Log.DEBUG)) {
            Log.d(TAG, "Connected to Google Api Service");
        }
        Wearable.DataApi.addListener(mGoogleApiClient, this);
    }

    @Override
    protected void onStop() {
        if (null != mGoogleApiClient && mGoogleApiClient.isConnected()) {
            Wearable.DataApi.removeListener(mGoogleApiClient, this);
            mGoogleApiClient.disconnect();
        }
        super.onStop();
    }

    @Override
    public void onDataChanged(DataEventBuffer dataEvents) {
        for (DataEvent event : dataEvents) {
            if (event.getType() == DataEvent.TYPE_DELETED) {
                Log.d(TAG, "DataItem deleted: " + event.getDataItem().getUri());
            } else if (event.getType() == DataEvent.TYPE_CHANGED) {
                Log.d(TAG, "DataItem changed: " + event.getDataItem().getUri());
            }
        }
    }
}
```

# Android TV應用

---

編寫:applepyarc - 原文:<http://developer.android.com/training/tv/index.html>

以下課程將教你如何為Android TV設備開發應用。

Note : 如何在Google Play發佈你的TV應用，請參考[Distributing to Android TV](#)。

## 創建TV應用

如何開發TV應用和移植已有應用到TV設備。

## 創建TV播放應用

如何開發提供媒體目錄和播放內容的應用。

## 幫助用戶在TV上找到內容

如何幫助用戶從你的應用發現所需內容。

## 創建TV遊戲應用

如何開發TV遊戲。

## 創建TV直播應用

如何開發TV直播應用。

## TV應用清單

TV應用的需求清單

# 創建TV應用

編寫:[applepyarc](#) - 原文:<http://developer.android.com/training/tv/start/index.html>

- Android 5.0(API level 21)或以上
- Android Studio 0.8或以上，Gradle 0.12或以上

Android提供豐富的用戶體驗，優化應用運行於諸如高清電視等大屏幕設備。TV應用讓用戶更開心，生活更美好。

TV應用使用與手機或平板應用相同的架構。這意味着你可以基於已知的Android應用開發來創建新的TV應用。或者移植已有的應用到TV設備上。但是，在UI上，TV和手機或平板大不相同。為了使應用順暢地運行在TV設備上，你必須設計能夠在即使3米之外也易於理解的新界面，提供可以使用方向鍵和選擇鍵操作的導航功能。

以下課程描述瞭如何開始創建TV應用，包括設置開發環境，界面及導航的基本要求，以及如何處理TV設備通常不具備的硬件特性。

Note: 鼓勵使用[Android Studio](#)創建TV應用，因為它提供了創建項目，包含庫和快捷打包。本課程假設你正在使用Android Studio。

## 課程

- [創建TV應用的第一步](#)

學習如何為要運行在TV設備上的已有應用創建一個新的Android Studio項目。

- [創建TV佈局](#)

學習TV界面的最小要求及其實現。

- [創建TV導航](#)

學習TV導航的需求以及如何實現TV兼容的導航。

- [處理TV硬件](#)

學習如何檢查應用是否運行在TV硬件上，處理不支持的硬件特性和管理控制器設備。

---

開始

# 創建TV應用的第一步

編寫:[awong1900](#) - 原文:<http://developer.android.com/training/tv/start/start.html>

TV應用使用與手機和平板同樣的架構。這種相似性意味着你可以修改現有的應用到TV設備或者用以前安卓應用的經驗開發TV應用。

Important: 在Google Play中的TV應用應滿足一些特定要求。更多信息, 參考[TV App Quality](#)中的要求列表。

本課程介紹如何準備您的TV應用開發環境,和使應用能夠運行在TV設備上所必須的最小改變。

## 創建TV項目

本節討論如何修改已有的應用或者新建一個應用使之能夠運行在電視設備上。在TV設備運行的應用必須使用這些主要組件:

- [Activity for TV \(必須\)](#) - 在您的application manifest中, 聲明一個可在TV設備上運行的activity。
- [TV Support Libraries \(可選\)](#) - 這些支持庫Support Libraries 可以提供搭建TV用戶界面的控件。

## 前提條件

在創建TV應用前, 必須做以下事情:

- [更新SDK tools到版本24.0.0或更高](#) 最新的SDK tools能確保編譯和測試TV應用
- [更新SDK為Android 5.0 \(API 21\)或更高](#) 最新的platform版本提供TV應用的新API
- [創建或更新應用項目](#) 為了支持TV新API, 你必須設置新項目或者修改原項目的目標平臺為Android 5.0 (API level 21)或者更高。

## 聲明一個TV Activity

一個應用想要運行在TV設備中, 必須在它的manifest中定義一個啓動activity, 用intent filter包含[CATEGORY\\_LEANBACK\\_LAUNCHER](#)。這個filter表明你的應用是在TV上可用, 並且為Google Play上發佈TV應用所必須。定義這個intent也意味着在點擊app在TV主屏幕的圖標時, 打開的就是這個activity。

接下來的代碼片段顯示如何在manifest中包含這個intent filter :

```
<application
    android:banner="@drawable/banner" >
    ...
    <activity
        android:name="com.example.android.MainActivity"
        android:label="@string/app_name" >

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

    <activity
        android:name="com.example.android.TvActivity"
        android:label="@string/app_name"
```

```
    android:theme="@style/Theme.Leanback">

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

</activity>
</application>
```

例子中第二個[activity](#)的manifest是TV設備中的一個啓動入口。

**Caution**：如果在你的應用中不包含[CATEGORY\\_LEANBACK\\_LAUNCHER](#) intent filter，它不會出現在TV設備的Google Play商店中。並且，即使你把不包含此filter的應用用開發工具裝載到TV設備中，應用仍然不會出現在TV用戶界面上。

如果你正在為TV設備修改現有的應用，不應該與手機和平板用同樣的[activity](#)佈局。TV的用戶界面（或者現有應用的TV部分）應該提供一個更簡單的界面，更容易坐在沙發上用遙控器操作。TV應用設計指南，參考[TV Design](#)指導。查看TV界面佈局的最低要求，參考：[Building TV Layouts](#)。

## 聲明Leanback支持

安卓TV需要你的應用使用Leanback用戶界面。如果你正在開發一個運行在移動設備（手機，可穿戴，平板等等）同時也包含安卓TV的應用，設置 `required` 屬性為 `false`。因為如果設置為 `true`，你的應用將僅能運行在用Leanback UI的設備上。

```
<manifest>
    <uses-feature android:name="android.software.leanback"
                  android:required="false" />
    ...
</manifest>
```

## 聲明不需要觸屏

運行在TV設備上的應用不依靠觸屏去輸入。為了清楚表明這一點，TV應用的manifest必須聲明 `android.hardware.touchscreen` 為不需要。這個設置表明應用能夠工作在TV設備上，並且也是Google Play認定你的應用為TV應用的要求。接下來的示例代碼展示這個manifest聲明：

```
<manifest>
    <uses-feature android:name="android.hardware.touchscreen"
                  android:required="false" />
    ...
</manifest>
```

**Caution**：必須在manifest中聲明觸屏是不需要的，否則應用不會出現在TV設備的Google Play商店中。

## 提供一個主屏幕橫幅

如果應用包含一個Leanback的intent filter，它必須提供每個語言的主屏幕橫幅。橫幅是出現在應用和遊戲欄的主屏的啓動點。在manifest中這樣描述橫幅：

```
<application
    ...
    android:banner="@drawable/banner" >
```

```
...
```

```
</application>
```

在 `application` 中添加 `android:banner` 屬性為所有的應用 `activity` 提供默認的橫幅，或者在特定的 `activity` 的 `activity` 中添加橫幅。

在 UI 模式和 TV 設計指導中查看 [Banners](#)。

## 添加TV支持庫

Android SDK 包含用於 TV 應用的支持庫。這些庫為 TV 設備提供 API 和用戶界面控件。這些庫位於 `<sdk>/extras/android/support/` 目錄。以下是這些庫的列表和它們的作用介紹：

- [v17 leanback library](#) - 提供 TV 應用的用戶界面控件，特別是用於媒體播放應用的。
- [v7 recyclerview library](#) - 提供了內存高效方式的長列表的管理顯示類。有一些 v17 leanback 庫的類依賴於本庫的類。
- [v7 cardview library](#) - 提供顯示信息卡的用戶界面控件，如媒體圖片和描述。

Note : TV 應用中可以不用這些庫。但是，我們強烈推薦你使用它們，特別是為應用提供媒體目錄瀏覽界面時。

如果你決定用 `v17 leanback library`，你應該注意它依賴於 [v4 support library](#)。這意味著要用 leanback 支持庫必須包含以下所有的支持庫：

- `v4 support library`
- `v7 recyclerview support library`
- `v17 leanback support library`

`v17 leanback library` 包含資源文件，需要你在應用中採取特定的步驟去包含它。插入帶資源文件的支持庫的說明，查看 [Support Library Setup](#)。

## 創建TV應用

在完成上面的步驟之後，到了給大屏幕創建應用的時候了！檢查一下這些額外的專題可以幫助您創建 TV 應用：

- [創建TV播放應用](#) - TV 主要是用來娛樂，因此安卓提供了一套用戶界面工具和控件，用來創建視頻和音樂的 TV 應用，並且讓用戶瀏覽想看到的內容。
- [幫助用戶找到TV內容](#) - 因為所有的內容選擇操作都用手指操作遙控器，所以幫助用戶找到想要的內容幾乎和提供內容同樣重要。這個主題討論如何在 TV 設備中處理內容。
- [TV遊戲](#) - TV 設備是非常好的遊戲平臺。參考這個主題去創造更好的 TV 遊戲體驗。

## 運行TV應用

在開發過程中運行應用是一個重要的部分。在安卓 SDK 中的 AVD 管理器提供了創建虛擬 TV 設備的功能，可以讓應用在虛擬設備中運行和測試。

### 創建一個虛擬 TV 設備

1. 打開 AVD 管理器。更多信息，參考 [AVD 管理器](#) 幫助。
2. 在 AVD 管理器窗口，點擊 `Device Definitions` 標籤。

3. 選擇安卓一個TV設備定義，並且點擊Create AVD。
4. 選擇模擬器選項並且點擊OK創建AVD。

Note：獲得TV模擬器設備的最佳性能，打開Use Host GPU option，支持虛擬設備加速。更多模擬器硬件加速信息，參考[Using the Emulator](#)。

#### 在虛擬設備中測試應用

1. 在開發環境中編譯TV應用。
2. 從開發環境中運行應用並選擇目標為TV虛擬設備。

更多模擬器信息：[Using the Emulator](#)。用Android Studio部署應用到模擬器，查看[Debugging with Android Studio](#)。用帶ADT插件的Eclipse部署應用到模擬器，查看[Building and Running from Eclipse with ADT](#)。

---

[下一節: 處理TV硬件](#)

# 處理TV硬件

編寫:[awong1900](#) - 原文:<http://developer.android.com/training/tv/start/hardware.html>

TV硬件和其他安卓設備有實質性的不同。TV不包含一些其他安卓設備具備的硬件特性，如觸摸屏，攝像頭，和GPS。TV也完全依賴於其他輔助硬件設備。為了讓用戶與TV應用交互，他們必須使用遙控器或者遊戲手柄。當您創建TV應用時，必須小心的考慮到TV硬件的限制和操作要求。

本節課程討論如何檢查應用是不是運行在TV上，怎樣去處理不支持的硬件特性，和討論處理TV設備控制器的要求。

## TV設備的檢測

如果你創建的應用同時支持TV設備和其他設備，你可能需要檢測應用當前運行在哪種設備上，並調整應用的執行。例如，如果你有一個應用通過Intent啓動，你的應用應該檢查設備特性然後決定是應該啓動TV導向的activity還是手機的activity。

檢查應用是否運行在TV設備上，推薦的方式是用UiModeManager.getCurrentModeType()方法檢測設備是否運行在TV模式。下面的示例代碼展示瞭如何檢查你的應用是否運行在TV設備上：

```
public static final String TAG = "DeviceTypeRuntimeCheck";

UiModeManager uiModeManager = (UiModeManager) getSystemService(UI_MODE_SERVICE);
if (uiModeManager.getCurrentModeType() == Configuration.UI_MODE_TYPE_TELEVISION) {
    Log.d(TAG, "Running on a TV Device")
} else {
    Log.d(TAG, "Running on a non-TV Device")
}
```

## 處理不支持的硬件特性

基於設計和應用的功能，你可能需要在某些硬件特性不可用的情況下工作。這節討論哪些硬件特性對於TV是典型不可用的，如何去檢測缺少的硬件特性，並且建議替代方法去用這些特性。

### 不支持的TV硬件特性

TV有不同於其他設備的用途，因此它們沒有一些其他安卓設備通常有的硬件特性。由於這個原因，TV設備的安卓系統不支持以下特性：

硬件	安卓特性描述
觸屏	android.hardware.touchscreen
觸屏模擬器	android.hardware.faketouch
電話	android.hardware.telephony
攝像頭	android.hardware.camera
藍牙	android.hardware.bluetooth
近場通訊（NFC）	android.hardware.nfc
GPS	android.hardware.location.gps

麥克風 [1]	android.hardware.microphone
傳感器	android.hardware.sensor

[1] 一些TV控制器有麥克風，但不是這裏描述的麥克風硬件特性。控制器麥克風是完全被支持的。

查看[Features Reference](#)獲得完全的特性列表，子特性，和他們的描述。

## 聲明TV硬件需求

安卓應用能通過在manifest中定義硬件特性需求來確保應用不能被安裝在不提供這些特性的設備上。如果你正在擴展應用到TV上，仔細地審查你的manifest的硬件特性需求，它有可能阻止應用安裝到TV設備上。

即使你的應用使用了TV上不存在的硬件特性（如觸屏或者攝像頭），應用也可以在沒有那些特性的情況下工作，需要修改應用的manifest來表明這些特性不是必須的。接下來的manifest代碼片段示範瞭如何聲明在TV設備中不可用的硬件特性，儘管你的應用在非TV設備上可能會用上這些特性。

```
<uses-feature android:name="android.hardware.touchscreen"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.faketouch"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.telephony"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.camera"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.bluetooth"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.nfc"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.gps"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.microphone"
    android:required="false"></uses>
<uses-feature android:name="android.hardware.sensor"
    android:required="false"></uses>
```

Note : 一些特性有子特性，如 `android.hardware.camera.front`，參考：[Feature Reference](#)。確保應用中任何子特性也標記為 `required="false"`。

所有想用在TV設備上的應用必須聲明觸屏特性不被需要，在[創建TV應用的第一步](#)有描述。如果你的應用使用了一個或更多的上面列表上的特性，改變manifest特性的 `android:required` 屬性為 `false`。

Caution : 表明一個硬件特性是必須的，設置它的值為 `true` 可以阻止應用在TV設備上安裝或者出現在安卓TV的主屏幕launcher上。

一旦你決定了應用的硬件特性選項，你必須檢查在運行時那些特性的可用性，然後調整你的應用的行為。下一節討論如何檢查硬件特性和改變應用行為的建議處理。

更多關於filter和在manifest裏聲明特性，參考：[uses-feature](#)。

## 聲明權限會隱含硬件特性

一些[uses-permission](#) manifest聲明隱含了硬件特性。這些行爲意味着在應用中請求一些權限能導致應用不能安裝和使用在TV設備上。下面普通的權限請求包含了一個隱式的硬件特性需求：

權限	隱式的硬件需求
<code>RECORD_AUDIO</code>	<code>android.hardware.microphone</code>

CAMERA	android.hardware.camera and android.hardware.camera.autofocus
ACCESS_COARSE_LOCATION	android.hardware.location and android.hardware.location.network
ACCESS_FINE_LOCATION	android.hardware.location and android.hardware.location.gps

包含隱式硬件特性需求的完整權限需求列表，參考：[uses-feature](#)。如果你的應用請求了上面列表上的特性的任何一個，在manifest中設置它的隱式硬件特性為不需要（`android:required="false"`）。

## 檢查硬件特性

在應用運行時，Android framework能告訴你硬件特性是否可用。用[hasSystemFeature\(String\)](#)方法在運行時檢查特定的特性。這個方法只需要一個字符串參數，即想檢查的特性名字。

接下來的示例代碼展示瞭如何在隕石檢測硬件特性的可用性：

```
// Check if the telephony hardware feature is available.  
if (getPackageManager().hasSystemFeature("android.hardware.telephony")) {  
    Log.d("HardwareFeatureTest", "Device can make phone calls");  
}  
  
// Check if android.hardware.touchscreen feature is available.  
if (getPackageManager().hasSystemFeature("android.hardware.touchscreen")) {  
    Log.d("HardwareFeatureTest", "Device has a touch screen.");  
}
```

## 觸屏

因為大部分的TV沒有觸摸屏，在TV設備上，安卓不支持觸屏交互。此外，用觸屏交互和坐在離顯示器3米外觀看是相互矛盾的。

在TV設備中，你應該設計應用交互模式支持遙控器上的方向鍵（D-pad）操作。更多關於正確地支持TV友好的控制器操作的信息，參考[Creating TV Navigation](#)。

## 攝像頭

儘管TV通常沒有攝像頭，但是你仍然可以提供拍照相關的TV應用，如果應用有拍照，查看和編輯圖片功能，在TV上可以關閉拍照功能但仍可以允許用戶查看甚至編輯圖片。如果你決定在TV上使用攝像相關的應用，在manifest裏添加接下來的特性聲明：

```
<uses-feature android:name="android.hardware.camera" android:required="false" ></uses>
```

如果在缺少攝像頭情況下運行你的應用，在你的應用中添加代碼去檢測是否攝像頭特性可用，並且調整應用的操作。接下來的示例代碼展示瞭如何檢測一個攝像頭的存在：

```
// Check if the camera hardware feature is available.  
if (getPackageManager().hasSystemFeature("android.hardware.camera")) {  
    Log.d("Camera test", "Camera available!");  
} else {  
    Log.d("Camera test", "No camera available. View and edit features only.");  
}
```

## GPS

TV是固定的室內設備，並且沒有內置的全球定位系統（GPS）接收器。如果你的應用使用定位信息，你仍可以允許用戶搜索位置，或者用固定位置提供商代替，如在TV設置中設置郵政編碼。

```
// Request a static location from the location manager
LocationManager locationManager = (LocationManager) this.getSystemService(
    Context.LOCATION_SERVICE);
Location location = locationManager.getLastKnownLocation("static");

// Attempt to get postal or zip code from the static location object
Geocoder geocoder = new Geocoder(this);
Address address = null;
try {
    address = geocoder.getFromLocation(location.getLatitude(),
        location.getLongitude(), 1).get(0);
    Log.d("Zip code", address.getPostalCode());
} catch (IOException e) {
    Log.e(TAG, "Geocoder error", e);
}
```

## 處理控制器

TV設備需要輔助硬件設備與應用交互，如一個基本形式的遙控器或者遊戲手柄。這意味着你的應用必須支持D-pad（十字方向鍵）輸入。它也意味着你的應用可能需要處理手柄掉線和更多類型的手柄輸入。

### D-pad必須的按鍵

默認的TV設備控制器是D-pad。通常，你可以用遙控器的上，下，左，右，選擇，返回，和Home鍵操作應用。如果你的應用是一個遊戲而需要遊戲手柄額外的控制，你的應用也應該嘗試允許用D-pad操作。這種情況下，你的應用也應該警告用戶需要手柄，並且允許他們用D-pad優雅的退出遊戲。更多關於在TV設備如理D-pad的操作，參考[Creating TV Navigation](#)。

### 處理手柄掉線

TV的手柄通常是藍牙設備，它為了省電而定期的休眠並且與TV設備斷開連接。這意味着如果不處理這些重連事件，應用可能被中斷或者重新開始。這些事件可以發生在下面任何情景中：

- 當在看幾分鐘的視頻，D-Pad或者遊戲手柄進入了睡眠模式，從TV設備上斷開連接並且隨後重新連接。
- 在玩遊戲時，新玩家用不是當前連接的遊戲手柄加入遊戲。
- 在玩遊戲時，一個玩家離開遊戲並且斷開遊戲手柄。

任何TV應用[activity](#)受制於斷開連接和重連事件，這些必須在manifest中配置為處理重連事件。。接下來的示例代碼展示瞭如何打開一個[activity](#)去處理配置改變，包括鍵盤或者操作設備連接，斷開連接，或者重新連接：

```
<activity
    android:name="com.example.android.TvActivity"
    android:label="@string/app_name"
    android:configChanges="keyboard|keyboardHidden|navigation"
    android:theme="@style/Theme.Leanback">

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

```
</activity>
```

這個配置改變允許應用通過重連事件繼續運行，比較而言Android framework強制重啓應用會導致一個不好的用戶體驗。

## 處理D-pad變種輸入

TV設備用戶可能有超過一種類型的控制器來操縱TV。例如，一個用戶可能有基本D-pad控制器和一個遊戲控制器。遊戲控制器用於D-pad功能的按鍵代碼可能和物理十字鍵提供的不相同。

你的應用應該要處理遊戲控制器D-pad的變種輸入，因此用戶不需要通過物理開關控制器去操作你的應用。更多信息關於處理這些變種輸入，參考[Handling Controller Actions](#)。

---

[下一節: 創建TV佈局](#)

# 創建TV佈局

編寫:awong1900 - 原文:<http://developer.android.com/training/tv/start/layouts.html>

TV通常在3米外觀看，並且它比大部分安卓設備大的多。TV屏幕沒有達到類似小設備的精細細節和顏色的水平。這些因素需要你在頭腦中考慮，並設計出對於TV設備更為有用且好的用戶體驗的應用佈局。

這節課程描述了創建有效的TV應用佈局的基本要求和實現細節。

## 用TV佈局主題

安卓主題能給你的TV應用佈局提供基礎框架。對於打算在TV設備上運行的應用activity，你應該用一款主題改變它的顯示。這節課程教你應該用哪個主題。

### Leanback主題

支持TV用戶界面的庫叫做v17 leanback libary，它提供了一個標準的TV activity主題，叫做 Theme.Leanback。這一主題為TV應用程序建立了一致的視覺風格。強烈推薦在任何用了v17 leanback類的TV應用中使用這個主題。接下來的代碼展示如何在應用中對給定的activity使用這個主題：

```
<activity
    android:name="com.example.android.TvActivity"
    android:label="@string/app_name"
    android:theme="@style/Theme.Leanback">
```

### NoTitleBar主題

在手機和平板的安卓應用中，標題欄是標準的用戶界面元素。但是在TV應用中是不適合的。如果沒有用v17 leanback類，你應該在TV activity使用這個主題來隱去標題欄的顯示。接下來的TV應用manifest代碼示範瞭如何應用這個主題來刪除標題欄。

```
<application>
    ...
    <activity
        android:name="com.example.android.TvActivity"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.NoTitleBar">
    ...
</activity>
</application>
```

## 創建基本的TV佈局

TV設備的佈局應該遵循一些基本的指引確保它們在大屏幕上是可用的和有效率的。遵循這些技巧去創建最優化的TV橫屏佈局。

- 創建橫屏佈局。TV屏幕總是顯示在橫屏模式。
- 把導航控件放置在屏幕的左邊或者右邊，並且保持內容在垂直區間。

- 創建分離的UI，用[Fragment](#)，並且用框架如[GridView](#)代替[ListView](#)獲得屏幕水平方向更好的使用。
- 用框架如[RelativeLayout](#)或者[LinearLayout](#)來排列視圖。基於對齊方式，縱橫比，和電視屏幕的像素密度，這個方法允許系統調整視圖大小的位置。
- 在佈局控件之間添加足夠的邊際，以避免成為一個雜亂的UI。

## Overscan

由於TV標準的演進，TV的佈局有一個獨特的需求是總是希望給觀眾顯示全屏圖像。因為這個原因，TV設備可能剪掉應用佈局的外邊緣去確保整個顯示器被填滿。這種行為通常簡稱為overscan。

避免屏幕元素由於overscan被剪掉，可以在佈局所有的邊緣增加總共10%的邊際。這換算為在[activity](#)的基礎佈局上左右邊緣留48dp的邊際和在上下留27dp的邊際。接下來的佈局例子展示瞭如何在TV應用根佈局上設置這些邊際。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/base_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_marginTop="27dp"
    android:layout_marginLeft="48dp"
    android:layout_marginRight="48dp"
    android:layout_marginBottom="27dp" >
</LinearLayout>
```

**Caution**：如果你正在使用v17 leanback類，不要在佈局中留overscan邊際，諸如[BrowseFragment](#)或者相關控件，因為那些佈局已經包含了overscan安全邊際。

## 創建方便使用的文本和控件

在TV應用佈局中的文本和控件應該在一定距離外是容易查看和導航的。接下來的技巧是確保你的用戶界面元素在一定距離外更容易查看。

- 分解文本為小塊，用戶可以快速瀏覽。
- 在暗背景下用亮色文字。這種風格在TV中更容易閱讀。
- 避免輕字體或者字體既窄且有非常寬闊的筆觸效果。用簡單的sans-serif字體並且去掉鋸齒效果以增加可讀性。
- 用安卓標準的字體大小。

```
<TextView
    android:id="@+id/atext"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:singleLine="true"
    android:textAppearance="?android:attr/textAppearanceMedium"/>
```

- 確保所有的控件是足夠大，使人們站在屏幕3米外（更大的屏幕這個距離會更大）可以看清楚。做這個最好的方式是用佈局相對大小而不是絕對大小，並且用密度無關像素（dip）單位代替像素單位。例如，設置控件的寬度，用 `wrap_content` 代替特定像素值，並且設置控件的邊際，用dip代替px值。更多關於密度無關像素和創建大尺寸屏幕的佈局，查看[Support Multiple Screens](#)。

## 管理TV佈局資源

通常的高清晰度TV分辨率是720p，1080i，1080p。假定你的TV佈局對象是一個 $1920 \times 1080$ 像素的屏幕，然後要允許安卓系統必要情況下縮減佈局元素到720p。通常，降低分辨率（刪除像素）不會降低佈局的外觀質量。但是增加分辨率會降低佈局顯示的質量，並且會對用戶體驗造成負面影響。

為了獲得最好的圖像縮放效果，儘可能提供9-patch圖片元素。如果在你的佈局中使用低質量或者小的圖片，它們將出現馬賽克，模糊或者顆粒，這不是一個好的用戶體驗。用高質量圖片代替它。

更多關於優化佈局和大屏幕的資源文件問題，參考[Designing for multiple screens](#)。

## 避免反模式佈局

有幾種創建佈局的方法你應該避免使用，因為它們不能在TV設備上很好的工作並且導致不好的用戶體驗。當開發TV佈局時，以下一些用戶界面是你應該明確不能使用的。

- 重用手機和平板佈局 - 不要重用沒有修改的手機或者平板應用的佈局。為其他安卓設備的佈局不適合TV設備，並且TV上應該簡化操作。
- ActionBar - 儘管這種用戶界面習慣是推薦使用在手機和平板上，但是他不適合TV界面。通常，狀態欄選項菜單（或者任何下拉菜單）堅決不要使用，因為用遙控器操作這樣的菜單是困難的。
- ViewPager - 在屏幕之間滑動能很好在手機或平板上工作，但是不要在TV上嘗試！更多信息關於設計適合TV的佈局，參考[TV Design](#)指導。

## 處理大圖片

TV設備，像任何其他安卓設備，內存有一定限制。如果你創建的應用中用了很高分辨率的圖片或者用了很多高分辨率圖片，它可能很快達到內存限制，並且導致內存溢出錯誤。避免這些類型的問題，遵循以下方法：

- 僅當圖片顯示在屏幕時才加載。例如，當在GridView或者Gallery中顯示多個圖片時，僅當getView()在視圖的Adapter中被調用時才加載圖片。
- 在Bitmap視圖中調用recycle()不再需要。
- 對存儲在內存中集合中的位圖對象使用弱引用。
- 如果你從網絡上獲取圖片，用AsyncTask去操作並且存儲它們在設備上以方便更快的存取。絕對不要在應用的主線程操作網絡傳輸。
- 當下載大圖片時，降低圖片到合適的尺寸，否則，下載圖片本身可能導致內存溢出問題。更多信息關於獲得最好的圖片操作性能，參考[Displaying Bitmaps Efficiently](#)。

## 提供有效的廣告

安卓TV的廣告必須總是全屏。廣告不可以出現在內容的旁邊或者覆蓋內容。用戶應當能用D-pad控制器關閉廣告。視頻廣告在開始時間後的30秒內應當能被關閉。

安卓TV不提供網頁瀏覽器。你的廣告不應該嘗試去啓動網頁瀏覽器或者重定向到Google Play Store。

Note：你能用WebView類登入服務器，如Google+和Facebook。

[下一節: 設計TV導航](#)

# 創建TV導航

編寫:awong1900 - 原文:<http://developer.android.com/training/tv/start/navigation.html>

TV設備為應用程序提供一組有限的導航控件。為你的TV應用創建有效的導航方案取決於理解這些有限的控件和用戶操作應用時的限制。因此當你為TV創建安卓應用時，額外注意當用戶用遙控器按鍵，而不是觸摸屏時，如何實際導航你的應用程序。

這節課解釋了創建有效的TV應用導航方案的最低要求和如何對應用程序使用這些要求。

## 使用D-pad導航

在TV設備上，用戶用遙控器設備的方向手柄（D-pad）或者方向鍵去控制控件。這類控制器限製為上下左右移動。為了創建最優化的TV應用，你必須提供一個用戶能快速學習如何使用有限控件導航的方案。

安卓framework自動地處理佈局元素之間的方嚮導航操作，因此你不需要在應用中做額外的事情。不管怎樣，你也應該用D-pad控制器實際測試去發現任何導航問題。接下來的指引是如何在TV設備上用D-pad測試應用的導航。

- 確保用戶能用D-pad控制器導航所有屏幕可見的控件。
- 對於滾動列表上的焦點，確保D-pad上下鍵能滾動列表，並且確定鍵能選擇列表中的項。檢查用戶可以選擇列表中的元素並且選中元素後仍可以滾動列表。
- 確定在控件之間切換是直接的和可預測的。

## 修改導航的方向

基於佈局元素中可選中的元素的相對位置，安卓framwork自動應用導航方向方案。你應該用D-pad控制器測試生成的導航方案。在測試後，如果你想用戶以一個特定的方式在佈局中移動，你可以在控件中設置明確的導航方向。

Note: 如果系統使用的默認順序不是很好，你應該僅用這些屬性去修改導航順序。

接下來的示例代碼展示如何為TextView佈局控件定義下一個控件焦點。

```
<TextView android:id="@+id/Category1"
          android:nextFocusDown="@+id/Category2"\>
```

接下來的列表展示了用戶接口控件所有可用的導航屬性。

屬性	功能
nextFocusDown	定義用戶按下導航時的焦點
nextFocusLeft	定義用戶按左導航時的焦點
nextFocusRight	定義用戶按右導航時的焦點
nextFocusUp	定義用戶按上導航時的焦點

去使用這些明確的導航屬性，設置另一個佈局控件的ID值（`android:id` 值）。你應該設置導航順序為一個循環，因此最後一個控件返回至第一個焦點。

# 提供清楚的焦點和選中狀態

在TV設備上的應用導航方案的成功是基於用戶如何容易的決定屏幕上的界面元素的焦點。如果你不提供清晰的焦點項顯示（和用戶能操作的選項），他們會很快泄氣並退出你的應用。同樣的原因，重要的是當你的應用打開或者任何空閒的時間，總是有焦點項可以立即操作。

你的應用佈局和實現應該用顏色，大小，動畫或者它們組在一起來幫助用戶容易地決定下一步操作。在應用中用一致的焦點顯示方案。

安卓提供[Drawable State List Resources](#)來實現高亮選中的焦點。接下來的示例代碼展示瞭如何為用戶導航到控件並選擇它時使用視覺化按鈕顯示：

```
<!-- res/drawable/button.xml -->
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

接下來的XML示例代碼對按鈕控件應用了上面的按鍵狀態列表drawable：

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

確保在可焦點的和可選中的控件中提供了充分的填充，以便它們的周圍高亮清晰可見。

更多建議關於TV應用中設計有效的選中和焦點，看[Patterns of TV](#)。

---

[下一節: 創建TV播放應用](#)

# 創建TV播放應用

---

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/tv/playback/index.html>

瀏覽和播放媒體文件往往是由一個電視應用程序提供的用戶體驗的一部分。從頭開始構建這樣的體驗，並同時確保它是快速，流暢，和有吸引力的是具有相當挑戰性的。您的應用程序提供訪問媒體類別無論大小，允許用戶快速瀏覽選項，並獲得他們想要的內容是很重要的。

Android框架通過[v17 leanback support library](#)為構建用戶界面提供接口。該庫提供類來創建用於瀏覽和播放多媒體的高效框架，為開發者減少代碼。該類可以進行擴展和定製，所以你可以為你的應用程序創建一個獨特的高效的類。

這節課將向您介紹如何用Leanback的支持庫構建用於瀏覽和播放電視媒體內容的電視應用程序。

## 主題

---

- [創建一個類別瀏覽器](#)

學習如何使用Leanback的支持庫，建立一個媒體類別的瀏覽界面。

- [提供一個卡片View](#)

學習使用Leanback的支持庫，建立一個卡片視圖的內容項目。

- [創建詳細信息View](#)

學習使用Leanback的支持庫，建立一個詳細內容展示頁。

- [顯示正在播放卡片](#)

學習如何使用MediaSession在主屏幕上顯示正在播放。

# 創建目錄瀏覽器

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/tv/playback/browse.html>

在TV上運行的 多媒體應用得允許用戶瀏覽,選擇和播放它所提供的內容。目錄瀏覽器的用戶體驗要簡單和直觀,以及賞心悅目,引人入勝。

這節課討論如何使用的[V17 Leanback](#)庫提供的類來實現用戶界面,用於從您的應用程序的媒體目錄瀏覽音樂或視頻。

## 創建一個目錄佈局

leanback 類庫中的[BrowseFragment](#)允許您用最少的代碼創建一個用於按行瀏覽的主佈局,下面的例子將演示如何創建包含[BrowseFragment](#)的佈局

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

    <fragment
        android:name="android.support.v17.leanback.app.BrowseFragment"
        android:id="@+id/browse_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />
</LinearLayout>
```

為了使 [activity](#) 工作,需要在佈局中取回[BrowseFragment](#)的元素。使用這個類中的方法設置顯示參數,如圖標,標題,以及該類別是否可用。下面的代碼簡單的演示了怎樣設置[BrowseFragment](#)佈局參數:

```
public class BrowseMediaActivity extends Activity {

    public static final String TAG = "BrowseActivity";
    protected BrowseFragment mBrowseFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.browse_fragment);

        final FragmentManager fragmentManager = getFragmentManager();
        mBrowseFragment = (BrowseFragment) fragmentManager.findFragmentById(
            R.id.browse_fragment);

        // Set display parameters for the BrowseFragment
        mBrowseFragment.setHeadersState(BrowseFragment.HEADERS_ENABLED);
        mBrowseFragment.setTitle(getString(R.string.app_name));
        mBrowseFragment.setBadgeDrawable(getResources().getDrawable(
            R.drawable.ic_launcher));
        mBrowseFragment.setBrowseParams(params);

    }
}
```

## 顯示媒體列表

[BrowseFragment](#)允許您定義和使用 adapter 和presenter 定義顯示可瀏覽媒體內容類別和媒體項目。Adapters 允許你連接本地或網絡數據資源。Presenters操控的媒體項目的數據，並提供佈局信息在屏幕上顯示的項目。

下面的示例代碼演示了一個為顯示字符串數據的Presenters的實現

```
public class StringPresenter extends Presenter {  
    private static final String TAG = "StringPresenter";  
  
    public ViewHolder onCreateViewHolder(ViewGroup parent) {  
        TextView textView = new TextView(parent.getContext());  
        textView.setFocusable(true);  
        textView.setFocusableInTouchMode(true);  
        textView.setBackground(  
            parent.getContext().getResources().getDrawable(R.drawable.text_bg));  
        return new ViewHolder(textView);  
    }  
  
    public void onBindViewHolder(ViewHolder viewHolder, Object item) {  
        ((TextView) viewHolder.view).setText(item.toString());  
    }  
  
    public void onUnbindViewHolder(ViewHolder viewHolder) {  
        // no op  
    }  
}
```

當你已經為你的媒體項目構建了一個Presenter類，你可以為[BrowseFragment](#)建立並添加一個適配器並在屏幕上顯示這些媒體項目。下面的示例代碼演示瞭如何用StringPresenter類構造一個類別和項目適配器：

```
private ArrayAdapter mRowsAdapter;  
private static final int NUM_ROWS = 4;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    ...  
  
    buildRowsAdapter();  
}  
  
private void buildRowsAdapter() {  
    mRowsAdapter = new ArrayAdapter(new ListRowPresenter());  
  
    for (int i = 0; i < NUM_ROWS; ++i) {  
        ArrayAdapter listRowAdapter = new ArrayAdapter(  
            new StringPresenter());  
        listRowAdapter.add("Media Item 1");  
        listRowAdapter.add("Media Item 2");  
        listRowAdapter.add("Media Item 3");  
        HeaderItem header = new HeaderItem(i, "Category " + i, null);  
        mRowsAdapter.add(new ListRow(header, listRowAdapter));  
    }  
  
    mBrowseFragment.setAdapter(mRowsAdapter);  
}
```

這個例子顯示了靜態實現適配器。典型的媒體瀏覽器使用網絡數據庫或網絡服務。使用從網絡取回的數據做的媒體瀏覽器，參看例子[Android TV](#)

## 更新背景

為了給媒體瀏覽應用增加視覺趣味，你可以在用戶瀏覽的內容時更新背景圖片。這種技術可以讓你的應用程序的互動感

倍增。

Leanback庫提供了[BackgroundManager](#)類為你的TV應用的[activity](#)更換背景。下面的例子演示瞭如何創建一個簡單的方法更換背景：

```
protected void updateBackground(Drawable drawable) {  
    BackgroundManager.getInstance(this).setDrawable(drawable);  
}
```

許多現有的媒體瀏覽應用在用戶瀏覽媒體列表自動更新的背景。為了做到這一點，你可以設置一個選擇監聽器，根據用戶的當前選擇自動更新背景。下面的例子演示瞭如何建立一個[OnItemViewSelectedListener](#)監聽選擇事件並更新背景：

```
protected void clearBackground() {  
    BackgroundManager.getInstance(this).setDrawable(mDefaultBackground);  
}  
  
protected OnItemViewSelectedListener getDefaultItemViewSelectedListener() {  
    return new OnItemViewSelectedListener() {  
        @Override  
        public void onItemSelected(Object item, Row row) {  
            if (item instanceof Movie ) {  
                URI uri = ((Movie)item).getBackdropURI();  
                updateBackground(uri);  
            } else {  
                clearBackground();  
            }  
        }  
    };  
}
```

注意：以上的示例是為了簡單。當你在自己的應用程序創建這個功能，你應該考慮運行在一個單獨的線程在後臺更新操作獲得更好的性能。此外，如果你正計劃在用戶觸發項目滾動時更新背景，考慮增加一個時延，直到用戶停止操作時再更新背景圖像。這樣可以避免過多的背景圖片的更新。

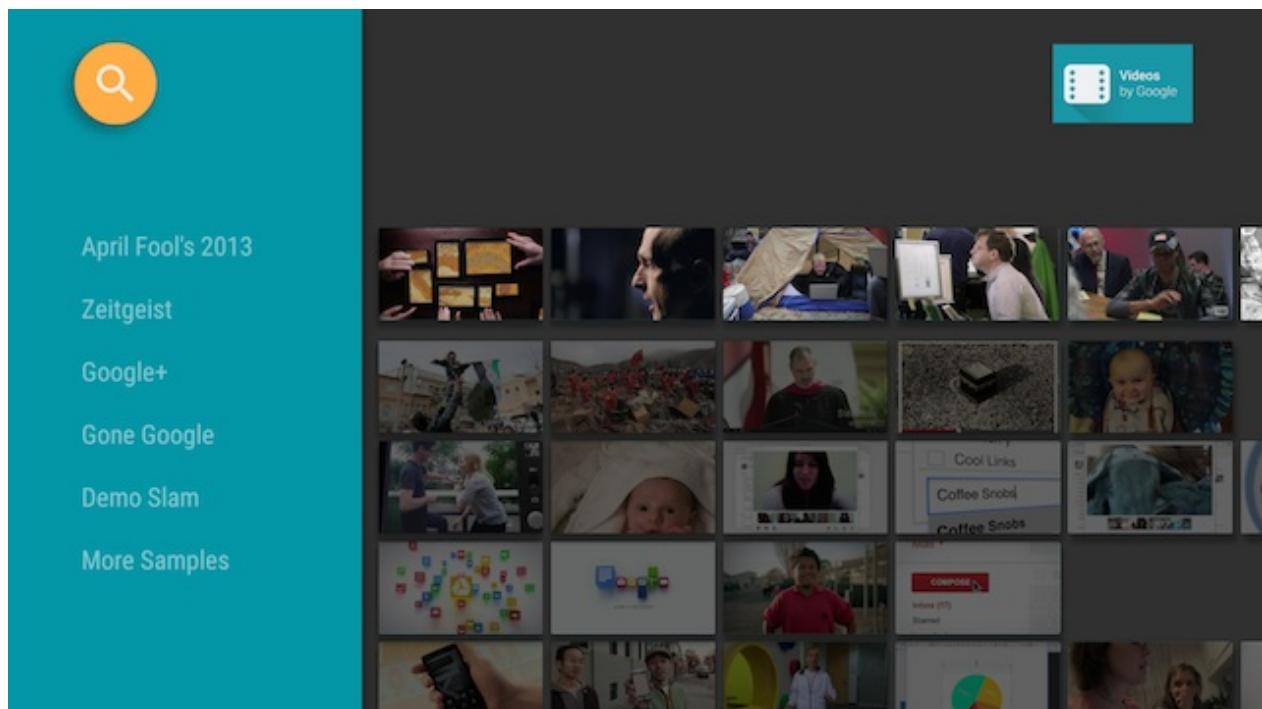
# 提供一個Card視圖

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/tv/playback/card.html>

在前面的課程中，我們創建一個目錄瀏覽器，實現了瀏覽 fragment，顯示了媒體項目的列表。在本課程中，我們將創建該卡視圖的媒體項目，並在瀏覽fragment中呈現出來。

[BaseCardView](#)類以及子類顯示與媒體項目相關聯的元數據。在本節課程中使用的[ImageCardView](#)類顯示隨着媒體項目的標題內容的圖像。

這節課介紹了GitHub上 [Android Leanback sample app](#)的示例應用程序代碼。使用該示例代碼，開始你自己的應用程序。



## 創建一個卡片呈現者

[Presenter](#)生成視圖並把類和它們綁定起來。在你的瀏覽 fragment 中將內容呈現給用戶,你為內容卡片創建[Presenter](#)並把它傳給適配器然後將內容呈現在屏幕上。在下面的代碼中,CardPresenter在 [LoaderManager](#)的[onLoadFinished](#))方法中被創建。

```
@Override  
public void onLoadFinished(Loader<HashMap<String, List<Movie>>> arg0,  
                            HashMap<String, List<Movie>> data) {  
  
    mRowsAdapter = new ArrayObjectAdapter(new ListRowPresenter());  
    CardPresenter cardPresenter = new CardPresenter();  
  
    int i = 0;  
  
    for (Map.Entry<String, List<Movie>> entry : data.entrySet()) {  
        ArrayObjectAdapter listRowAdapter = new ArrayObjectAdapter(cardPresenter);  
        List<Movie> list = entry.getValue();  
  
        for (int j = 0; j < list.size(); j++) {
```

```

        listRowAdapter.add(list.get(j));
    }
    HeaderItem header = new HeaderItem(i, entry.getKey(), null);
    i++;
    mRowsAdapter.add(new ListRow(header, listRowAdapter));
}

HeaderItem gridHeader = new HeaderItem(i, getString(R.string.more_samples),
    null);

GridItemPresenter gridPresenter = new GridItemPresenter();
ArrayObjectAdapter gridRowAdapter = new ArrayObjectAdapter(gridPresenter);
gridRowAdapter.add(getString(R.string.grid_view));
gridRowAdapter.add(getString(R.string.error_fragment));
gridRowAdapter.add(getString(R.string.personal_settings));
mRowsAdapter.add(new ListRow(gridHeader, gridRowAdapter));

setAdapter(mRowsAdapter);

updateRecommendations();
}

```

## 創建一個卡片視圖

在這步中,你將用view holder創建一個卡片presenter來為卡片視圖呈現媒體項目。注意,每個presenter只能創建一個view類別。如果你有兩個不同的卡片視圖,你就得創建兩個不同的presenter

在presenter實現`onCreateViewHolder`)時創建一個可以呈現內容項目的view holder。

```

@Override
public class CardPresenter extends Presenter {

    private Context mContext;
    private static int CARD_WIDTH = 313;
    private static int CARD_HEIGHT = 176;
    private Drawable mDefaultCardImage;

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent) {
        mContext = parent.getContext();
        mDefaultCardImage = mContext.getResources().getDrawable(R.drawable.movie);
        ...
    }
}

```

在`onCreateViewHolder`)方法中,創建呈現內容的卡片視圖。下面的例子用的是`ImageCardView`

當卡片被選中時,默認的行為是放大展開。如果你想創建不同顏色的卡片可以向下面這樣調用`setSelected`)方法中實現。

```

...
ImageCardView cardView = new ImageCardView(mContext) {
    @Override
    public void setSelected(boolean selected) {
        int selected_background = mContext.getResources().getColor(R.color.detail_background);
        int default_background = mContext.getResources().getColor(R.color.default_background);
        int color = selected ? selected_background : default_background;
        findViewById(R.id.info_field).setBackgroundColor(color);
        super.setSelected(selected);
    }
};
...

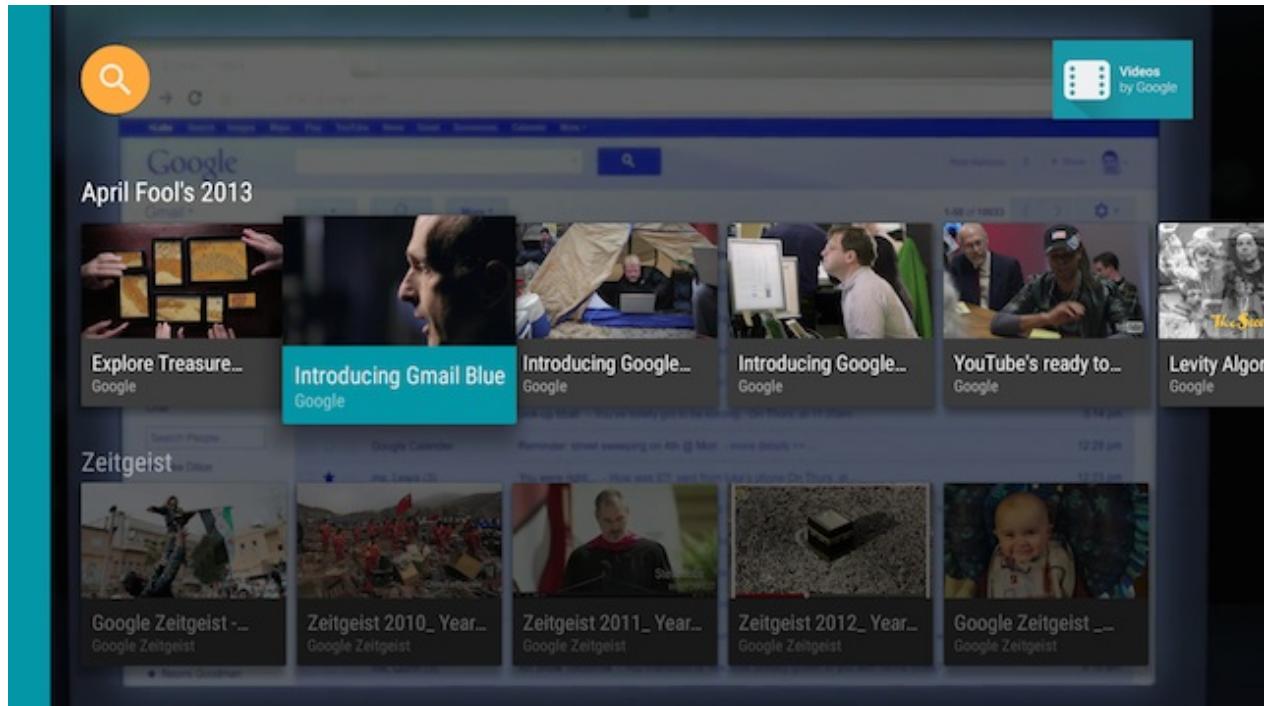
```

當用戶打開你的應用時,`Presenter.ViewHolder` 為內容項目顯示了卡片視圖。你需要調用`setFocusable(true)`

和`setFocusableInTouchMode(true)`方法設置接收來自D-pad的焦點控制。

```
...  
    cardView.setFocusable(true);  
    cardView.setFocusableInTouchMode(true);  
    return new ViewHolder(cardView);  
}
```

當用戶選中`ImageCardView`時，它用你制定的顏色背景展開文字內容，就像下面這樣。



# 創建詳情頁

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/tv/playback/details.html>

待認領進行編寫，有意向的小夥伴，可以直接修改對應的markdown文件，進行提交！

[v17 leanback support library](#) 庫提供的媒體瀏覽接口包含顯示附加媒體信息的類,比如描述和預覽,以及對項目的操作,比如購買或播放。

這節課討論如何為媒體項目的詳細信息創建 presenter 類，以及用戶選擇一個媒體項目時如何擴展 [DetailsFragment](#) 類來實現顯示媒體詳細信息視圖。

小貼士: 這裏的實現例子用的是包含 [DetailsFragment](#) 的附加 [activity](#)。但也可以在同一個 [activity](#) 中用 fragment 轉換將 [BrowseFragment](#)替換為 [DetailsFragment](#).更多關於fragment的信息請參考[Building a Dynamic UI with Fragments](#)

## 創建詳細Presenter

在leanback庫提供的媒體瀏覽框架中,可以用presenter對象控制屏幕顯示數據,包括媒體詳細信息。[AbstractDetailsDescriptionPresenter](#) 類提供的框架幾乎是媒體項目詳細信息的完全繼承。你只需要實現[onBindDescription\(\)](#)方法,像下面這樣把數據信息和視圖綁定起來。

```
public class DetailsDescriptionPresenter  
    extends AbstractDetailsDescriptionPresenter {  
  
    @Override  
    protected void onBindDescription(ViewHolder viewHolder, Object itemData) {  
        MyMediaItemDetails details = (MyMediaItemDetails) itemData;  
        // In a production app, the itemData object contains the information  
        // needed to display details for the media item:  
        // viewHolder.getTitle().setText(details.getShortTitle());  
  
        // Here we provide static data for testing purposes:  
        viewHolder.getTitle().setText(itemData.toString());  
        viewHolder.getSubtitle().setText("2014 Drama TV-14");  
        viewHolder.getBody().setText("Lorem ipsum dolor sit amet, consectetur "  
            + "adipiscing elit, sed do eiusmod tempor incididunt ut labore "  
            + "et dolore magna aliqua. Ut enim ad minim veniam, quis "  
            + "nostrud exercitation ullamco laboris nisi ut aliquip ex ea "  
            + "commodo consequat.");  
    }  
}
```

## 擴展詳細fragment

當使用 [DetailsFragment](#) 類顯示你的媒體項目詳細信息時,擴展該類並提供像預覽圖片,操作等附加內容。你也可以提供一系列的相關媒體信息。

下面的例子演示了怎樣用presenter類為媒體項目添加預覽圖片和操作。這個例子也演示了添加相關媒體行。

```
public class MediaItemDetailsFragment extends DetailsFragment {  
    private static final String TAG = "MediaItemDetailsFragment";  
    private ArrayAdapter mRowsAdapter;
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    Log.i(TAG, "onCreate");
    super.onCreate(savedInstanceState);

    buildDetails();
}

private void buildDetails() {
    ClassPresenterSelector selector = new ClassPresenterSelector();
    // Attach your media item details presenter to the row presenter:
    DetailsOverviewRowPresenter rowPresenter =
        new DetailsOverviewRowPresenter(new DetailsDescriptionPresenter());

    selector.addClassPresenter(DetailsOverviewRow.class, rowPresenter);
    selector.addClassPresenter(ListRow.class,
        new ListRowPresenter());
    mRowsAdapter = new ArrayObjectAdapter(selector);

    Resources res = getActivity().getResources();
    DetailsOverviewRow detailsOverview = new DetailsOverviewRow(
        "Media Item Details");

    // Add images and action buttons to the details view
    detailsOverview.setImageDrawable(res.getDrawable(R.drawable.jelly_beans));
    detailsOverview.addAction(new Action(1, "Buy $9.99"));
    detailsOverview.addAction(new Action(2, "Rent $2.99"));
    mRowsAdapter.add(detailsOverview);

    // Add a Related items row
    ArrayObjectAdapter listRowAdapter = new ArrayObjectAdapter(
        new StringPresenter());
    listRowAdapter.add("Media Item 1");
    listRowAdapter.add("Media Item 2");
    listRowAdapter.add("Media Item 3");
    HeaderItem header = new HeaderItem(0, "Related Items", null);
    mRowsAdapter.add(new ListRow(header, listRowAdapter));

    setAdapter(mRowsAdapter);
}
}

```

## 創建詳細信息activity

像 [DetailsFragment](#)這樣的 fragment 為了使用或顯示必須包含[activity](#)。為你的詳細信息與瀏覽分開創建[activity](#)並通過傳遞Intent打開。這節演示瞭如何創建一個包含媒體詳細信息的[activity](#)。

創建詳細信息前先為 [DetailsFragment](#)創建一個佈局文件:

```

<!-- file: res/layout/details.xml -->

<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.example.android.mediarow.MediaItemDetailsFragment"
    android:id="@+id/details_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>

```

接下來用上面的佈局文件創建一個[activity](#):

```

public class DetailsActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState) {

```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
    }
}
```

最後在manifest文件中申明activity。記得添加Leanback主題以確保用戶界面中有媒體瀏覽activity。

```
<application>
    ...
<activity android:name=".DetailsActivity"
    android:exported="true"
    android:theme="@style/Theme_Leanback"/>
</application>
```

## 為點擊項目添加Listener

實現 DetailsFragment後，在用戶點擊媒體條目時將你的媒體瀏覽view切換詳細信息view。為了確保動作的實現，在BrowserFragment中添加[OnItemClickListener]通過Intent開啓詳細信息activity。

下面的例子演示了實現怎樣在媒體瀏覽view中實現一個 listener開啓詳細信息view。

```
public class BrowseMediaActivity extends Activity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // create the media item rows
        buildRowsAdapter();

        // add a listener for selected items
        mBrowseFragment.OnItemViewClickedListener(
            new OnItemViewClickedListener() {
                @Override
                public void onItemClicked(Object item, Row row) {
                    System.out.println("Media Item clicked: " + item.toString());
                    Intent intent = new Intent(BrowseMediaActivity.this,
                        DetailsActivity.class);
                    // pass the item information
                    intent.getExtras().putLong("id", item.getId());
                    startActivity(intent);
                }
            });
    }
}
```

# 顯示正在播放卡片

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/tv/playback/now-playing.html>

TV應用允許用戶在使用其他應用時後臺播放音樂或其他媒體。如果你的應用程序允許後臺，它必須要為用戶提供返回該應用暫停音樂或切換到一個新的歌曲的方法。Android框架允許電視應用通過在主屏幕上顯示正在播放卡做到這一點。

正在播放卡片是系統的組建，它可以在推薦的行上顯示正在播放的媒體會話。它包括了媒體元數據，如專輯封面，標題和應用程序圖標。當用戶選擇它，系統將打開擁有該會話的應用程序。

這節課將演示如何使用 `MediaSession` 類實現正在播放卡片。

## 開啟媒體會話

一個播放應用可以作為 `activity` 或者 `service` 運行。`service` 是當 `activity` 結束時依然可以後臺播放的。在這節討論中，媒體播放應用是假設在 `MediaBrowserService` 下運行的。

在 `service` 的 `onCreate()` 方法中創建一個新的 `MediaSession`，設置適當的回調函數和標誌，並設置 `MediaBrowserService` 令牌。

```
mSession = new MediaSession(this, "MusicService");
mSession.setCallback(new MediaSessionCallback());
mSession.setFlags(MediaSession.FLAG_HANDLES_MEDIA_BUTTONS |
    MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);

// for the MediaBrowserService
setSessionToken(mSession.getSessionToken());
```

注意：正在播放卡片只有在媒體會話設置了 `FLAG_HANDLES_TRANSPORT_CONTROLS` 標誌時才可以顯示。

## 顯示正在播放卡片

如果會話是系統最高優先級的會話那麼正在播放卡片將在 `setActivity(true)` 調用後顯示。同時你的應用必須像在 [Managing Audio Focus](#) 一節中那樣請求音頻焦點。

```
private void handlePlayRequest() {
    tryToGetAudioFocus();
    if (!mSession.isActive()) {
        mSession.setActive(true);
    }
    ...
}
```

如果另一個應用發起媒體播放請求並調用 `setActivity(false)` 後這個卡片將從主屏上移除。

## 更新播放狀態

正如任何媒體的應用程序，在 `MediaSession` 中更新播放狀態，使卡片可以顯示當前的元數據，如在下面的例子：

```
private void updatePlaybackState() {
    long position = PlaybackState.PLAYBACK_POSITION_UNKNOWN;
    if (mMediaPlayer != null && mMediaPlayer.isPlaying()) {
        position = mMediaPlayer.getCurrentPosition();
    }
    PlaybackState.Builder stateBuilder = new PlaybackState.Builder()
        .setActions(getAvailableActions());
    stateBuilder.setState(mState, position, 1.0f);
    mSession.setPlaybackState(stateBuilder.build());
}

private long getAvailableActions() {
    long actions = PlaybackState.ACTION_PLAY |
        PlaybackState.ACTION_PLAY_FROM_MEDIA_ID |
        PlaybackState.ACTION_PLAY_FROM_SEARCH;
    if (mPlayingQueue == null || mPlayingQueue.isEmpty()) {
        return actions;
    }
    if (mState == PlaybackState.STATE_PLAYING) {
        actions |= PlaybackState.ACTION_PAUSE;
    }
    if (mcurrentIndexOnQueue > 0) {
        actions |= PlaybackState.ACTION_SKIP_TO_PREVIOUS;
    }
    if (mcurrentIndexOnQueue < mPlayingQueue.size() - 1) {
        actions |= PlaybackState.ACTION_SKIP_TO_NEXT;
    }
    return actions;
}
```

## 顯示媒體元數據

為當前正在播放通過[setMetadata\(\)](#)方法設置 [MediaMetadata](#)。這個方法可以讓你為正在播放卡提供有關軌道，如標題，副標題，和各種圖標等信息。下面的例子假設你的播放數據存儲在自定義的[MediaData](#)類中。

```
private void updateMetadata(MediaData myData) {
    MediaMetadata.Builder metadataBuilder = new MediaMetadata.Builder();
    // To provide most control over how an item is displayed set the
    // display fields in the metadata
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_DISPLAY_TITLE,
        myData.displayTitle);
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_DISPLAY_SUBTITLE,
        myData.displaySubtitle);
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_DISPLAY_ICON_URI,
        myData.artUri);
    // And at minimum the title and artist for legacy support
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_TITLE,
        myData.title);
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_ARTIST,
        myData.artist);
    // A small bitmap for the artwork is also recommended
    metadataBuilder.putString(MediaMetadata.METADATA_KEY_ART,
        myData.artBitmap);
    // Add any other fields you have for your data as well
    mSession.setMetadata(metadataBuilder.build());
}
```

## 響應用戶的動作

當用戶選擇正在播放卡片時，系統打開應用並擁有會話。如果你的應用在[setSessionActivity\(\)](#)有[PendingIntent](#)要傳遞，系統將會像下面演示的那樣開啟[activity](#)。如果不是，則系統默認的Intent打開。您指定的活動必須提供播放控制，允許用戶暫停或停止播放。

```
Intent intent = new Intent(mContext, MyActivity.class);
PendingIntent pi = PendingIntent.getActivity(context, 99 /*request code*/,
    intent, PendingIntent.FLAG_UPDATE_CURRENT);
mSession.setSessionActivity(pi);
```

# 幫助用戶在TV上找到內容

---

編寫:[awong1900](#) - 原文:<http://developer.android.com/training/tv/discovery/index.html>

TV設備為用戶提供了許多的休閒娛樂選擇。它們提供上千個應用和相關的內容服務。同時，大部分用戶操作TV時，喜歡比較少的輸入操作。面對用戶可能的選擇，重要的一點是應用開發者為用戶提供快速容易的路徑，發現和享受你的內容。

安卓framework層幫助你為用戶提供若幹路徑，去找到內容，包括主屏幕的推薦和應用的內容目錄的搜索。

這節課展示如何幫助用戶找到應用內容，通過推薦和應用內搜索。

## 主題

---

- [推薦TV內容](#) 學習如何推薦內容給用戶，使它出現在TV設備的主屏幕推薦欄。
  - [使TV應用是可被搜索的](#) 學習如何使內容在安卓TV主屏幕中被搜索到。
  - [TV應用內搜索](#) 學習如何在應用內使用內置的TV搜索界面。
- 

[開始](#)

# 推薦TV內容

編寫:[awong1900](#) - 原文:<http://developer.android.com/training/tv/discovery/recommendations.html>

當操作TV時，用戶通常喜歡使用最少的輸入操作來找內容。許多用戶的理想場景是，坐下，打開TV然後觀看。用最少的步驟讓用戶觀看他們的喜歡的內容是最好的方式。

安卓framework為了實現少交互而提供了主屏幕推薦欄。在設備第一次使用時候，內容推薦出現在TV主屏幕的第一欄。從你的應用程序的內容目錄提供建議可以幫助把用戶帶回到你的應用程序。

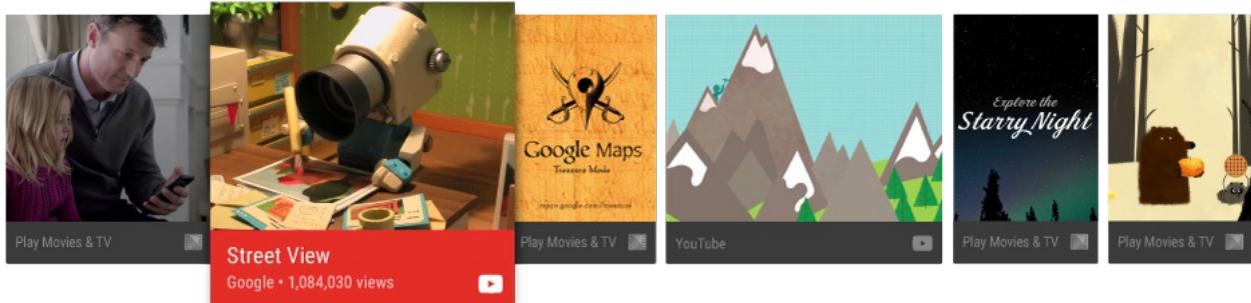


圖1. 一個推薦欄的例子

這節課教你如何創建推薦和提供他們到安卓framework，這樣用戶能容易的發現和使用你的應用內容。這個討論描述了一些代碼，在[安卓Leanback示例代碼](#)。

## 創建推薦服務

內容推薦是被後臺處理創建。為了把你的應用去提供到推薦，創建一個週期性添加列表服務，從應用目錄到系統推薦列表。

接下來的代碼描繪瞭如何擴展[IntentService](#)為你的應用創建推薦服務：

```
public class UpdateRecommendationsService extends IntentService {
    private static final String TAG = "UpdateRecommendationsService";
    private static final int MAX_RECOMMENDATIONS = 3;

    public UpdateRecommendationsService() {
        super("RecommendationService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.d(TAG, "Updating recommendation cards");
        HashMap<String, List<Movie>> recommendations = VideoProvider.getMovieList();
        if (recommendations == null) return;

        int count = 0;

        try {
            RecommendationBuilder builder = new RecommendationBuilder()
                .setContext(getApplicationContext())
                .setSmallIcon(R.drawable.videos_by_google_icon);

            for (Map.Entry<String, List<Movie>> entry : recommendations.entrySet()) {
                for (Movie movie : entry.getValue()) {

```

```

        Log.d(TAG, "Recommendation - " + movie.getTitle());

        builder.setBackground(movie.getCardImageUrl())
            .setId(count + 1)
            .setPriority(MAX_RECOMMENDATIONS - count)
            .setTitle(movie.getTitle())
            .setDescription(getString(R.string.popular_header))
            .setImage(movie.getCardImageUrl())
            .setIntent(buildPendingIntent(movie))
            .build();

        if (++count >= MAX_RECOMMENDATIONS) {
            break;
        }
    }
    if (++count >= MAX_RECOMMENDATIONS) {
        break;
    }
}
} catch (IOException e) {
    Log.e(TAG, "Unable to update recommendation", e);
}
}

private PendingIntent buildPendingIntent(Movie movie) {
    Intent detailsIntent = new Intent(this, DetailsActivity.class);
    detailsIntent.putExtra("Movie", movie);

    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(DetailsActivity.class);
    stackBuilder.addNextIntent(detailsIntent);
    // Ensure a unique PendingIntent, otherwise all recommendations end up with the same
    // PendingIntent
    detailsIntent.setAction(Long.toString(movie.getId()));

    PendingIntent intent = stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
    return intent;
}
}

```

為了服務被系統意識到和運行，在應用manifest中註冊它，接下來的代碼片段展示瞭如何定義這個類的服務：

```

<manifest ... >
<application ... >
    ...
    <service
        android:name="com.example.android.tvleanback.UpdateRecommendationsService"
        android:enabled="true" />
</application>
</manifest>

```

## 刷新推薦

基於用戶的行爲和數據來推薦，例如播放列表，喜愛列表和相關內容。當刷新推薦時，不僅僅是刪除和重新加載他們，因為這樣導致推薦出現在推薦欄的結尾。一旦一個內容項被播放，如一個影片，從推薦中[刪除它](#)。

應用的推薦被保存依據哪個應用提供他們。framework `interleave`應用推薦基於推薦質量，用戶習慣的收集。最好的推薦使應用推薦更幸運的出現在列表前面。

## 創建推薦

一旦你的推薦服務開始運行，它必須創建推薦和送他們到安卓framework。Framework收到推薦作爲[通知](#)對象。它用特

定的模板並且標記為特定的目錄。

## 設置值

去設置推薦卡片的UI元素，創建一個builder類用接下來的builder樣式描述。首先，設置推薦卡片元素的值。

```
public class RecommendationBuilder {  
    ...  
  
    public RecommendationBuilder setTitle(String title) {  
        mTitle = title;  
        return this;  
    }  
  
    public RecommendationBuilder setDescription(String description) {  
        mDescription = description;  
        return this;  
    }  
  
    public RecommendationBuilder setImage(String uri) {  
        mImageUri = uri;  
        return this;  
    }  
  
    public RecommendationBuilder setBackground(String uri) {  
        mBackgroundUri = uri;  
        return this;  
    }  
    ...  
}
```

## 創建通知

一旦你設置了值，然後去創建通知，從builder類分配值到通知，並且調用[NotificationCompat.Builder.build\(\)](#)。

並且，確信調用[setLocalOnly\(\)](#)，這樣[NotificationCompat.BigPictureStyle](#)通知不將顯示在另一個設備。

接下來的代碼示例展示瞭如何創建推薦。

```
public class RecommendationBuilder {  
    ...  
  
    public Notification build() throws IOException {  
        ...  
  
        Notification notification = new NotificationCompat.BigPictureStyle(  
            new NotificationCompat.Builder(mContext)  
                .setContentTitle(mTitle)  
                .setContentText(mDescription)  
                .setPriority(mPriority)  
                .setLocalOnly(true)  
                .setOngoing(true)  
                .setColor(mContext.getResources().getColor(R.color.fastlane_background))  
                .setCategory(Notification.CATEGORY_RECOMMENDATION)  
                .setLargeIcon(image)  
                .setSmallIcon(mSmallIcon)  
                .setContentIntent(mIntent)  
                .setExtras(extras))  
            .build();  
  
        return notification;  
    }  
}
```

# 運行推薦服務

你的應用推薦服務必須週期性運行，原因是創建當前的推薦。去運行你的服務，創建一個類運行計時器和在週期間隔關聯它。接下來的代碼例子擴展了BroadcastReceiver類去開始每半小時的推薦服務的週期性執行：

```
public class BootupActivity extends BroadcastReceiver {
    private static final String TAG = "BootupActivity";
    private static final long INITIAL_DELAY = 5000;

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "BootupActivity initiated");
        if (intent.getAction().endsWith(Intent.ACTION_BOOT_COMPLETED)) {
            scheduleRecommendationUpdate(context);
        }
    }

    private void scheduleRecommendationUpdate(Context context) {
        Log.d(TAG, "Scheduling recommendations update");

        AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
        Intent recommendationIntent = new Intent(context, UpdateRecommendationsService.class);
        PendingIntent alarmIntent = PendingIntent.getService(context, 0, recommendationIntent, 0);

        alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
            INITIAL_DELAY,
            AlarmManager.INTERVAL_HALF_HOUR,
            alarmIntent);
    }
}
```

這個BroadcastReceiver類的實現必須運行在TV設備啓動後。去完成這個，註冊這個類在應用manifest的intet filter中，它監聽設備啓動完成。接下來的代碼展示瞭如何添加這個配置到manifest。

```
<manifest ... >
<application ... >
    <receiver android:name="com.example.android.tvleanback.BootupActivity"
              android:enabled="true"
              android:exported="false">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
        </intent-filter>
    </receiver>
</application>
</manifest>
```

Important：接收一個啓動完成通知需要你的應用有RECEIVE\_BOOT\_COMPLETED權限。更多信息，查看ACTION\_BOOT\_COMPLETED。

在推薦服務類的onHandleIntent())方法，提交推薦到管理器，如下：

```
Notification notification = notificationBuilder.build();
mNotificationManager.notify(id, notification);
```

下一節：使TV應用可搜索

# 使TV應用是可被搜索的

編寫:awong1900 - 原文:<http://developer.android.com/training/tv/discovery/searchable.html>

安卓TV使用安卓[搜索接口](#)從安裝的應用中檢索內容數據並且釋放搜索結果給用戶。你的應用內容數據能被包含在這些結果中，去給用戶即時訪問應用程序中的內容。

你的應用必須提供安卓TV數據字段，它是用戶在搜索框中輸入字符生成的建議搜索結果。去做這個，你的應用必須實現[Content Provider](#)，在[searchable.xml](#)配置文件描述content provider和其他必要的安卓TV信息。你也需要一個[activity](#)在用戶選擇一個建議的搜索結果時處理intent的觸發。所有的這些被描述在[Adding Custom Suggestions](#)。本文描述安卓TV應用搜索的關鍵點。

這節課展示安卓中搜索的知識，展示如何使你的應用在安卓TV裏是可被搜索的。確信你熟悉[Search API guide](#)的解釋。在下面的這節課程之前，查看[Adding Search Functionality](#)訓練。

這個討論描述了一些代碼，從[安卓Leanback示例代碼](#)摘出。代碼可以在Github上找到。

## 識別列

[SearchManager](#)描述了數據字段，它被代表為SOLite數據庫的列。不管你的數據格式，你必須把你的數據字段填到那些列，通常用存取你的內容數據的類。更多信息，查看[Building a suggestion table\(\)](#)。

SearchManager類為安卓TV包含了幾個列。下面是重要的一些列：

值	描述	
SUGGEST_COLUMN_TEXT_1	內容名字 (required)	
SUGGEST_COLUMN_TEXT_2	內容的文本描述	
SUGGEST_COLUMN_RESULT_CARD_IMAGE	圖片/封面	
SUGGEST_COLUMN_CONTENT_TYPE	媒體的MIME類型 (required)	
SUGGEST_COLUMN_VIDEO_WIDTH	媒體的分辨率寬度 SUGGEST_COLUMN_VIDEO_HEIGHT	媒體的分辨率高度
SUGGEST_COLUMN_PRODUCTION_YEAR	內容的產品年份 (required)	
SUGGEST_COLUMN_DURATION	媒體的時間長度	

搜索framework需要以下的列：

- [SUGGEST\\_COLUMN\\_TEXT\\_1](#)
- [SUGGEST\\_COLUMN\\_CONTENT\\_TYPE](#)
- [SUGGEST\\_COLUMN\\_PRODUCTION\\_YEAR](#)

當這些內容的列的值匹配Google服務的providers提供的的值時，系統提供一個[深鏈接](#)到你的應用，用於詳情查看，以及指嚮應用的其他Providers的鏈接。更多討論在[在詳情頁顯示內容](#)。

你的應用的數據庫類可能定義以下的列：

```
public class VideoDatabase {  
    //The columns we'll include in the video database table  
    public static final String KEY_NAME = SearchManager.SUGGEST_COLUMN_TEXT_1;
```

```
public static final String KEY_DESCRIPTION = SearchManager.SUGGEST_COLUMN_TEXT_2;
public static final String KEY_ICON = SearchManager.SUGGEST_COLUMN_RESULT_CARD_IMAGE;
public static final String KEY_DATA_TYPE = SearchManager.SUGGEST_COLUMN_CONTENT_TYPE;
public static final String KEY_IS_LIVE = SearchManager.SUGGEST_COLUMN_IS_LIVE;
public static final String KEY_VIDEO_WIDTH = SearchManager.SUGGEST_COLUMN_VIDEO_WIDTH;
public static final String KEY_VIDEO_HEIGHT = SearchManager.SUGGEST_COLUMN_VIDEO_HEIGHT;
public static final String KEY_AUDIO_CHANNEL_CONFIG =
        SearchManager.SUGGEST_COLUMN_AUDIO_CHANNEL_CONFIG;
public static final String KEY_PURCHASE_PRICE = SearchManager.SUGGEST_COLUMN_PURCHASE_PRICE;
public static final String KEY_RENTAL_PRICE = SearchManager.SUGGEST_COLUMN_RENTAL_PRICE;
public static final String KEY_RATING_STYLE = SearchManager.SUGGEST_COLUMN_RATING_STYLE;
public static final String KEY_RATING_SCORE = SearchManager.SUGGEST_COLUMN_RATING_SCORE;
public static final String KEY_PRODUCTION_YEAR = SearchManager.SUGGEST_COLUMN_PRODUCTION_YEAR;
public static final String KEY_COLUMN_DURATION = SearchManager.SUGGEST_COLUMN_DURATION;
public static final String KEY_ACTION = SearchManager.SUGGEST_COLUMN_INTENT_ACTION;
...
...
```

當你創建從SearchManager列填充到你的數據字段時，你也必須定義\_ID去獲得每行的獨一無二的ID。

```
...
private static HashMap buildColumnMap() {
    HashMap map = new HashMap();
    map.put(KEY_NAME, KEY_NAME);
    map.put(KEY_DESCRIPTION, KEY_DESCRIPTION);
    map.put(KEY_ICON, KEY_ICON);
    map.put(KEY_DATA_TYPE, KEY_DATA_TYPE);
    map.put(KEY_IS_LIVE, KEY_IS_LIVE);
    map.put(KEY_VIDEO_WIDTH, KEY_VIDEO_WIDTH);
    map.put(KEY_VIDEO_HEIGHT, KEY_VIDEO_HEIGHT);
    map.put(KEY_AUDIO_CHANNEL_CONFIG, KEY_AUDIO_CHANNEL_CONFIG);
    map.put(KEY_PURCHASE_PRICE, KEY_PURCHASE_PRICE);
    map.put(KEY_RENTAL_PRICE, KEY_RENTAL_PRICE);
    map.put(KEY_RATING_STYLE, KEY_RATING_STYLE);
    map.put(KEY_RATING_SCORE, KEY_RATING_SCORE);
    map.put(KEY_PRODUCTION_YEAR, KEY_PRODUCTION_YEAR);
    map.put(KEY_COLUMN_DURATION, KEY_COLUMN_DURATION);
    map.put(KEY_ACTION, KEY_ACTION);
    map.put(BaseColumns._ID, "rowid AS " +
            BaseColumns._ID);
    map.put(SearchManager.SUGGEST_COLUMN_INTENT_DATA_ID, "rowid AS " +
            SearchManager.SUGGEST_COLUMN_INTENT_DATA_ID);
    map.put(SearchManager.SUGGEST_COLUMN_SHORTCUT_ID, "rowid AS " +
            SearchManager.SUGGEST_COLUMN_SHORTCUT_ID);
    return map;
}
...
...
```

在上面的例子中，注意填充SUGGEST\_COLUMN\_INTENT\_DATA\_ID字段。這是URI的一部分，指向獨一無二的內容到這一列的數據，那是URI描述的內容被存儲的最後部分。在URI的第一部分，與所有表格的列同樣，是設置在 searchable.xml 文件，用 android:searchSuggestIntentData 屬性。屬性被描述在 Handle Search Suggestions 。

如果URI的第一部分是不同於表格的每一列，你填充SUGGEST\_COLUMN\_INTENT\_DATA字段的值。當用戶選擇這個內容時，這個intent被啓動依據SUGGEST\_COLUMN\_INTENT\_DATA\_ID的混合intent數據或者 android:searchSuggestIntentData 屬性和SUGGEST\_COLUMN\_INTENT\_DATA字段值之一。

## 提供搜索建議數據

實現一個Content Provider去返回搜索術語建議到安卓TV搜索框。系統需要你的內容容器提供建議，通過調用每次一個字母類型query())方法。在query())的實現中，你的內容容器搜索你的建議數據並且返回一個光標指向你已經指定的建議列。

```
@Override
```

```

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
                    String sortOrder) {
    // Use the UriMatcher to see what kind of query we have and format the db query accordingly
    switch (URI_MATCHER.match(uri)) {
        case SEARCH_SUGGEST:
            Log.d(TAG, "search suggest: " + selectionArgs[0] + " URI: " + uri);
            if (selectionArgs == null) {
                throw new IllegalArgumentException(
                        "selectionArgs must be provided for the Uri: " + uri);
            }
            return getSuggestions(selectionArgs[0]);
        default:
            throw new IllegalArgumentException("Unknown Uri: " + uri);
    }
}

private Cursor getSuggestions(String query) {
    query = query.toLowerCase();
    String[] columns = new String[]{
        BaseColumns._ID,
        VideoDatabase.KEY_NAME,
        VideoDatabase.KEY_DESCRIPTION,
        VideoDatabase.KEY_ICON,
        VideoDatabase.KEY_DATA_TYPE,
        VideoDatabase.KEY_IS_LIVE,
        VideoDatabase.KEY_VIDEO_WIDTH,
        VideoDatabase.KEY_VIDEO_HEIGHT,
        VideoDatabase.KEY_AUDIO_CHANNEL_CONFIG,
        VideoDatabase.KEY_PURCHASE_PRICE,
        VideoDatabase.KEY_RENTAL_PRICE,
        VideoDatabase.KEY_RATING_STYLE,
        VideoDatabase.KEY_RATING_SCORE,
        VideoDatabase.KEY_PRODUCTION_YEAR,
        VideoDatabase.KEY_COLUMN_DURATION,
        VideoDatabase.KEY_ACTION,
        SearchManager.SUGGEST_COLUMN_INTENT_DATA_ID
    };
    return mVideoDatabase.getWordMatch(query, columns);
}
...

```

在你的manifest文件中，內容容器接受特殊處理。相比被標記為一個activity，它是被描述為[provider] (<http://developer.android.com/guide/topics/manifest/provider-element.html>)。provider包括 android:searchSuggestAuthority 屬性去告訴系統你的內容容器的名字空間。並且，你必須設置它的 android:exported 屬性為 "true"，這樣安卓全局搜索能用它返回的搜索結果。

```

<provider android:name="com.example.android.tvleanback.VideoContentProvider"
          android:authorities="com.example.android.tvleanback"
          android:exported="true" />

```

## 處理搜索建議

你的應用必須包括[res/xml/searchable.xml](#)文件去配置搜索建議設置。它包括android:searchSuggestAuthority屬性去告訴系統內容容器的名字空間。這必須匹配在 `AndroidManifest.xml` 文件的[provider] (<http://developer.android.com/guide/topics/manifest/provider-element.html>)元素的android:authorities 屬性的字符串值。

`searchable.xml`文件必須也包含在 "android.intent.action.VIEW" 的android:searchSuggestIntentAction值去定義提供自定義建議的intent action。這與提供一個搜索術語的intent action不同，下面解釋。查看[Declaring the intent action](#)用另一種方式去定義建議的intent action。

同intent action一起，你的應用必須提供你定義的android:searchSuggestIntentData屬性的intent數據。這是指向內容

的URI的第一部分。它描述在填充的內容表格中URI所有共同列的部分。URI的獨一無二的部分用 `SUGGEST_COLUMN_INTENT_DATA_ID` 字段建立每一列，以上被描述在 [識別列](#)。查看 [Declaring the intent data](#) 用另一種方式去定義建議的intent數據。

並且，注意 `android:searchSuggestSelection="?"` 屬性為特定的值。這個值作為 `query()` 方法 `selection` 參數。方法的問題標記(?)值被代替為請求文本。

最後，你也必須包含 `android:includeInGlobalSearch` 屬性值為 "true"。這是一個 `searchable.xml` 文件的例子：

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"  
    android:label="@string/search_label"  
    android:hint="@string/search_hint"  
    android:searchSettingsDescription="@string/settings_description"  
    android:searchSuggestAuthority="com.example.android.tvleanback"  
    android:searchSuggestIntentAction="android.intent.action.VIEW"  
    android:searchSuggestIntentData="content://com.example.android.tvleanback/video_database_leanback"  
    android:searchSuggestSelection="?"  
    android:searchSuggestThreshold="1"  
    android:includeInGlobalSearch="true"  
>  
</searchable>
```

## 處理搜索術語

一旦搜索框有一個字匹配到了應用列中的一個（被描述在上文的 [識別列](#)），系統啓動 `ACTION_SEARCH` intent。你應用的 `activity` 處理 intent 搜索列的給定的字段資源，並且返回一個那些內容項的列表。在你的 `AndroidManifest.xml` 文件中，你指定的 `activity` 處理 `ACTION_SEARCH` intent，像這樣：

```
...  
<activity  
    android:name="com.example.android.tvleanback.DetailsActivity"  
    android:exported="true">  
  
    <!-- Receives the search request. -->  
    <intent-filter>  
        <action android:name="android.intent.action.SEARCH" />  
        <!-- No category needed, because the Intent will specify this class component -->  
    </intent-filter>  
  
    <!-- Points to searchable meta data. -->  
    <meta-data android:name="android.app.searchable"  
        android:resource="@xml/searchable" />  
    </activity>  
...  
    <!-- Provides search suggestions for keywords against video meta data. -->  
    <provider android:name="com.example.android.tvleanback.VideoContentProvider"  
        android:authorities="com.example.android.tvleanback"  
        android:exported="true" />  
...
```

`activity` 必須參考 `searchable.xml` 文件描述可搜索的設置。用 [全局搜索框](#)，`manifest` 必須描述 `activity` 應該收到的搜索請求。`manifest` 必須描述 [`provider`](<http://developer.android.com/guide/topics/manifest/provider-element.html>) 元素，詳細被描述在 `searchable.xml` 文件。

## 深鏈接到應用的詳情頁

如果你有設置 [處理搜索建議](#) 描述的搜索配置和填充

`SUGGEST_COLUMN_TEXT_1`, `SUGGEST_COLUMN_CONTENT_TYPE`和`SUGGEST_COLUMN_PRODUCTION_YEAR`字段到識別列，一個深鏈接去查看詳情頁的內容。當用戶選擇一個搜索結果時，詳情頁將打開。如圖1。

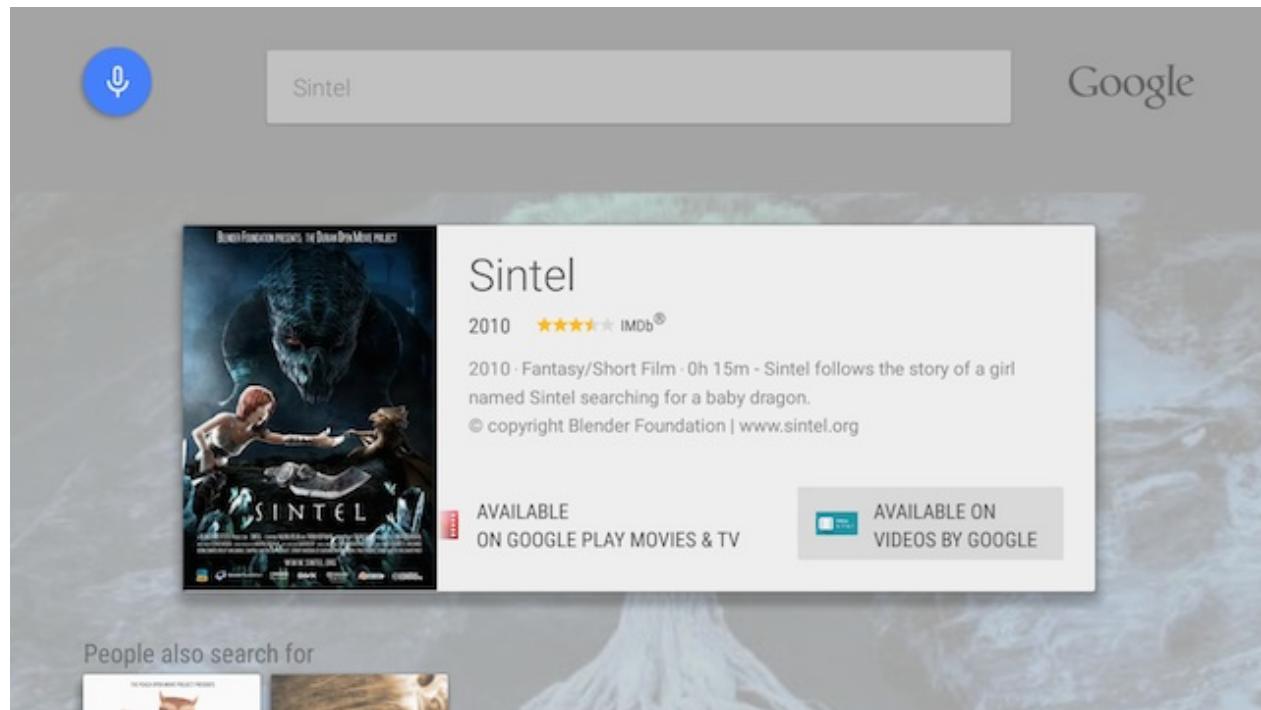


圖1 詳情頁顯示一個深鏈接為Google(Leanback)的視頻代碼。Sintel: © copyright Blender Foundation, [www.sintel.org](http://www.sintel.org).

當用戶選擇你的應用鏈接，“Available On”按鈕被標識在詳情頁，系統啓動activity處理ACTION\_VIEW（在searchable.xml文件設置android:searchSuggestIntentAction值為“android.intent.action.VIEW”）。

你也能設置用戶intent去啓動你的activity，這個在[在安卓Leanback示例代碼應用](#)中演示。注意示例應用啓動它自己的LeanbackDetailsFragment去顯示被選擇媒體的詳情，但是你應該啓動activity去播放媒體。立即去保存用戶的另一次或兩次點擊。

---

[下一節: 使TV應用是可被搜索的](#)

# TV應用內搜索

編寫:[awong1900](#) - 原文:<http://developer.android.com/training/tv/discovery/in-app-search.html>

當用TV用媒體應用時，用戶腦中通常有特定的內容。如果你的應用包含一個大的內容目錄，為用戶找到他們想找到的內容時，用特定的標題瀏覽可能不是最有效的方式為。一個搜索界面能幫助你的用戶獲得他們想快速瀏覽的內容。

[Leanback support library](#)提供一套類庫去使用標準的搜索界面。在你的應用內使用類庫，可以和TV其他搜索功能，如聲音輸入，獲得一致性。

這節課討論如何在你的應用中用Leanback支持類庫提供搜索界面。

## 添加搜索操作

當你用[BrowseFragment](#)類做一個媒體瀏覽界面時，你能使用搜索界面作為用戶界面的一個標準部分。當你設置[View.OnClickListener](#)在[BrowseFragment](#)對象時，搜索界面作為一個圖標出現在佈局中。接下來的示例代碼展示了這個技術。

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.browse_activity);  
  
    mBrowseFragment = (BrowseFragment)  
        getSupportFragmentManager().findFragmentById(R.id.browse_fragment);  
  
    ...  
  
    mBrowseFragment.setOnSearchClickedListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            Intent intent = new Intent(BrowseActivity.this, SearchActivity.class);  
            startActivity(intent);  
        }  
    });  
  
    mBrowseFragment.setAdapter(buildAdapter());  
}
```

\*\*Note\*\*: You can set the color of the search icon using the `setSearchAffordanceColor(int)`.-->

Note：你能設置搜索圖標的顏色用[setSearchAffordanceColor\(int\)](#)。

## 添加搜索輸入和結果展示

當用戶選擇搜索圖標，系統通過定義的intent關聯一個搜索activity。你的搜索activity應該用包括[SearchFragment](#)的線性佈局。這個fragment必須實現[SearchFragment.SearchResultProvider](#)界面去顯示搜索結果。

接下來的示例代碼展示瞭如何擴展[SearchFragment](#)類去提供搜索界面和結果：

```
public class MySearchFragment extends SearchFragment  
    implements SearchFragment.SearchResultProvider {  
  
    private static final int SEARCH_DELAY_MS = 300;  
    private ArrayAdapter mRowsAdapter;
```

```
private Handler mHandler = new Handler();
private SearchRunnable mDelayedLoad;

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

    mRowsAdapter = new ArrayObjectAdapter(new ListRowPresenter());
    setSearchResultProvider(this);
    setOnItemClickedListener(getDefaultItemClickedListener());
    mDelayedLoad = new SearchRunnable();
}

@Override
public ObjectAdapter getResultsAdapter() {
    return mRowsAdapter;
}

@Override
public boolean onQueryTextChange(String newQuery) {
    mRowsAdapter.clear();
    if (!TextUtils.isEmpty(newQuery)) {
        mDelayedLoad.setSearchQuery(newQuery);
        mHandler.removeCallbacks(mDelayedLoad);
        mHandler.postDelayed(mDelayedLoad, SEARCH_DELAY_MS);
    }
    return true;
}

@Override
public boolean onQueryTextSubmit(String query) {
    mRowsAdapter.clear();
    if (!TextUtils.isEmpty(query)) {
        mDelayedLoad.setSearchQuery(query);
        mHandler.removeCallbacks(mDelayedLoad);
        mHandler.postDelayed(mDelayedLoad, SEARCH_DELAY_MS);
    }
    return true;
}
}
```

上面的示例代碼展示了在分開的線程用獨立的 `SearchRunnable` 類去運行搜索請求。這個技巧是從阻塞的主線程保持了潛在的慢運行請求。

---

[下一節: 創建TV遊戲](#)

# 創建TV遊戲應用

編寫:dupengwei - 原文:<http://developer.android.com/training/tv/games/index.html>

電視屏幕為手機遊戲開發者提供了大量的新思考。這些領域包括它的大尺寸，它的控制方案和所有玩家可以同時觀看的事實。

## 顯示器

開發TV屏幕遊戲時有兩點要記住，就是TV屏幕具有共享顯示器的特性，和橫向設計遊戲的需求。

### 考慮共享顯示

客廳TV帶來了多人遊戲的設計挑戰，客廳TV遊戲時所有玩家都可以看到。這個問題與遊戲，特別是依靠每個玩家用於隱藏信息的遊戲（如紙牌遊戲、戰略遊戲）息息相關。你可以通過實現一些機制來解決一個玩家竊取另一玩家信息的問題。這些機制是：

- 屏幕罩可以幫助隱藏信息。例如，在一個回合制遊戲，像單詞或卡片遊戲，一次只有一個玩家能看到顯示的內容。當這個玩家完成一個步驟，遊戲允許他用一個能阻礙其他人看到祕密信息的罩遮住屏幕。當下一個玩家開始操作，這個罩就會打開顯示他自己的信息。
- 在手機或平板電腦上運行一個夥伴app作為第二屏幕，通過這種方式讓玩家隱藏信息。

### 支持橫向顯示

TV總是單向顯示的：你不能翻轉它的屏幕，且沒有縱向顯示。要總是以橫向顯示模式設計你的TV遊戲。

## 輸入設備

TV沒有觸摸屏接口，所以更重要的是獲取控制要正確，並確保玩家使用起來要直觀和有趣。處理控制器還介紹了其他一些問題需要注意，如跟蹤多個控制器，，處理斷開要適當。

### 支持D-pad控制

圍繞方向鍵（D-pad）控制來計劃你的控制方案，因為這種控制是Android TV設備的默認設置。玩家需要在遊戲的所有方面使用方向鍵（D-pad）——不僅僅是控制核心遊戲設置，而且能導航菜單和廣告。因此，你還應該確保你的Android TV遊戲不能涉及觸摸屏。例如，一個Android TV遊戲不應該告訴玩家> 點擊這裏繼續。如何塑造玩家使用控制器與遊戲進行互動的方式將是實現良好用戶體驗的關鍵：

- 通信控制器的要求。利用安卓市場上app的產品描述將控制器的期望傳達給玩家們。如果一個遊戲使用搖桿遊戲手柄比只用一個方向鍵更合適，請將這一事實說清楚。玩家使用一個不適合遊戲的控制器玩遊戲很可能導致遊戲體驗欠佳，從而對遊戲的評價造成不利影響。
- 使用一致的按鈕映射。直觀和靈活的按鈕映射是良好用戶體驗的關鍵。例如，你應該遵守使用A按鈕接受，而B按鈕取消的既定習慣。你也可以提供重映射形式方面的靈活性。關於按鈕映射的更多信息，參見[Handling Controller Actions](#)。
- 檢測控制器功能並相應地調整。查詢控制器的能力以優化控制器和遊戲直接的匹配程度。例如，你可能打算讓一個玩家通過搖晃控制器來控制一個對象。然而，如果玩家的控制器缺少加速計和陀螺儀硬件設施，搖晃控制器並不會產生效果。所以，你的遊戲應該檢查控制器，如果該控制器不支持運行檢測，則切換到另一個可用的控制方案。更多關於檢測控制器功能的信息，參見[Controllers Across Android Versions](#)。

## 提供適當的後退按鈕的行為

返回按鈕不應該作為切換。例如，不能使用它打開和關閉一個菜單。它應該只能導航後退，breadcrumb-style，玩家之前訪問過屏幕頁面，例如：遊戲界面>遊戲暫停界面>遊戲主界面>Android主界面。由於返回按鈕應該只能進行線性導航（後退），你可以使用返回按鈕離開一個遊戲內菜單（由不同的按鈕打開），回到遊戲界面。更多關於導航設計的信息，參見[Navigation with Back and Up](#)。學習更多關於實現的信息，參見[Providing Proper Back Navigation](#)。

## 使用適當的按鈕

並不是所有的遊戲控制器提供開始、搜索或菜單按鈕。確保你的UI不取決於這些按鈕的使用。

## 處理多個控制器

當多個玩家玩遊戲，每個都有他或她自己的控制器，做好每對“玩家-控制器”的映射是很重要的。關於如何實現“控制器數量”識別的信息，參見[Input Devices](#)。

## 處理控制器的斷開

當控制器從遊戲中斷開時，遊戲應該暫停，並彈出一個對話框促使斷開的玩家重新連接他或她的控制器。對話框還應提供排除故障的提示（如，一個彈出的對話框告訴玩家“檢查你的藍牙連接”）關於實現輸入設備支持的更多信息，參見[Handling Controller Actions](#)。具體關於藍牙連接的信息，參見[Bluetooth](#)。

## 展示控制器說明

如果你的遊戲提供了可視化的遊戲控制說明，控制器圖片應該是免費的、品牌化的，並且只能包含與Android兼容的按鈕。Android兼容的控制器樣圖，點擊[Android TV Gamepad Template \(ZIP\)](#)下載。它包含一個黑底的白色控制器和一個白底的黑色控制器，是一個PNG類型的Adobe®Illustrator®文件。

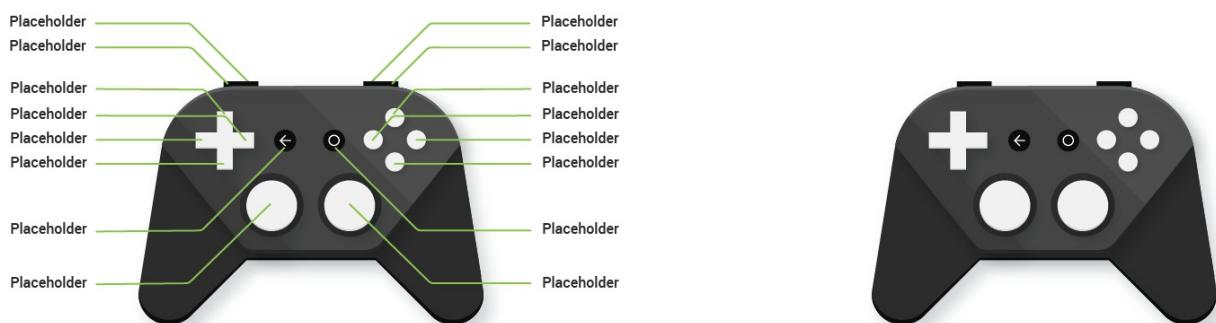


Figure 1. 控制器說明的示例請使用[Android TV Gamepad Template \(ZIP\)](#)

## Manifest

有一些特殊的東西應該包含在遊戲的Android Manifest裏。

## 在屏幕主界面顯示遊戲

Android TV主界面採用單獨一行來顯示遊戲，與常規應用分開顯示。為了讓遊戲出現在遊戲列表，設置遊戲的manifest清單的標簽下的 `android:isGame` 屬性為 "true" 。例如：

```
<application  
...  
    android:isGame="true"  
...
```

>

## 聲明遊戲控制器支持

遊戲控制器對於TV設備的用戶來說可能不是有效的。為了適當的通知用戶，遊戲需要（或只支持）一個控制器，你必須在app的manifest裏包含這些條目。如果你需要一個遊戲控制器，你必須在app的manifest中包含以下條目：

```
<uses-feature android:name="android.hardware.gamepad"/>
```

如果你的遊戲使用了一個遊戲控制器，但是不需要，在app的manifest裏包含以下的功能條目：

```
<uses-feature android:name="android.hardware.gamepad" android:required="false"/>
```

更多關於manifest條目的信息，參見[App Manifest](#)。

## Google Play Game 服務

如果你的遊戲集成了Google Play Game 服務，你應該記住一些關於成果的注意事項，登錄，保存遊戲，和多人遊戲。

### 成就

你的遊戲應包含至少5個(可獲取的)成果。只有一個用戶從一個受支持的輸入設備控制遊戲應該能夠獲得成就。關於成就的更多信息以及如何實現，參見[Achievements in Android](#)。

### 登錄

你的遊戲應該試圖在啓動的時候讓用戶登錄。如果玩家連續幾次拒絕登錄後，遊戲應該停止詢問。學習更多關於登錄的信息在[Implementing Sign-in on Android](#)。

### 保存

使用Google Play Services[保存遊戲](#)來存儲保存的遊戲。你應該將保存的遊戲綁定到一個特定的谷歌帳號，作為唯一標識，甚至在跨設備時也不受影響。無論玩家使用手機或TV，遊戲應該可以從同一個用戶帳號獲取到保存的遊戲信息。

你也應該在你的遊戲的UI提供一個選項，讓玩家刪除本地和雲存儲端的數據。你可能把選項放在遊戲的設置界面。使用Play Services保存遊戲的實現細節，參見[Saved Games in Android](#)

### 多人遊戲

一個遊戲要提供多人遊戲體驗，必須允許至少2個玩家進入一個房間。進一步瞭解Android的多人遊戲信息，參見Android developer網站的[Real-time Multiplayer](#)和[Turn-based Multiplayer](#)文檔。

### 退出

提供一個一致和明顯的UI元素，讓用戶適當的退出遊戲。這個元素應該用方向鍵導航按鈕訪問，這樣做而不是依賴Home鍵提供退出功能，是因為在使用不同的控制器時，若依賴Home鍵提供退出功能，這既不一致也不可靠。

## Web

---

不要讓Android TV的遊戲瀏覽網頁。Android TV不支持web瀏覽器。

Note：你可以使用[WebView](#)類實現登錄像Google+ 和 Facebook這樣的服務。

## Networking

---

遊戲經常需要更大的帶寬提供最佳的性能，許多用戶寧願選擇以太網而不願選擇WiFi來提供性能。你的app應該對以太網和WiFi連接都進行檢查。如果你的app只針對電視，你不需要檢查3G/LTE服務，而移動app則需要檢查3G/LTE服務。

# 創建TV直播應用

編寫:dupengwei - 原文:<http://developer.android.com/training/tv/tif/index.html>

看電視直播節目和其他連續的、基於頻道的內容是TV體驗的主要部分。Android 通過Android 5.0中的TV Input Framework支持直播視頻內容的接收和重放（API Level 21）。該框架提供了一個統一的方法，從硬件源（如HDMI端口和內置調諧器）和軟件源（如流傳在互聯網上的視頻）接收音頻和視頻內容。

該框架能使開發人員通過實現TV輸入服務定義直播TV輸入源。該服務發佈一個頻道和節目列表到一個TV Provider上。電視設備的直播電視應用從TV Provider獲取可用的頻道和節目列表並顯示給用戶。當用戶選擇某個特定的頻道，直播TV應用軟件通過TV Input Manager為相關TV輸入服務創建一個會話，並告訴TV輸入服務調整到請求頻道，然後將內容顯示到TV應用軟件提供的顯示器上。

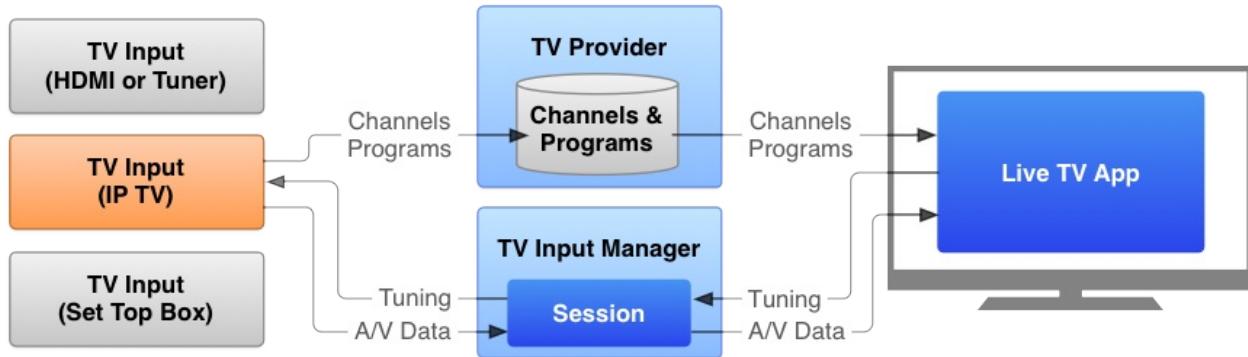


圖1.電視輸入框架功能圖

TV Input Framework 的設計目的是提供各種各樣的TV輸入源並把它們整合到一個單一的用戶界面供用戶瀏覽、查看和享受內容。為你想要傳播的節目構建一個TV輸入服務之後，用戶可以更加輕易地通過TV設備收看這些節目。

更多關於TV輸入框架的信息，請參考[android.media.tv](#)。

# TV應用清單

編寫:awong1900 - 原文:<http://developer.android.com/training/tv/publishing/checklist.html>

用戶喜歡的TV應用應是體驗一致的，有邏輯的和可預測的。他們可以在應用內四處瀏覽，並且不會迷失在應用中，或者重設UI導致重頭開始。用戶欣賞乾淨的，有色彩的和起作用的界面，這樣的體驗會很好。把這些想法放在腦子中，你能創造適合安卓TV的應用並達到用戶的期望。

這個清單覆蓋了應用和遊戲的開發的主要方面去確保你的應用提供了最好的體驗。額外的遊戲注意事項僅被包含在遊戲小節。

關於Google Play中安卓TV應用的質量標準，參考[TV App Quality](#)。

## TV形式的支持

這些清單項目使用在遊戲和應用中。

1. 確定manifest的主activity使用 `CATEGORY_LEANBACK_LAUNCHER`。 查看[Declare a TV Activity](#)。
2. 提供每種語言的主屏幕橫幅支持。
  - Launcher應用橫幅大小為320x180 px
  - 橫幅資源放在 `drawables/xhdpi` 目錄
  - 橫幅圖像包含本地化的文本去識別應用。 查看[Provide a home screen banner](#)。
3. 消除不支持的硬件要求。 查看[Declaring hardware requirements for TV](#)。
4. 確保沒有隱式的權限需求。 查看[Declaring permissions that imply hardware features](#)。

## 用戶界面設計

這些清單項使用在遊戲和應用中。

1. 提供適合橫屏模式的佈局資源。 查看[Build Basic TV Layouts](#)。
2. 確保文本和控件在一定距離外看是足夠大的。 查看[Build Useable Text and Controls](#)。
3. 為HDTV屏幕提供高分辨率的位圖和圖標。 查看[Manage Layout Resources for TV](#)。
4. 確保你的圖標和logo符合安卓TV的規範。 查看[Manage Layout Resources for TV](#)。
5. 允許佈局使用overscan。 查看[Overscan](#)。
6. 使每一個佈局元素都能用D-pad和遊戲控制器操作。 查看[Creating Navigation](#) 和[Handling Controllers](#)。
7. 當用戶通過文本搜索時改變背景圖像。 查看[Update the Background](#)。
8. 在Leanback fragments中定製背景顏色去匹配品牌。 查看[Customize the Card View](#)。
9. 確保你的UI不需要觸摸屏。 查看[Touch screen and Declare touch screen not required](#)。
10. 遵循有效的廣告的指導。 查看[Provide Effective Advertising](#)。

## 搜索和發現內容

這些清單項使用在遊戲和應用中。

1. 在安卓TV全局搜索框中提供搜索結果。 查看[Provide Data](#)。
2. 提供TV特定數據字段的搜索。 查看[Identify Columns](#)。
3. 確保應用的詳情屏幕有可發現的內容以便用戶立即開始觀看。 查看[Display Your App in the Details Screen](#)。
4. 放置相關的，可操作的內容和目錄在主屏幕，使用戶容易的發現內容。 查看[Recommending TV Content](#)。

# 遊戲

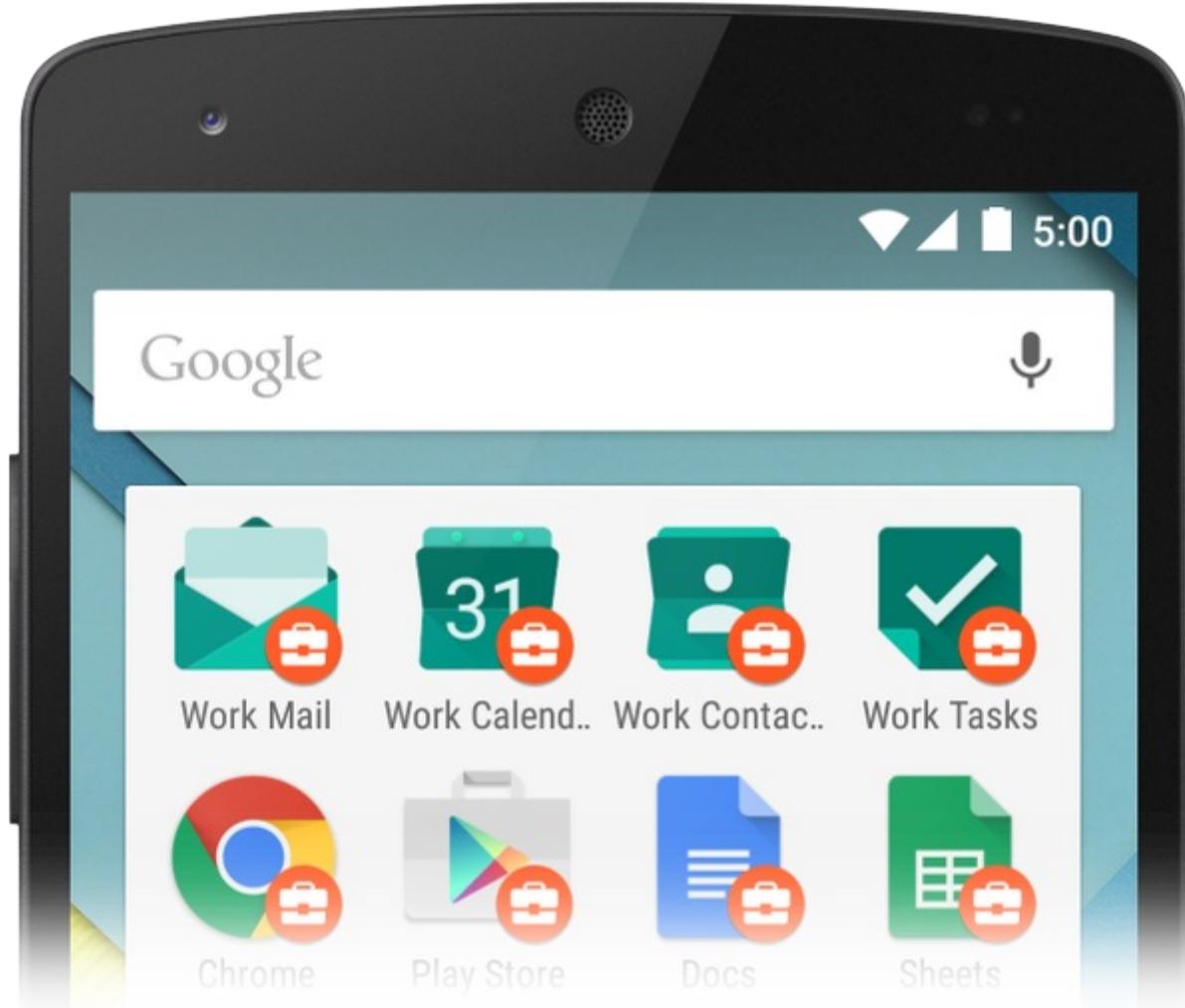
---

這些清單項目使用在遊戲。

1. 在manifest中用 `isGame` 標記讓遊戲顯示在主屏幕上。 查看[Show your game on the home screen](#)。
  2. 確保遊戲控制器支持不依靠開始，選擇，或者菜單鍵操作(不是所有控制器有這些按鍵)。 查看[Input Devices](#)。
  3. 使用通常的遊戲手柄佈局（不包括特殊的控制器品牌）去顯示遊戲按鍵示意圖。 查看[Show controller instructions](#)。
  4. 檢查網絡和WiFi連接。 查看[Networking](#)。
  5. 提供給用戶清晰的退出提示。 查看[Exit](#).
-

# 創建企業級應用

編寫:craftsmanBai - <http://z1ng.net> - 原文:<http://developer.android.com/training/enterprise/index.html>



Android框架提供安全支持、數據分離、企業環境管理的功能。作為應用開發者，通過適當地處理企業安全和功能限制，你可以讓你的應用程序吸引更多的企業客戶。也可以修改你的應用使技術管理員可遠程配置使用企業資源。

為了幫助企業將安卓設備和應用程序進入工作場所，Google通過Android for Work為設備的分配和管理提供了一套API和服務。通過這項計劃，企業可以連接到企業移動性管理（EMM）供應商，將Android整合到工作中。

通過下面的鏈接獲取，可以瞭解更多關於如何更新您的Android應用程序來支持企業環境或建立企業解決方案的信息。

## 企業級應用開發

瞭解在企業環境中如何使您的應用程序運行順暢，限制設備的功能和數據訪問。通過加入限制進一步支持企業使用你的app，讓管理員可以遠程配置使用你的應用程序：

確保與管理兼容：

<http://developer.android.com/training/enterprise/app-compatibility.html>

加入應用限制：

<http://developer.android.com/training/enterprise/app-restrictions.html>

應用限制計劃：

<http://developer.android.com/samples/AppRestrictionSchema/index.html>

應用限制執行者：

<http://developer.android.com/samples/AppRestrictionEnforcer/index.html>

## 設備與應用管理

---

學習如何為應用程序建立策略控制器，使企業的技術管理人員來管理設備，管理企業應用程序，並提供訪問公司資源的權限：

建立工作策略控制：

<http://developer.android.com/training/enterprise/work-policy-ctrl.html>

基本管理模型：

<http://developer.android.com/samples/BasicManagedProfile/index.html>

# Ensuring Compatibility with Managed Profiles

---

| 編寫: - 原文:

待認領進行編寫，有意向的小夥伴，可以直修改對應的markdown文件，進行提交！

# Implementing App Restrictions

---

| 編寫: - 原文:

待認領進行編寫，有意向的小夥伴，可以直修改對應的markdown文件，進行提交！

# Building a Work Policy Controller

---

| 編寫: - 原文:

待認領進行編寫，有意向的小夥伴，可以直接修改對應的markdown文件，進行提交！

# Android交互設計

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/best-ux.html>

These classes focus on the best Android user experience for your app. In some cases, the success of your app on Android is heavily affected by whether your app conforms to the user's expectations for UI and navigation on an Android device. Follow these recommendations to ensure that your app looks and behaves in a way that satisfies Android users.

## Designing Effective Navigation

How to plan your app's screen hierarchy and forms of navigation so users can effectively and intuitively traverse your app content using various navigation patterns.

## Implementing Effective Navigation

How to implement various navigation patterns such as swipe views, a navigation drawer, and up navigation.

## Notifying the User

How to display messages called notifications outside of your application's UI.

## Adding Search Functionality

How to properly add a search interface to your app and create a searchable database.

## Making Your App Content Searchable by Google

How to enable deep linking and indexing of your application content so that users can open this content directly from their mobile search results.

# 設計高效的導航

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/index.html>

設計開發 App 的起初步驟之一就是決定用戶能夠在App上看到什麼和做什麼。一旦你知道用戶在App上和哪種內容互動，下一步就是去設計容許用戶在 App 的不同內容塊間切換，進入，回退的交互。

本課程演示如何為你的應用規劃出高標準的界面層次，然後為它選擇適宜的導航形式來允許用戶高效而直觀的瀏覽內容。按粗略的先後順序，每堂課涵蓋Android應用導航交互設計過程中的不同階段。學過這些課之後，你應該可以應用這些列出的方法和設計範例到你自己的應用中，為你的用戶提供一致的導航體驗了。

## Lessons

- [規劃界面和他們之間的關係](#)

學習如何選擇你應用應該包含的界面。並且學習如何選擇其他界面可直達的界面。這節課介紹了一個假想的新聞應用為以後課程作例子。

- [為多種大小的屏幕進行規劃](#)

學習如何在大屏設備上組合相關界面來優化用戶可視界面空間。

- [提供向下和橫嚮導航](#)

學習容許用戶深入某一層或者在內容層次間橫跨的技巧。而且學習一些特定導航 UI 元素在不同情景下的優缺點和最佳用法。

- [提供向上和歷史導航](#)

學習如何容許用戶在內容層級向上導航。並且學習 Back 鍵和歷史導航的最佳做法，也即導航到和層次無關的之前的畫面。

- [綜合：設計樣例 App](#)

學習如何創建界面的 Wireframe（線框圖，模糊的圖形模型）來代表新聞應用基於設想信息模型的界面。這些 Wireframe 利用上述課程討論的導航元件來展示直觀高效導航。

# 規劃界面和他們之間的關係

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/screen-planning.html>

多數 App 都有一種內在的信息模型，它能被表示成一個用對象類型構成的樹或圖。更淺顯的說，你可以畫一個有不同類型信息的圖，這些信息代表用戶在你 App 裏用戶與之互動的各種東西。軟件工程師和數據架構師經常使用實例-關係圖（Entity-Relationship Diagram，ERD）描述一個應用的信息模型。

讓我們考慮一個讓用戶瀏覽一羣已分類好的新聞事件和圖片的應用例子。這種 App 一個可能的模型如下 ERD 圖。

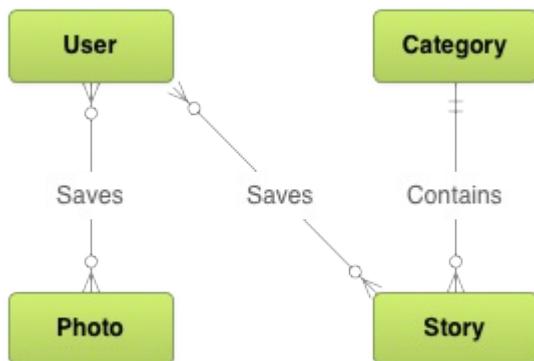


Figure 1. 新聞應用例子的實例關係圖

## 創建一個界面列表

一旦你定義了信息模型，你就可以開始定義那些能使用戶在你的 App 中有效地發掘，查看和操作數據的上下文環境了。實際上，其中一種方法就是確定供用戶導航和交互數據所需的 界面完備集（歸納了所有界面的集合）。但我們實際發現的界面集合應該根據目標設備變化。在設計過程中早點考慮到這點很重要，這樣可以保證程序可以適應運行環境。

在我們的例子中，我們想讓用戶查看，保存和分享分類好了的新聞和圖片。下面是涵蓋了這些用例的界面完備列表。

- 用來訪問新聞和圖片的 Home 或者 "Launchpad" 畫面
- 類別列表
- 某個分類下的新聞列表
- 新聞詳情 View（在這裏我們可以保存和分享）
- 圖片列表，不分類
- 圖片詳情 View（在這裏我們可以保存和分享）
- 所有保存項列表
- 圖片保存列表
- 新聞保存列表

## 圖示界面關係

現在我們可以定義界面間的有向關繫了。一個從界面 A 指向另一個界面 B 的箭頭表示通過用戶在畫面 A 的某個交互動作可直達畫面 B。一旦我們定義了界面集和他們之間的關係，我們可以將他們一起全部表示在一張界面圖中了：

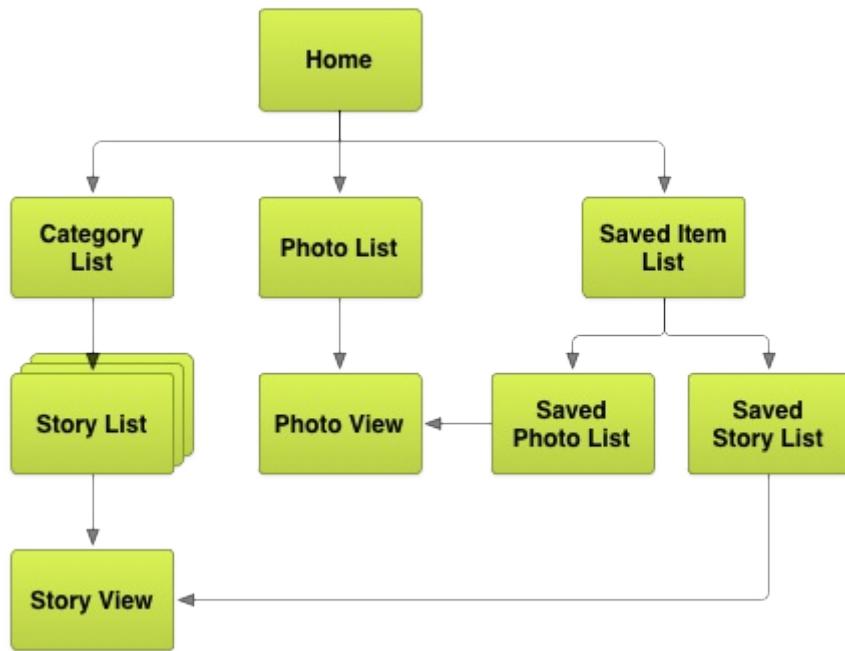


Figure 2. 新聞應用例子的界面完備Map

如果之後我們想允許用戶提交新聞事件或者上傳圖片，我們可以在圖中加額外的界面。

## 脫離簡陋設計

這時，我們可以據這張完備的界面圖設計一個功能完備應用了。可以由列表和導向子界面的按鈕構成一個簡單的UI：

- 導向不同頁面的按鈕（例如，新聞，圖片，保存的項目）
- 縱向列表表示集合（例如，新聞列表，圖片列表，等等）
- 詳細信息（例如，新聞 View，圖片 View，等等）

但是，你可以利用屏幕組合技術和更高深導航元素以一種更直觀，設備更理解的方式呈現內容。下節課，我們探索屏幕組合技術，比如為平板而生的多視窗（Multi-pane）佈局。之後，我將深入講解更多不同的 Android 常見導航模式。

[下節課：規劃多種觸屏大小](#)

# 為多種大小的屏幕進行規劃

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/multiple-sizes.html>

雖然上節中的界面完備圖在手持設備和相似大小設備上可行，但並不是和某個設備因素綁死的。Android應用需要適配一大把不同類型的設備，從3"的手機到10"的平板到42"的電視。這節課中我們探討把完備圖中不同界面組合起來的策略和原因。

Note: 為電視設計應用程序還需要注意其他的因素，包括互動方式（就是說，它沒觸屏），長距離情況下文本的可讀性，還有其他的。雖然這個討論在本課範疇之外，你仍然可以在 [Google TV](#) 文檔的設計模式中找到有關為電視設計的信息。

## 用多視窗佈局（Multi-pane Layout）組合界面

多視窗佈局（Multi-pane Layout）設計

設計指南請閱讀 Android 設計部分的[多視窗佈局](#)。

3 到 4英寸的屏幕通常只適合每次展示單個縱向內容視窗，一個列表，或某列表項的具體信息，等等。所以在這些設備上，界面通常對映於信息層次上的某一級（類別 → 列表 → 詳情）。

更大的諸如平板和電視上的屏幕通常會有更多的可用界面空間，並且他們能夠展示多個內容視窗。橫屏中，視窗從左到右以細節程度遞增的順序排列。因常年使用桌面應用和網站，用戶變得特別適應大屏上的多視窗。很多桌面應用和網站提供左側導航視窗，或者使用總/分（master/detail）兩個視窗佈局。

為了符合這些用戶期望，通常很有必要為平板提供多個信息視窗來避免留下過多空白或無意間引入尷尬的交互，比如  $10 \times 0.5"$  按鈕。

下面圖例示範了當把 UI 設計遷移到更大的佈局時出現的一些問題，並且展示瞭如何用多視窗佈局來處理這些問題：

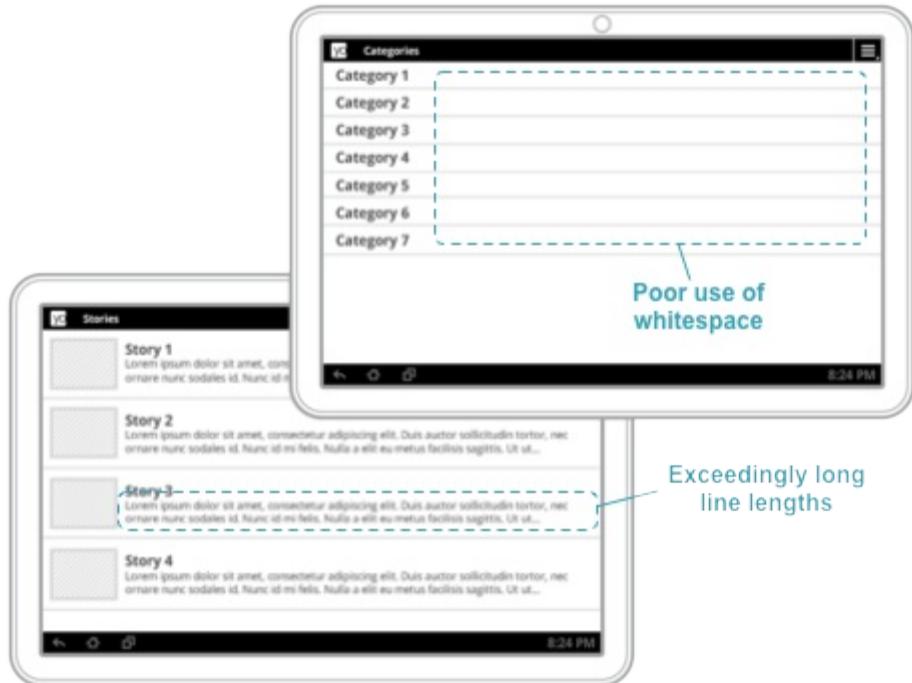


圖 1. 大橫屏使用單視窗導致尷尬的空白和過長行。



圖 2. 橫屏多視窗佈局產生更好的視覺平衡，更大的效用和可讀性。

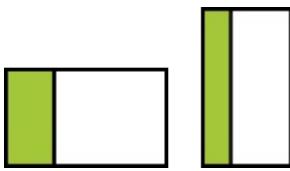
實現提醒：當決定好了區分使用單視窗佈局和多視窗佈局的屏幕大小基準線後，你就可以為不同屏幕大小區間（例如 `large/xlarge`）或最低屏幕寬度（例如 `sw600dp`）提供不同的佈局了。

實現提醒：單一界面被實現為 `Activity` 的子類，單獨的內容視窗則可實現為 `Fragment` 的子類。這樣最大化了跨越不同結構因素和不同屏幕內容的代碼複用。

## 為不同平板方向設計

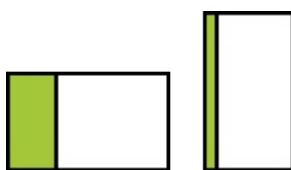
雖然現在我們還沒有開始在我們的屏幕上排布 UI 元素，但現在很是時候來考慮下我們的多視窗界面如何適配不同的設備方向了。多視窗佈局在橫屏時表現的非常棒，因為有大量可用的橫向空間。然而，在豎屏時，你的橫向空間被限制了，所以你需要為這個方向設計一個單獨的佈局。

下面是一些創建豎屏佈局的常見策略：



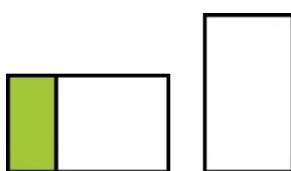
- 伸縮

最直接的策略就是簡單地伸縮每個視窗的寬度來最好地在豎屏下的呈現內容。視窗可設置固定寬度或佔可用界面寬度的一定比例。



- 展開/摺疊

伸縮策略的一個變種就是在豎屏中摺疊左側視窗的內容。當遇到總/分（master/detail）視窗中左側（master）視窗包含易摺疊列表項時，這個策略很有效。以一個實時聊天應用為例。橫屏中，左側列表可能包含聊天聯繫人的照片，姓名和在線狀態。在豎屏中，橫向空間可以將通過隱藏聯繫人姓名而且只顯示照片和在線狀態的提示圖標的方式來摺疊。也可以選擇性的提供展開控制，這種控制允許用戶展開左側視窗或反向操作。



- 顯示/隱藏

這個方案中，左側視窗在豎屏模式下完全隱藏。然而，為了保證你界面的功能等價性，左側視窗必須功能可見（比如，添一個按鈕）。通常適合在 Action Bar 使用 Up 按鈕（詳見Android設計的模式文檔）來展示左側視窗，這將在之後討論。



- 堆疊

最後的策略就是在豎屏時垂直地堆放你一般橫向排布的視窗。當你的視窗不是簡單的文本列表，或者當有多個內容模塊與基本內容視窗同時運行時，這個策略很奏效。但是當心使用這個策略時出現上面提到的尷尬的空白問題。

## 組合界面圖中的界面

既然現在我們能夠通過提供大屏設備上的多視窗佈局來組合單獨的界面，那麼就讓我們把這個技術應用到我們[上節課](#)界面完備圖上吧，這樣我們應用的界面層次在這類設備上變得更具體了：

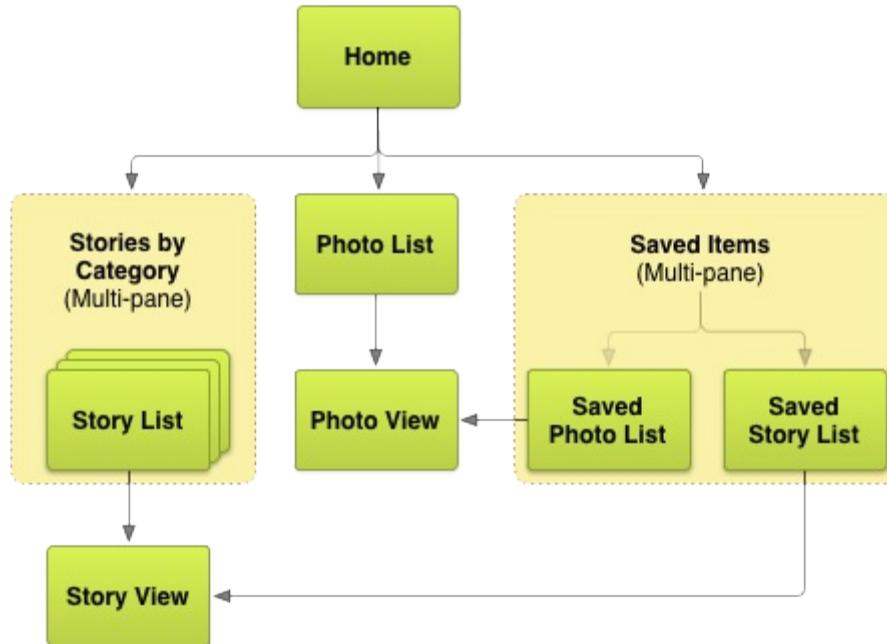


Figure 3. 更新後新聞應用例子的界面完備Map

下節課我們將討論 向下 和 橫向 導航，並且探討更多方法來組合界面使能最大化應用 UI 的直觀性和內容獲取速度。

[下一節：提供向下和橫嚮導航](#)

# 提供向下與橫嚮導航

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/descendant-lateral.html>

一種提供查看應用整體界面結構的方式就是顯示層級導航。這節課我們討論 向下導航，它允許用戶進入子界面。我們還討論 橫向 導航，它允許用戶訪問同級界面。

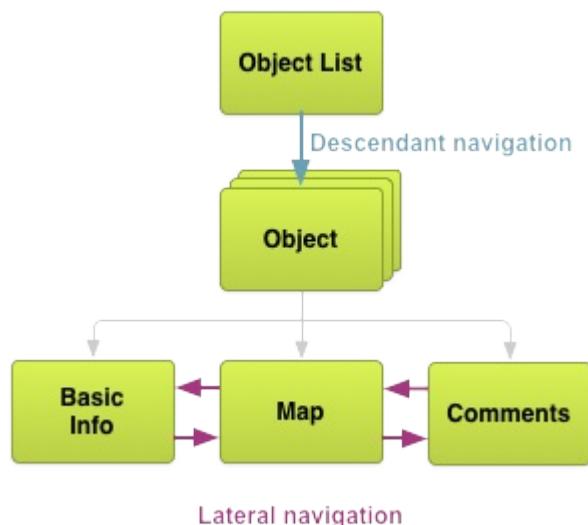


Figure 1. 向下和橫嚮導航

有兩種同級界面：容器關聯和區塊關聯界面。容器關聯（Collection-related）界面展示由父界面放入同個容器裏地那些條目。區塊關聯(Section-related) 界面展示父界面不同部分的信息，例如：一個部分可能展示某對象的文字信息，可是另一個部分則提供對象地理位置的地圖。一個父界面的區塊關聯界面數量通常較少。

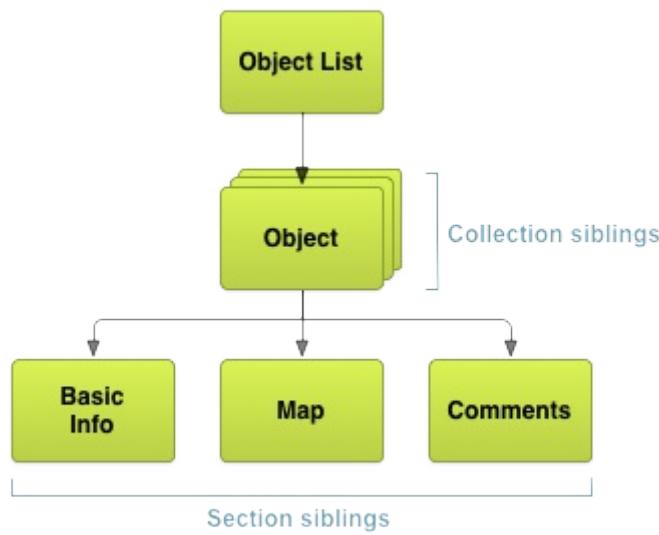


Figure 2. 容器關聯子界面和區塊關聯子界面。

向下和橫嚮導航可用List（列表），Tab（標籤）或者其他UI模式來實現。UI模式，與軟件設計模式很類似，是重復交互設計問題的一般化解決方案。下幾章，我們將探究一些常用的橫嚮導航模式。

## Button和簡單的控件

## Button設計

設計指南請閱讀 Android 設計文檔的[Button](#)指導

對於區塊關聯的界面，最直接和熟悉的導航界面就是提供可觸或鍵盤可得焦點的控件。例如，Button，固定大小的 List View 或 文本鏈接，雖然後者不是一個觸屏導航的理想 UI 元素。一旦點選了這些控件，子界面被打開，完全替代當前上下文環境（屏幕）。Button或其他簡單地控件很少被用來呈現容器中的項目。

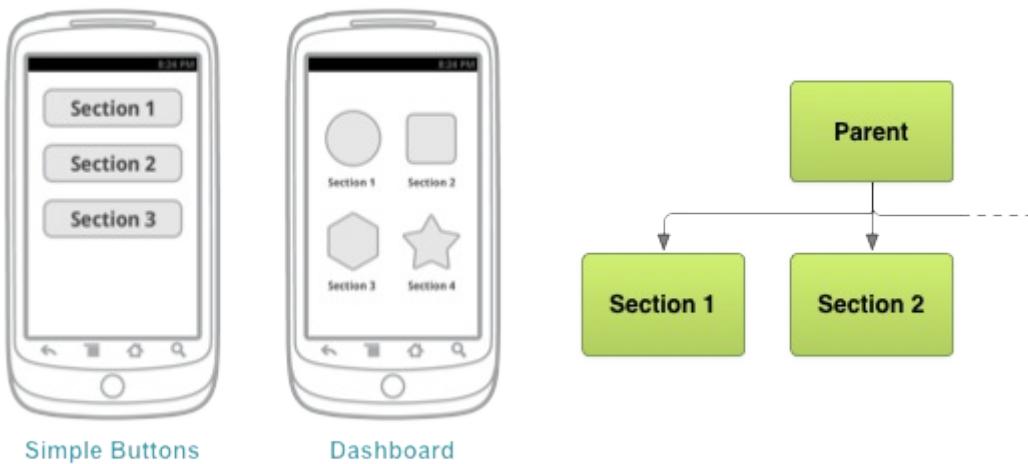


Figure 3. Button導航模式例子和對應界面圖。Dashboard 模式見下文。

Dashboard（操作面板）模式是一種一般以Button為主來獲取不同應用劃分模塊的模式。一個dashboard就是個大圖標Button表格，它表示了父界面絕大部分內容。這個表格通常是2、3行或列，取決於 App 的頂層劃分。此模式展示全部區塊的視覺效果非常豐富。巨大的觸摸控件也讓 UI 特別好使。當每個區塊都同等重要時，Dashboard模式最好用。然而，這個模式在大屏上效果不佳，他讓用戶直接獲取 App 內容時多走了一步彎路。

還有更多套用了各種其他 UI 模式來提升內容即得性和獨特的展示效果，但仍保持着直觀特點的高級 UI 模式。

## Lists, Grids, Carousels, and Stacks

### List 和 Grid List 設計

設計指南請閱讀 Android 設計文檔的[Lists](#)和[Grid Lists](#)指導。

對於容器關聯的界面，特別是文字信息，垂直滑動列表通常是最直接最熟悉的做法。對於視覺更豐富的內容（例如，圖片，視頻），可用垂直滑動的 Grid，水平滾動的 List（有時被叫做 Carousel），或 Stack（有時叫做卡片，Card）來代替。這些 UI 元素通常用在呈現容器內的條目，或大量子界面最好，而不是零星的毫無關聯的同級子界面。

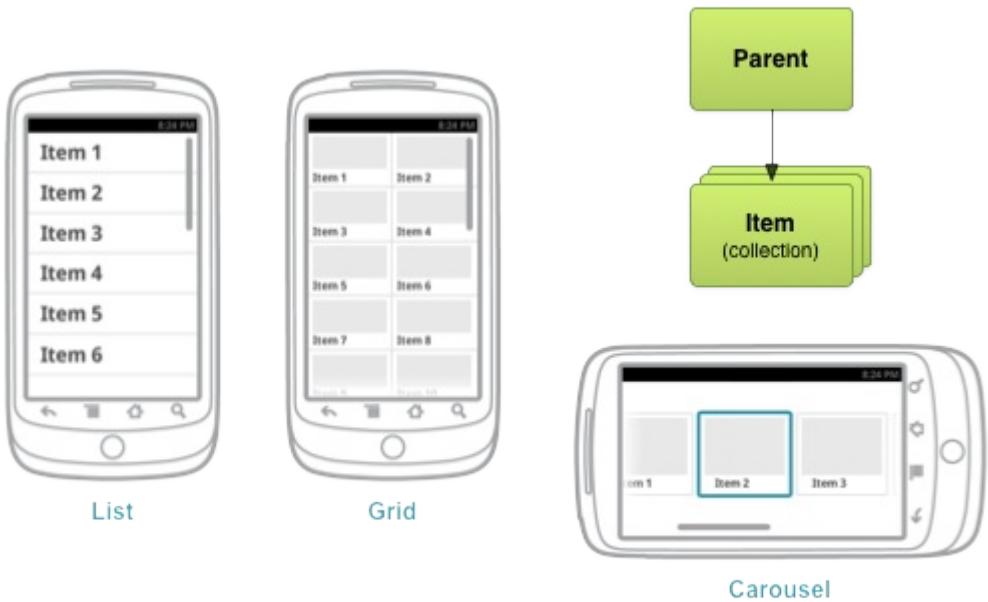


Figure 4. 控件例子和對應界面圖

這個模式還有些問題。深層列表導航常常叫 drill-down（鑽井）列表導航，它的list層層嵌套。這種導航笨拙低效。獲得某塊內容需要點擊多次，帶給用戶很差的體驗，特別是活躍用戶。

使用縱向list也可能帶來尷尬的用戶交互，並且如果list條目簡單地的拉伸話也可能用不好大屏空白。解決方法就是提供額外的信息，例如用文字彙總填充那些可用的水平空間。或者在左右添加個視窗。

## Tabs (標籤)

### Tab 設計

設計指南請閱讀 Android 設計文檔的[Tab指導](#)

Tab是非常流行的橫嚮導航。這個模式允許組合同級界面，就是說tab可嵌入原本可能成為另一個界面的子界面內容。Tab適合用在小量的區塊關聯界面。

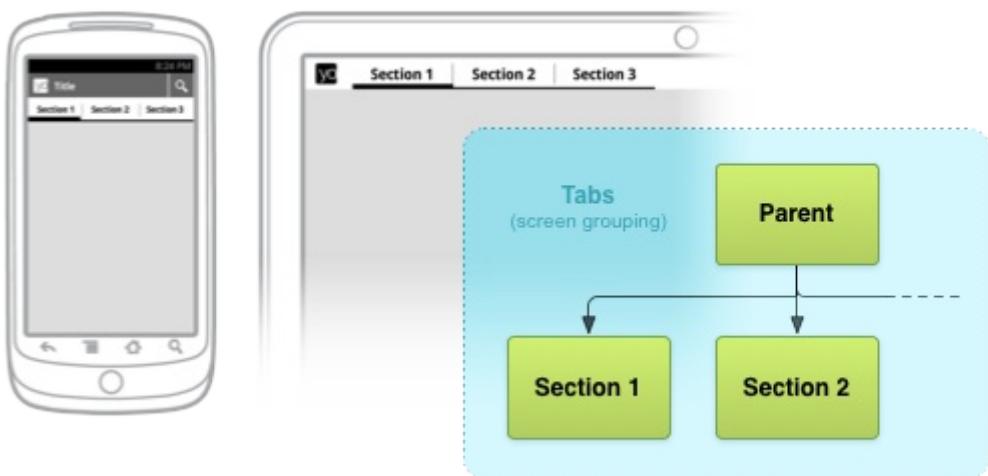


Figure 5. 手機和平板導航例子和對應界面圖

幾個使用Tab時的最佳做法。Tab在關聯界面種應該一直存在，只有指定內容區域發生改變，並且tab提示在任何時候都可用。此外，tab切換不能算作歷史。例如，如果用戶從 Tab A 切換到 Tab B，按 Back 按鈕（詳情看下節）不該重選 Tab A。Tab通常水平排布，可是有時其他tab展現形式，例如Action bar（詳見Android 設計的模式章節）的下拉菜單，也是可以的。最後，最重要的是，tab應該在界面頂端和內容對應。

tab導航相對於list和button導航，有很多即得的優點：

- 既然只有一個初始時既選的活動tab，用戶能立即從界面獲取tab的內容。
- 用戶可在相關界面內快速導航，不用重新訪問父界面。

注意：當切換Tab時，保證立即切換很重要。不要加載時彈個確認對話框來阻塞tab的訪問。

導致這個模式被批評常見的原因就是必須從展示內容的屏幕空間分一些給tab提示欄。但是結果還能接受，權衡一般都向使用此模式的方向傾斜。你可以隨意個性化你的tab提示欄，加點文字或圖標什麼的讓縱向空間合理利用。但是調整tab寬度時，請確保tab夠大到能讓人無誤點擊。

## 水平分頁 (Swipe View)

### Swipe View 設計

設計指南請閱讀 Android 設計文檔的[Swipe View](#)指導

另一種橫嚮導航的模式就是水平分頁，也叫做 Swipe View。這個模式在容器關聯的同級界面上最好用，例如類別列表（世界，金融，技術和健康新聞）。就像Tab，這個模式也允許組合界面，這樣父界面就能在佈局內嵌入子界面的內容。

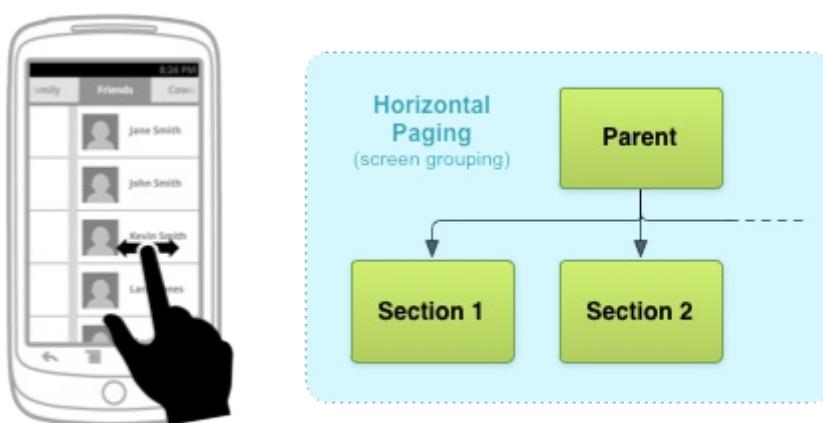


Figure 6. 水平分頁導航例子和對應界面圖

在水平分頁 UI 中，一次只展示一個子界面（這兒叫頁，page）。用戶能通過觸摸屏幕然後按想要訪問相鄰頁面的方向拖拽導航到同級界面。為補充這種手勢交互通常由另一種 UI 元素提示當前頁和可訪問頁。這樣能幫助用戶發覺內容並且也提供了更多的上下文環境信息給用戶。當為區塊關聯的同級模塊使用這種模式的水平導航時，這個做法很有必要。這些提示界面元素的例子包括點標（tick mark），滑動標註（scrolling label）和標籤（tab）：

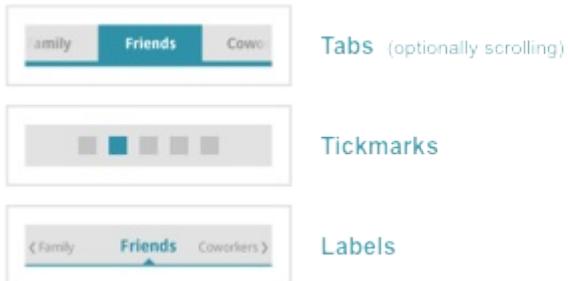


Figure 7. 搭配分頁的 UI 元件。

當子界麵包含水平平移視圖時（例如地圖）也最好避免使用這種模式，因為這些衝突的交互會威脅你界面的易用性。

此外，對於同級關聯界面，如果內容類型具有一定相似性而且同級界面數量較少時，水平分頁再適合不過了。就這一點，這個模式可以和tab一起用。tab放在內容上方來最大化界面直觀性。對於容器關聯界面，當界面間有天然的順序時，水平分頁是最符合直覺的，例如頁面代表連續的日歷日。對於無窮無盡的數據，特別是雙向都有內容數據，分頁機制效果非常棒。

下節課，我們討論在內容層級中允許用戶往上和回退到之前訪問界面的導航的機制。

[下節課：提供向上和時間導航](#)

# 提供向上導航與歷史導航

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/ancestral-temporal.html>

既然現在我們能進入應用界面某個層級，我需要提供一個方法來在層級裏向上導航到父親或祖先界面中。此外，我們應該保證通過 Back 按鈕來回退歷史導航記錄。

回退/向上導航設計

設計指南請閱讀 Android 設計文檔的[Navigation](#)模式指導

## 支持歷史導航：Back

歷史導航，或者說在歷史的界面間導航，在 Android 系統中由來已久。不論其他狀態如何，所有 Android 用戶都期望 Back 按鈕能帶他們回到之前的界面。歷史界面集全都以用戶的 Launcher 應用為基礎（電話的“Home”鍵）。也就是說，按下 Back 鍵足夠多次數後你應該回到 Launcher，之後 Back 鍵不做任何事情。



Figure 1. 從 Contacts (聯繫人) app 中進入電子郵件 app 然後按 Back 鍵的行為

應用自身通常不必考慮去管理 Back 按鈕。系統自己自動處理 task 和 back H1H2H3H4 stack (回退棧)，或者叫歷史界面列表。Back 按鈕默認反向訪問界面列表，然後當按鈕被按下時從列表中移除當前界面。

但是總是有一些你可能需要重寫 Back 行為的例子。比如，你屏幕包含一個嵌入的網頁瀏覽器，在這個瀏覽器中你的用戶可和頁面元件進行交互來在網頁間導航。你可能希望當用戶按下設備的 Back 鍵時觸發嵌入瀏覽器的默認 back 操作。當到達了瀏覽器內部歷史的起始點，你就應該遵從系統 Back 按鈕的默認行爲了。

## 提供向上導航：Up 和 Home

Android 3.0 之前，最常見的向上導航的形式以 Home 表示。大體上是以在設備 Menu 按鈕裏提供一個 Home 的可選項這樣的方法來實現，或者 Home 按鈕出現在屏幕的左上角作為 Action Bar (詳見Android 設計的[模式](#)章節)的一個組件。當選中 Home 後，用戶被帶到界面層級的頂層，通常被叫做應用的主界面。

提供對程序主界面的直接訪問能帶給用戶一種舒適感和安全感。無論位於應用程序何處，如果你在 App 中迷路了，你可以點選 Home 然後回到那熟悉的主界面。

Android 3.0 引入了 Up 記號，它被展示在了 Action Bar 上代替了上述的 Home 按鈕。點擊 Up，用戶將被帶入到結構中的父界面。這個導航操作通常就是進入前一個界面（就像之前 Back 按鈕討論中描述的一樣），但是並不是永遠都這樣。因此，開發者必須保證 Up 對於每個界面都會導航到某個既定的父親界面。

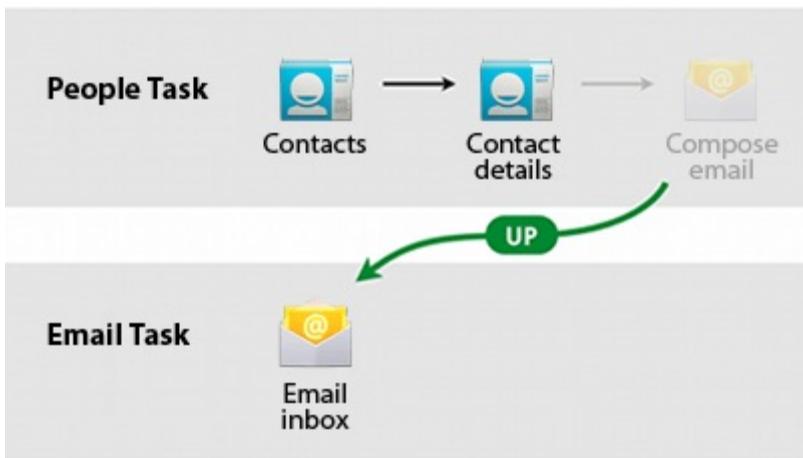


Figure 2. 從聯繫人 App 中進入電子郵件 App 然後按 Up 導航的行為

某些情況下，Up 適合執行某個行為而非導航到一個父親節點。以 Android 3.0 平板上的 Gmail 應用為例。當查看一封郵件的對話時把設備平放，對話列表和對話詳情將並排顯示。這是一種之前課程中的父、子界面組合。然而，當豎屏查看郵件對話時，只有對話詳情被顯示。Up 按鈕被用來使父視窗滑入屏幕顯示。當左側視窗可見時再按一次 Up 按鈕，單個對話便回到全屏的對話列表中。

**實現提醒：** 實現 Home 或 Up 導航的最佳做法就是保證清除back stack中的子界面。對於 Home，Home 界面是唯一留在back stack中的界面。對於 Up 導航，當前界面也應該從back stack中移除，除非 Back 在不同界面層級間導航。你可以將 `FLAG_ACTIVITY_CLEAR_TOP` 和 `FLAG_ACTIVITY_NEW_TASK` 這兩個 Intent 標記一起使用來實現它。

最後一節課中，我們應用現在為止所有課程中討論的概念來為我們新聞應用例子創建交互設計 Wireframe（線框圖）。

下節課：綜合：設計我們的樣例 App

# 綜合：設計我們的樣例 App

編寫:XizhiXu - 原文:<http://developer.android.com/training/design-navigation/wireframing.html>

現在我們對導航模式和界面組合技術有了深入的理解，是時候應用到我們的界面上了。讓我再看看我們第一節課上提到的新聞應用的界面完備圖：

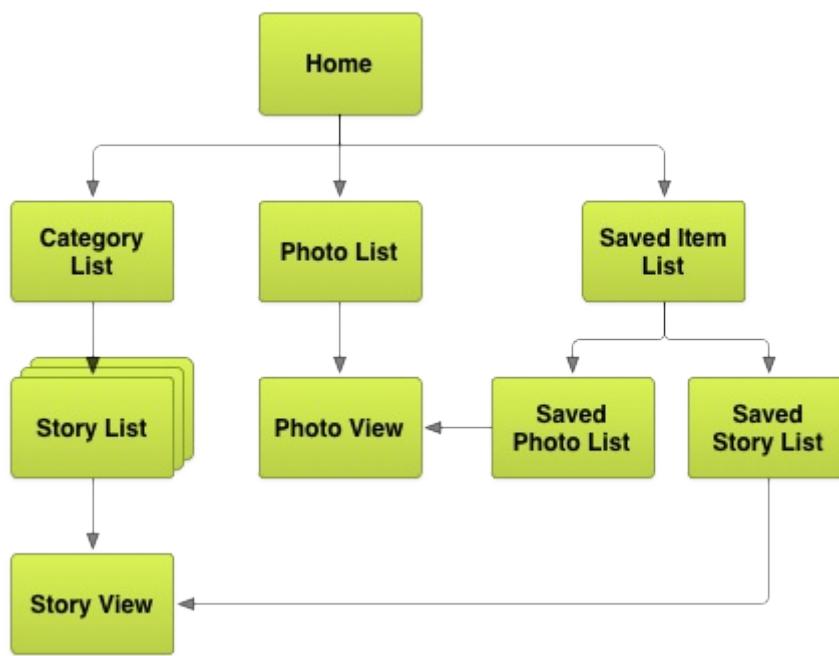


Figure 1. 新聞應用例子的界面完備集

我們下一步得去我們前幾節討論的導航模式選擇，然後應用到這個界面圖中。這樣就能最大化導航速度並且最少化獲取內容的點擊次數，但又能參考 Android 做法來保證界面的直觀性和一致性。此外，我們也需要根據我們不同目標設備的參數做出不同的決定。為方便，我們集中討論平板和手持設備。

## 選擇模式

首先，我們二級界面（新聞類別列表，圖片列表 和 保存列表）可用 Tab 組合在一起。注意到我們不必使用水平排列的 Tab；某些情況下下拉菜單可作爲合適的替代品，特別在手機這種窄屏設備上。在手機上，我們能用 Tab 把 圖片保存列表 和 新聞保存列表 組合到一起，或在平板上用多個縱向排列的內容視窗。

最後，讓我們看看如何展示新聞。第一個簡化不同新聞類別間導航的選項：使用水平分頁，然後再在滑動區域上添加一組標籤來提示當前可見和臨近的新聞類別。對於平板橫屏，我們可以進一步地展示能水平分頁的 新聞列表 界面作爲左邊的視窗，並且把 新聞詳情 View 界面作爲基礎內容視窗放在右邊。

下圖分別表示在手持設備和平板上應用了這些導航模式後的新界面圖。

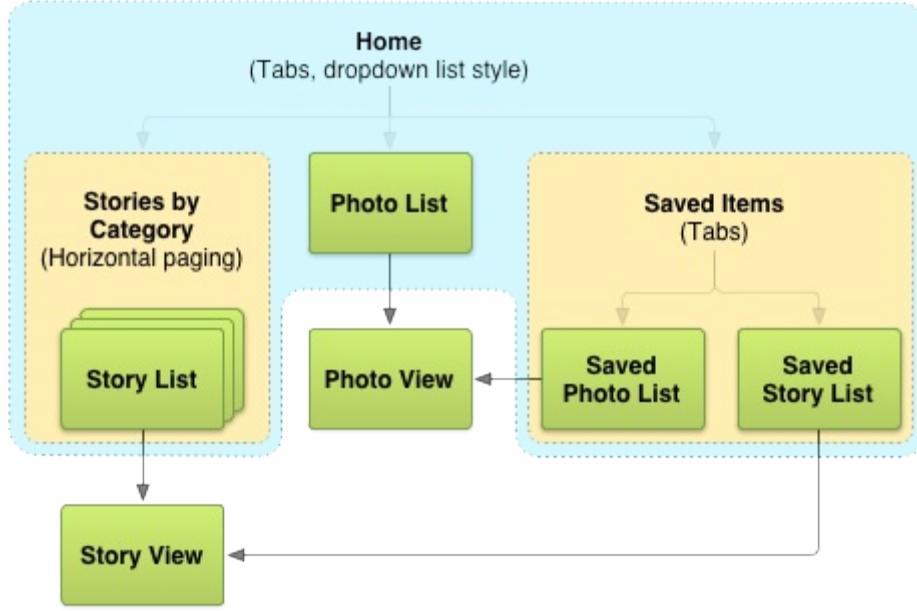


Figure 2. 手持設備上新聞應用例子的最終界面集

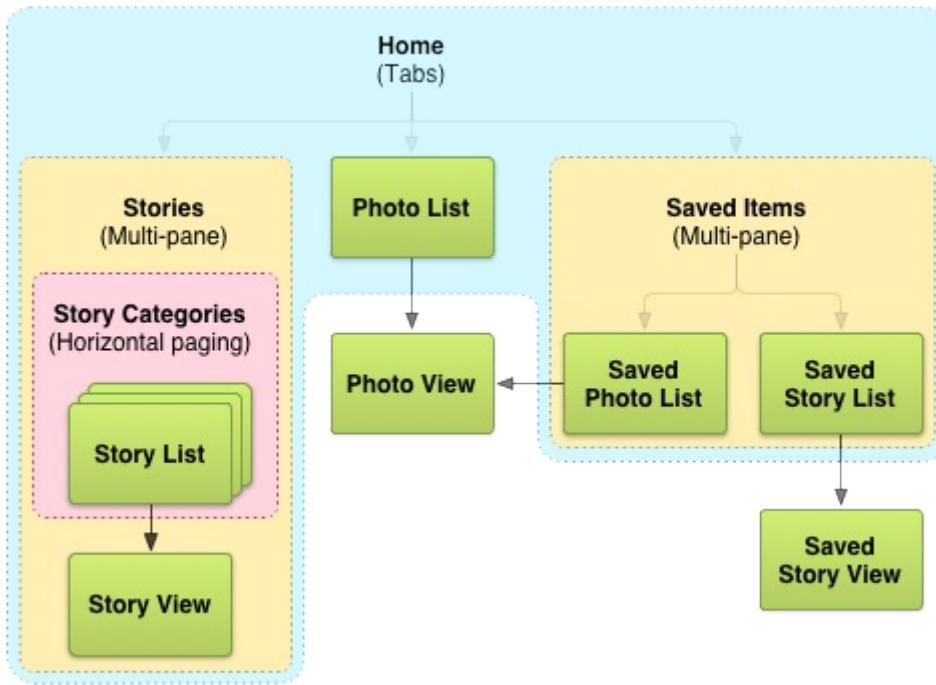


Figure 3. 平板上新聞應用例子的最終界面集，橫屏

至此，得好好考慮下界面圖的衍化了，以免我們選擇的模式實際上用不了（比如當你畫應用界面佈局的草圖時）。下面有個為平板衍化的界面圖樣例，它並排展示不同類別的 新聞列表，但是 新聞詳情View 保持獨立。

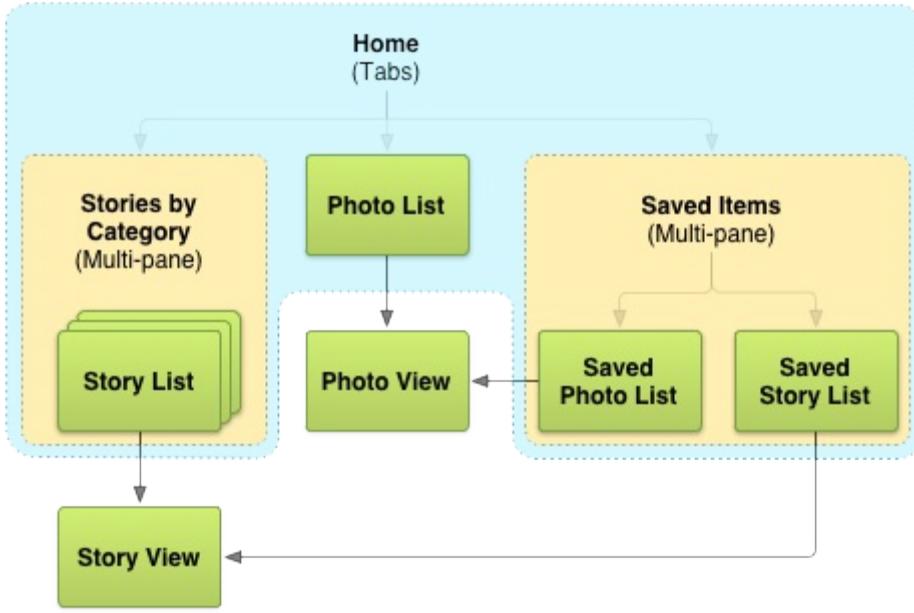


Figure 4. 平板上新聞應用例子的最終界面集，豎屏

## 畫草稿

**Wireframing**就是設計過程中你開始排布界面的那步。發揮你的創造性，想想怎麼排列這些 UI 元件來幫助你的用戶在你的 App 中導航。這時你要記住細枝末節是不重要的（別去想着做個實物）。

最簡單快速的起步方法就是用紙筆手畫你界面。一旦你開始畫，你會發現在你原本的界面圖或在你決定使用的模式中有很多實際的問題。某些情況下，模式理論上能很好的解決特定設計問題，但實際上他們可能失效並且給視覺交互添亂（例如，界面上出現了兩行 Tab）。如果那樣，探索下其他的導航模式，或在選擇的模式上做點變化，來讓你的草稿更優。

當你對初稿滿意後，繼續用一些軟件畫你的數字wireframe吧，例如：Adobe® Illustrator，Adobe® Fireworks，OmniGraffle 或者 向量圖工具。選擇畫圖工具時，考慮以下特性：

- 能畫體現交互的 wireframe 麼？像Adobe® Fireworks就能提供這個功能。
- 有界面“大師”功能（允許不同界面的視覺元素重用）？例如，Action Bar必須在你應用的每個界面都出現。
- 學習曲線怎樣？專業向量圖工具可能有個陡峭的學習曲線（越學越難），但有些功能小巧的 wireframing 設計工具可能更適合這個任務。

最後，XML 佈局編輯器，Android 開發工具包（ADT）裏面的一個 Eclipse 插件，經常被用來畫草圖原型。但是，你應當貫注於高質量的佈局而非細節視覺設計。

## 創建數字草圖

在紙上畫完草圖並且選擇好一款心儀的數字wireframing工具後，你可以創建一個數字wireframe作為你應用視覺設計的起點。下面就是一些我們新聞客戶端wireframe例子，他們和我們之前的界面圖一一對應。

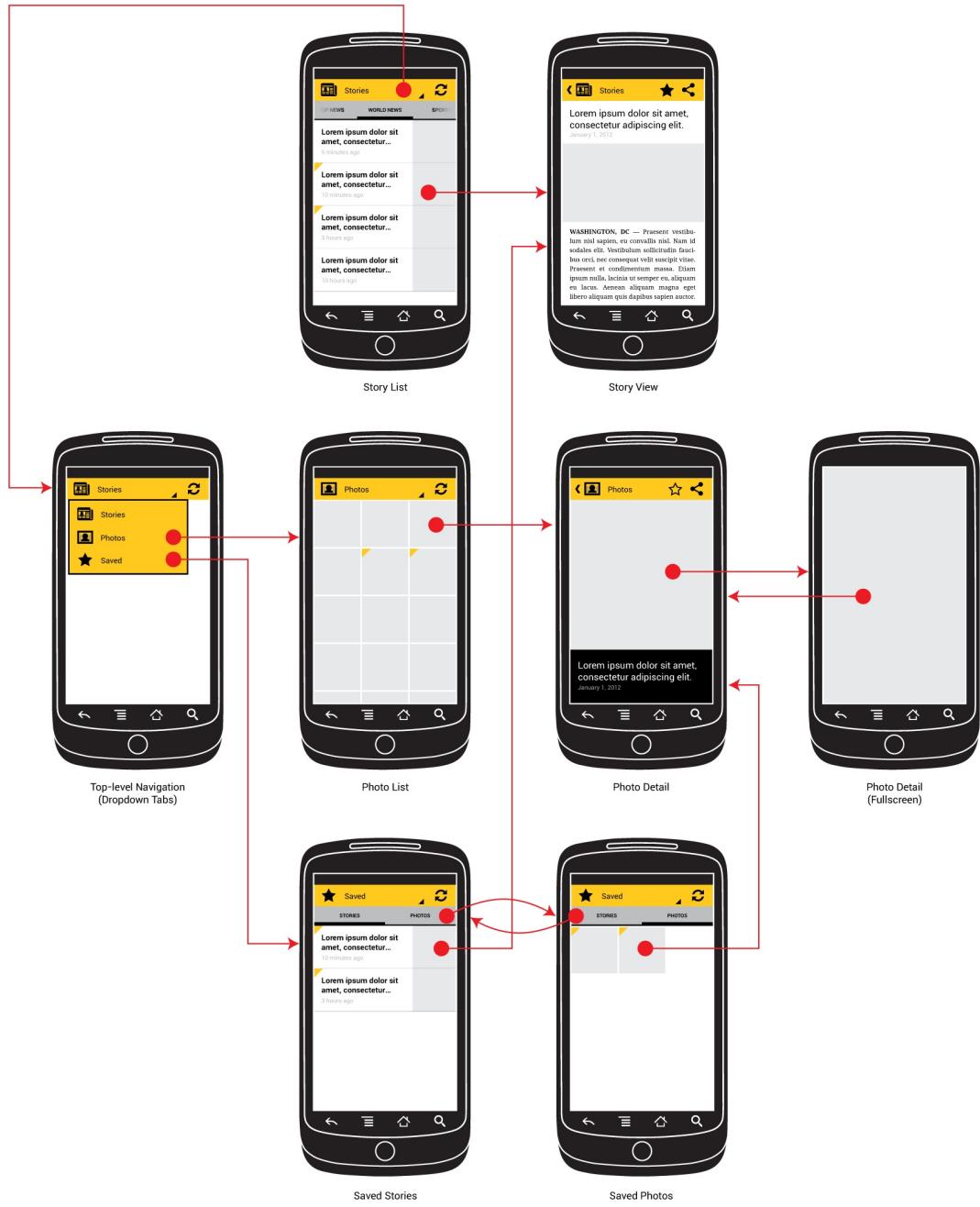


Figure 5. 新聞客戶端手機豎屏Wireframe樣例（下載 [SVG 圖](#)）



Figure 6. 新聞客戶端平板橫屏Wireframe樣例（下載 [SVG 圖](#)）

（下載表示設備的 Wireframe 的 SVG 圖）

## 下一步

現在你已經為你的應用設計出了高效直觀的 App 內部導航，你可用開始花時間來為單個界面改善 UI 了。例如，展示交互內容時，你可以選擇使用更花哨的控件來代替簡單的文本標籤，圖像和按鈕。你也可以開始定義你應用的視覺風格。在這過程中把你品牌的元素作為視覺語言融入其中吧。

最後，也適時實現你的設計吧，使用 Android SDK 為你的應用寫寫代碼。想開始？看看下面的這些資源吧：

- [開發者指導：UI](#) :學習如何用 Android SDK 實現你的 UI 設計。
- [Action Bar](#) :實現tab，向上導航，屏幕上動作，等等。
- [Fragment](#) :實現可重用，多視窗佈局
- [支持庫](#) :用 ViewPager 實現水平分頁（Swipe View）

# 實現高效的導航

編寫:Lin-H - 原文:<http://developer.android.com/training/implementing-navigation/index.html>

這節課將會演示如何實現在Designing Effective Navigation中所詳述的關鍵導航設計模式。

在閱讀這節課程內容之後，你會對如何使用tabs, swipe views, 和navigation drawer實現導航模式有一個深刻的理解。也會明白如何提供合適的向前向後導航(Up and Back navigation)。

Note:本節課中的幾個元素需要使用Support Library API。如果你之前沒有使用過Support Library，可以按照Support Library Setup文檔說明來使用。

## Sample Code

[EffectiveNavigation.zip](#)

## Lessons

- [使用Tabs創建Swipe View](#)

學習如何在action bar中實現tab，並提供橫向分頁(swipe views)在tab之間導航切換。

- [創建抽屜導航\(Navigation Drawer\)](#)

學習如何建立隱藏於屏幕邊上的界面，通過劃屏(swipe)或點擊action bar中的app圖標來顯示這個界面。

- [提供向上導航](#)

學習如何使用action bar中的app圖標實現向上導航

- [提供適當的向後導航](#)

學習如何正確處理特殊情況下的向後按鈕(Back button)，包括在通知或app widget中的深度鏈接，如何將activity插入後退棧(back stack)中。

- [實現Descendant Navigation](#)

學習更精細地導航進入你的應用信息層。

# 使用Tabs創建Swipe視圖

編寫:Lin-H - 原文:<http://developer.android.com/training/implementing-navigation/lateral.html>

Swipe View提供在同級屏幕中的橫嚮導航，例如通過橫向劃屏手勢切換的tab(一種稱作橫向分頁的模式)。這節課會教你如何使用swipe view創建一個tab layout實現在tab之間切換，或顯示一個標題條替代tab。

## Swipe View 設計

在實現這些功能之前，你要先明白在[Designing Effective Navigation, Swipe Views design guide](#)中的概念和建議

## 實現Swipe View

你可以使用[Support Library](#)中的[ViewPager](#)控件在你的app中創建swipe view。ViewPager是一個子視圖在layout上相互獨立的佈局控件(layout widget)。

使用[ViewPager](#)來設置你的layout，要添加一個 `<ViewPager>` 元素到你的XML layout中。例如，在你的swipe view中如果每一個頁面都會佔用整個layout，那麼你的layout應該是這樣:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

要插入每一個頁面的子視圖，你需要把這個layout與[PagerAdapter](#)掛鉤。有兩種adapter(適配器)你可以用:

### FragmentPagerAdapter

在同級屏幕(sibling screen)只有少量的幾個固定頁面時，使用這個最好。

### FragmentStatePagerAdapter

當根據對象集的數量來劃分頁面，即一開始頁面的數量未確定時，使用這個最好。當用戶切換到其他頁面時，fragment會被銷燬來降低內存消耗。

例如，這裏的代碼是當你使用[FragmentStatePagerAdapter](#)來在[Fragment](#)對象集閭中進行橫屏切換:

```
public class CollectionDemoActivity extends FragmentActivity {
    // 當被請求時，這個adapter會返回一個DemoObjectFragment,
    // 代表在對象集中的一個對象。
    DemoCollectionPagerAdapter mDemoCollectionPagerAdapter;
    ViewPager mViewPager;

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

        // ViewPager和他的adapter使用了support library
        // fragments,所以要用getSupportFragmentManager.
        mDemoCollectionPagerAdapter =
            new DemoCollectionPagerAdapter(
```

```

        getSupportFragmentManager());
        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mDemoCollectionPagerAdapter);
    }
}

// 因為這是一個對象集所以使用FragmentStatePagerAdapter,
// 而不是FragmentPagerAdapter.
public class DemoCollectionPagerAdapter extends FragmentStatePagerAdapter {
    public DemoCollectionPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int i) {
        Fragment fragment = new DemoObjectFragment();
        Bundle args = new Bundle();
        // 我們的對象只是一個整數 :-P
        args.putInt(DemoObjectFragment.ARG_OBJECT, i + 1);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public int getCount() {
        return 100;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return "OBJECT " + (position + 1);
    }
}

// 這個類的實例是一個代表了數據集中一個對象的fragment
public static class DemoObjectFragment extends Fragment {
    public static final String ARG_OBJECT = "object";

    @Override
    public View onCreateView(LayoutInflater inflater,
                           ViewGroup container, Bundle savedInstanceState) {
        // 最後兩個參數保證LayoutParams能被正確填充
        View rootView = inflater.inflate(
            R.layout.fragment_collection_object, container, false);
        Bundle args = getArguments();
        ((TextView) rootView.findViewById(android.R.id.text1)).setText(
            Integer.toString(args.getInt(ARG_OBJECT)));
        return rootView;
    }
}

```

這個例子只顯示了創建swipe view的必要代碼。下面一節向你說明如何通過添加tab使導航更方便在頁面間切換。

## 添加Tab到Action Bar

Action bar tab能給用戶提供更熟悉的界面來在app的同級屏幕中切換和分辨。

使用ActionBar來創建tab，你需要啓用NAVIGATION\_MODE\_TABS，然後創建幾個ActionBar.Tab的實例，並對每個實例實現ActionBar.TabListener接口。例如在你的activity的onCreate()方法中，你可以使用與下面相似的代碼：

```

@Override
public void onCreate(Bundle savedInstanceState) {
    final ActionBar actionBar = getActionBar();
    ...

    // 指定在action bar中顯示tab.
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

```

```

// 創建一個tab listener，在用戶切換tab時調用。
ActionBar.TabListener tabListener = new ActionBar.TabListener() {
    public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
        // 顯示指定的tab
    }

    public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {
        // 隱藏指定的tab
    }

    public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {
        // 可以忽略這個事件
    }
};

// 添加3個tab，並指定tab的文字和TabListener
for (int i = 0; i < 3; i++) {
    actionBar.addTab(
        actionBar.newTab()
            .setText("Tab " + (i + 1))
            .setTabListener(tabListener));
}
}

```

根據你如何創建你的內容來處理ActionBar.TabListener回調改變tab。但是如果你是像上面那樣，通過ViewPager對每個tab使用fragment，下面這節就會說明當用戶選擇一個tab時如何切換頁面，當用戶劃屏切換頁面時如何更新相應頁面的tab。

## 使用Swipe View切換Tab

當用戶選擇tab時，在ViewPager中切換頁面，需要實現ActionBar.TabListener來調用在ViewPager中的setCurrentItem()來選擇相應的頁面：

```

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

    // Create a tab listener that is called when the user changes tabs.
    ActionBar.TabListener tabListener = new ActionBar.TabListener() {
        public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
            // 當tab被選中時，切換到ViewPager中相應的頁面。
            mViewPager.setCurrentItem(tab.getPosition());
        }
        ...
    };
}

```

同樣的，當用戶通過觸屏手勢(touch gesture)切換頁面時，你也應該選擇相應的tab。你可以通過實現ViewPager.OnPageChangeListener接口來設置這個操作，當頁面變化時當前的tab也相應變化。例如：

```

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

    mViewPager = (ViewPager) findViewById(R.id.pager);
    mViewPager.setOnPageChangeListener(
        new ViewPager.SimpleOnPageChangeListener() {
            @Override
            public void onPageSelected(int position) {
                // 當劃屏切換頁面時，選擇相應的tab。
                getActionBar().setSelectedNavigationItem(position);
            }
        });
}

```

```
    });
    ...
}
```

## 使用標題欄替代Tab

如果你不想使用action bar tab，而想使用scrollable tabs來提供一個更簡短的可視化配置，你可以在swipe view中使用PagerTitleStrip。

下面是一個內容為ViewPager，有一個PagerTitleStrip頂端對齊的activity的layout XML文件示例。單個頁面(adapter提供)佔據ViewPager中的剩餘空間。

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTitleStrip
        android:id="@+id/pager_title_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#33b5e5"
        android:textColor="#fff"
        android:paddingTop="4dp"
        android:paddingBottom="4dp" />

</android.support.v4.view.ViewPager>
```

# 創建抽屜式導航(navigation drawer)

編寫:Lin-H - 原文: <http://developer.android.com/training/implementing-navigation/nav-drawer.html>

Navigation drawer是一個在屏幕左側邊緣顯示導航選項的面板。大部分時候是隱藏的，當用戶從屏幕左側劃屏，或在top level模式的app中點擊action bar中的app圖標時，纔會顯示。

這節課敘述如何使用Support Library中的DrawerLayout API，來實現navigation drawer。

Navigation Drawer 設計：在你決定在你的app中使用Navigation Drawer之前，你應該先理解在Navigation Drawer design guide中定義的使用情況和設計準則。

## 創建一個Drawer Layout

要添加一個navigation drawer，在你的用戶界面layout中聲明一個用作root view(根視圖)的DrawerLayout對象。在DrawerLayout中為屏幕添加一個包含主要內容的view(當drawer隱藏時的主layout)，和其他一些包含navigation drawer內容的view。

例如，下面的layout使用了有兩個子視圖(child view)的DrawerLayout:一個FrameLayout用來包含主要內容(在運行時被Fragment填入)，和一個navigation drawer使用的ListView。

```
<android.support.v4.widget.DrawerLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/drawer_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <!-- 包含主要內容的 view -->  
    <FrameLayout  
        android:id="@+id/content_frame"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent" />  
    <!-- navigation drawer(抽屜式導航) -->  
    <ListView android:id="@+id/left_drawer"  
        android:layout_width="240dp"  
        android:layout_height="match_parent"  
        android:layout_gravity="start"  
        android:choiceMode="singleChoice"  
        android:divider="@android:color/transparent"  
        android:dividerHeight="0dp"  
        android:background="#111"/>  
</android.support.v4.widget.DrawerLayout>
```

這個layout展示了一些layout的重要特點:

- 主內容view(上面的FrameLayout)，在DrawerLayout中必須是第一個子視圖，因為XML的順序代表著Z軸(垂直於手機屏幕)的順序，並且drawer必須在內容的前端。
- 主內容view被設置為匹配父視圖的寬和高，因為當navigation drawer隱藏時，主內容表示整個UI部分。
- drawer視圖(ListView)必須使用 android:layout\_gravity 屬性指定它的horizontal gravity。為了支持從右邊閱讀的語言(right-to-left(RTL) language)，指定它的值為 "start" 而不是 "left" (當layout是RTL時drawer在右邊顯示)。
- drawer視圖以 dp 為單位指定它的寬和高來匹配父視圖。drawer的寬度不能大於320dp，這樣用戶總能看到部分主內容。

# 初始化Drawer List

在你的activity中，首先要做的事就是要初始化drawer的item列表。這要根據你的app內容來處理，但是一個navigation drawer通常由一個ListView組成，所以列表應該通過一個Adapter(例如ArrayAdapter或SimpleCursorAdapter)填入。

例如，如何使用一個字符串數組(string array)來初始化導航列表(navigation list):

```
public class MainActivity extends Activity {  
    private String[] mPlanetTitles;  
    private DrawerLayout mDrawerLayout;  
    private ListView mDrawerList;  
    ...  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mPlanetTitles = getResources().getStringArray(R.array.planets_array);  
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);  
        mDrawerList = (ListView) findViewById(R.id.left_drawer);  
  
        // 為list view設置adapter  
        mDrawerList.setAdapter(new ArrayAdapter<String>(this,  
            R.layout.drawer_list_item, mPlanetTitles));  
        // 為list設置click listener  
        mDrawerList.setOnItemClickListener(new DrawerItemClickListener());  
  
        ...  
    }  
}
```

這段代碼也調用了setOnItemClickListener()來接收navigation drawer列表的點擊事件。下一節會說明如何實現這個接口，並且當用戶選擇一個item時如何改變內容視圖(content view)。

## 處理導航的點擊事件

當用戶選擇drawer列表中的item，系統會調用在setOnItemClickListener()中所設置的OnItemClickListener的onItemClick()。

在onItemClick()方法中做什麼，取決於你如何實現你的app結構(app structure)。在下面的例子中，每選擇一個列表中的item，就插入一個不同的Fragment到主內容視圖中(FrameLayout元素通過R.id.content\_frame ID辨識):

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {  
    @Override  
    public void onItemClick(AdapterView parent, View view, int position, long id) {  
        selectItem(position);  
    }  
}  
  
/** 在主內容視圖中交換fragment */  
private void selectItem(int position) {  
    // 創建一個新的fragment並且根據行星的位置來顯示  
    Fragment fragment = new PlanetFragment();  
    Bundle args = new Bundle();  
    args.putInt(PlanetFragment.ARG_PLANET_NUMBER, position);  
    fragment.setArguments(args);  
  
    // 通過替換已存在的fragment來插入新的fragment  
    FragmentManager fragmentManager = getFragmentManager();  
    fragmentManager.beginTransaction()
```

```

        .replace(R.id.content_frame, fragment)
        .commit();

    // 高亮被選擇的item，更新標題，並關閉drawer
    mDrawerList.setItemChecked(position, true);
    setTitle(mPlanetTitles[position]);
    mDrawerLayout.closeDrawer(mDrawerList);
}

@Override
public void setTitle(CharSequence title) {
    mTitle = title;
    getActionBar().setTitle(mTitle);
}

```

## 監聽打開和關閉事件

要監聽drawer的打開和關閉事件，在你的`DrawerLayout`中調用`setDrawerListener()`，並傳入一個`DrawerLayout.DrawerListener`的實現。這個接口提供drawer事件的回調例如`onDrawerOpened()`和`onDrawerClosed()`。

但是，如果你的activity包含有action bar可以不用實現`DrawerLayout.DrawerListener`，你可以繼承`ActionBarDrawerToggle`來替代。`ActionBarDrawerToggle`實現了`DrawerLayout.DrawerListener`，所以你仍然可以重寫這些回調。這麼做也能使action bar圖標和navigation drawer的交互操作變得更容易(在下節詳述)。

如`Navigation Drawer design guide`中所述，當drawer可見時，你應該修改action bar的內容，比如改變標題和移除與主文字內容相關的action item。下面的代碼向你說明如何通過`ActionBarDrawerToggle`類的實例，重寫`DrawerLayout.DrawerListener`的回調方法來實現這個目的：

```

public class MainActivity extends Activity {
    private DrawerLayout mDrawerLayout;
    private ActionBarDrawerToggle mDrawerToggle;
    private CharSequence mDrawerTitle;
    private CharSequence mTitle;

    ...

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

        mTitle = mDrawerTitle = getTitle();
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
            R.drawable.ic_drawer, R.string.drawer_open, R.string.drawer_close) {

            /** 當drawer處於完全關閉的狀態時調用 */
            public void onDrawerClosed(View view) {
                super.onDrawerClosed(view);
                getActionBar().setTitle(mTitle);
                invalidateOptionsMenu(); // 創建對onPrepareOptionsMenu()的調用
            }

            /** 當drawer處於完全打開的狀態時調用 */
            public void onDrawerOpened(View drawerView) {
                super.onDrawerOpened(drawerView);
                getActionBar().setTitle(mDrawerTitle);
                invalidateOptionsMenu(); // 創建對onPrepareOptionsMenu()的調用
            }
        };

        // 設置drawer觸發器為DrawerListener
        mDrawerLayout.setDrawerListener(mDrawerToggle);
    }
}

```

```

/* 當invalidateOptionsMenu()調用時調用 */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // 如果nav drawer是打開的，隱藏與內容視圖相關聯的action items
    boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerList);
    menu.findItem(R.id.action_websearch).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}
}

```

下一節會描述ActionBarDrawerToggle的構造參數，和處理與action bar圖標交互所需的其他步驟。

## 使用App圖標來打開和關閉

用戶可以在屏幕左側使用劃屏手勢來打開和關閉navigation drawer，但是如果你使用action bar，你也應該允許用戶通過點擊app圖標來打開或關閉。並且app圖標也應該使用一個特殊的圖標來指明navigation drawer的存在。你可以通過使用上一節所說的ActionBarDrawerToggle來實現所有的這些操作。

要使ActionBarDrawerToggle起作用，通過它的構造函數創建一個實例，需要用到以下參數：

- Activity用來容納drawer。
- DrawerLayout。
- 一個drawable資源用作drawer指示器。標準的navigation drawer可以在[Download the Action Bar Icon Pack](#)獲得。
- 一個字符串資源描述"打開抽屜"操作(便於訪問)
- 一個字符串資源描述"關閉抽屜"操作(便於訪問)

那麼，不論你是否創建了用作drawer監聽器的ActionBarDrawerToggle的子類，你都需要在activity生命週期中的某些地方根據你的ActionBarDrawerToggle來調用。

```

public class MainActivity extends Activity {
    private DrawerLayout mDrawerLayout;
    private ActionBarDrawerToggle mDrawerToggle;

    ...

    public void onCreate(Bundle savedInstanceState) {
        ...

        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerToggle = new ActionBarDrawerToggle(
            this,                               /* 承載 Activity */
            mDrawerLayout,                      /* DrawerLayout 對象 */
            R.drawable.ic_drawer,               /* nav drawer 圖標用來替換'Up'符號 */
            R.string.drawer_open,               /* "打開 drawer" 描述 */
            R.string.drawer_close              /* "關閉 drawer" 描述 */
        ) {

            /** 當drawer處於完全關閉的狀態時調用 */
            public void onDrawerClosed(View view) {
                super.onDrawerClosed(view);
                getActionBar().setTitle(mTitle);
            }

            /** 當drawer處於完全打開的狀態時調用 */
            public void onDrawerOpened(View drawerView) {
                super.onDrawerOpened(drawerView);
                getActionBar().setTitle(mDrawerTitle);
            }
        }
    }
}

```

```
};

// 設置drawer觸發器為DrawerListener
mDrawerLayout.setDrawerListener(mDrawerToggle);

getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 在onRestoreInstanceState發生後，同步觸發器狀態。
    mDrawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mDrawerToggle.onConfigurationChanged(newConfig);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // 將事件傳遞給ActionBarDrawerToggle，如果返回true：表示app 圖標點擊事件已經被處理
    if (mDrawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
    // 處理你的其他action bar items...

    return super.onOptionsItemSelected(item);
}

...
}
```

一個完整的navigation drawer例子,可以在原文頁面頂端的sample下載

# 提供向上的導航

編寫:Lin-H - 原文:<http://developer.android.com/training/implementing-navigation/ancestral.html>

所有不是從主屏幕("home"屏幕)進入app的，都應該給用戶提供一種方法，通過點擊action bar中的Up按鈕。可以回到app的結構層次中邏輯父屏幕。本課程向你說明如何正確地實現這一操作。

Up Navigation 設計

Designing Effective Navigation和the [Navigation design guide](#)中描述了向上導航的概念和設計準則。

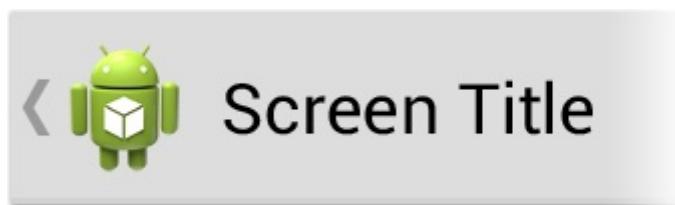


Figure 1. action bar中的Up按鈕。

## 指定父Activity

要實現向上導航，第一步就是為每一個activity聲明合適的父activity。這麼做可以使系統簡化導航模式，例如向上導航，因為系統可以從manifest文件中判斷它的邏輯父(logical parent)activity。

從Android 4.1 (API level 16)開始，你可以通過指定 `<activity>` 元素中的`android:parentActivityName`屬性來聲明每一個activity的邏輯父activity。

如果你的app需要支持Android 4.0以下版本，在你的app中包含[Support Library](#)並添加 `<meta-data>` 元素到 `<activity>` 中。然後指定父activity的值為 `android.support.PARENT_ACTIVITY`，並匹配`android:parentActivityName`的值。

例如:

```
<application ... >
    ...
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>

    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

在父activity這樣聲明後，你可以使用[NavUtils API](#)進行向上導航操作，就像下面這節。

# 添加向上操作(Up Action)

要使用action bar的app圖標來完成向上導航，需要調用`setDisplayHomeAsUpEnabled()`:

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
    getActionBar().setDisplayHomeAsUpEnabled(true);  
}
```

這樣，在app旁添加了一個左向符號，並用作操作按鈕。當用戶點擊它時，你的activity會接收一個對`onOptionsItemSelected()`的調用。操作的ID是`android.R.id.home`。

## 向上導航至父activity

要在用戶點擊app圖標時向上導航，你可以使用`NavUtils`類中的靜態方法`navigateUpFromSameTask()`。當你調用這一個方法時，系統會結束當前的activity並啓動(或恢復)相應的父activity。如果目標activity在任務的後退棧中(back stack)，則目標activity會像`FLAG_ACTIVITY_CLEAR_TOP`定義的那樣，提到棧頂。提到棧頂的方式取決於父activity是否處理了對`onNewIntent()`的調用。

例如:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // 對action bar的Up/Home按鈕做出反應  
        case android.R.id.home:  
            NavUtils.navigateUpFromSameTask(this);  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

但是，只能是當你的app擁有當前任務(current task)(用戶從你的app中發起這一任務)時`navigateUpFromSameTask()`纔有用。如果你的activity是從別的app的任務中啓動的話，向上導航操作就應該創建一個屬於你的app的新任務，並需要你創建一個新的後退棧。

## 用新的後退棧來向上導航

如果你的activity提供了任何允許被別的app啓動的intent filters，那麼你應該實現`onOptionsItemSelected()`回調，在用戶從別的app任務進入你的activity後，點擊Up按鈕，在向上導航之前你的app用相應的後退棧開啓一個新的任務。

在這麼做之前，你可以先調用`shouldUpRecreateTask()`來檢查當前的activity實例是否在另一個不同的app任務中。如果返回true，就使用`TaskStackBuilder`創建一個新任務。或者，你可以向上面那樣使用`navigateUpFromSameTask()`方法。

例如:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // 對action bar的Up/Home按鈕做出反應  
        case android.R.id.home:
```

```
Intent upIntent = NavUtils.getParentActivityIntent(this);
if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
    // 這個activity不是這個app任務的一部分，所以當向上導航時創建
    // 用合成後退棧(synthesized back stack)創建一個新任務。
    TaskStackBuilder.create(this)
        // 添加這個activity的所有父activity到後退棧中
        .addNextIntentWithParentStack(upIntent)
        // 向上導航到最近的一個父activity
        .startActivities();
} else {
    // 這個activity是這個app任務的一部分，所以
    // 向上導航至邏輯父activity。
    NavUtils.navigateUpTo(this, upIntent);
}
return true;
}
return super.onOptionsItemSelected(item);
}
```

Note:為了能使`addNextIntentWithParentStack()`方法起作用，你必須像上面說的那樣，在你的manifest文件中使用`android:parentActivityName`(和相應的 `<meta-data>` 元素)屬性聲明所有的activity的邏輯父activity。

# 提供向後的導航

編寫:Lin-H - 原文:<http://developer.android.com/training/implementing-navigation/temporal.html>

向後導航(Back navigation)是用戶根據屏幕歷史記錄返回之前所查看的界面。所有Android設備都可以為這種導航提供後退按鈕，所以你的app不需要在UI中添加後退按鈕。

在幾乎所有情況下，當用戶在應用中進行導航時，系統會保存activity的後退棧。這樣當用戶點擊後退按鈕時，系統可以正確地向後導航。但是，有少數幾種情況需要手動指定app的後退操作，來提供更好的用戶體驗。

## Back Navigation 設計

在繼續閱讀篇文章之前，你應該先在Navigation design guide中對後退導航的概念和設計準則有個瞭解。

手動指定後退操作需要的導航模式：

- 當用戶從notification(通知)，app widget，navigation drawer直接進入深層次activity。
- 用戶在fragment之間切換的某些情況。
- 當用戶在WebView中對網頁進行導航。

下面說明如何在這幾種情況下實現恰當的向後導航。

## 為深度鏈接合併新的後退棧

一般而言，當用戶從一個activity導航到下一個時，系統會遞增地創建後退棧。但是當用戶從一個在自己的任務中啓動activity的深度鏈接進入app，你就有必要去同步新的後退棧，因為新的activity是運行在一個沒有任何後退棧的任務中。

例如，當用戶從通知進入你的app中的深層activity時，你應該添加別的activity到你的任務的後退棧中，這樣當點擊後退(Back)時向上導航，而不是退出app。這個模式在Navigation design guide中有更詳細的介紹。

## 在manifest中指定父activity

從Android 4.1 (API level 16)開始，你可以通過指定`<activity>`元素中的`android:parentActivityName`屬性來聲明每一個activity的邏輯父activity。這樣系統可以使導航模式變得更容易，因為系統可以根據這些信息判斷邏輯Back Up navigation的路徑。

如果你的app需要支持Android 4.0以下版本，在你的app中包含Support Library並添加`<meta-data>`元素到`<activity>`中。然後指定父activity的值為`android.support.PARENT_ACTIVITY`，並匹配`android:parentActivityName`的值。

例如：

```
<application ... >
  ...
  <activity
    android:name="com.example.myfirstapp.MainActivity" ...>
    ...
  </activity>
```

```
<activity
    android:name="com.example.myfirstapp.DisplayMessageActivity"
    android:label="@string/title_activity_display_message"
    android:parentActivityName="com.example.myfirstapp.MainActivity" >
    <!-- 4.1 以下的版本需要使用meta-data元素 -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.myfirstapp.MainActivity" />
</activity>
</application>
```

當父activity用這種方式聲明，你就可以使用NavUtils API，通過確定每個activity相應的父activity來同步新的後退棧。

## 在啓動activity時創建後退棧

在發生用戶進入app的事件時，開始添加activity到後退棧中。就是說，使用TaskStackBuilder API定義每個被放到新後退棧的activity，不使用startActivity()。然後調用startActivities()來啓動目標activity，或調用getPendingIntent()來創建相應的PendingIntent。

例如，當用戶從通知進入你的app中的深層activity時，你可以使用這段代碼來創建一個啓動activity並把新後退棧插入目標任務的PendingIntent。

```
// 當用戶選擇通知時，啓動activity的intent
Intent detailsIntent = new Intent(this, DetailsActivity.class);

// 使用TaskStackBuilder創建後退棧，並獲取PendingIntent
PendingIntent pendingIntent =
    TaskStackBuilder.create(this)
        // 添加所有DetailsActivity的父activity到棧中，
        // 然後再添加DetailsActivity自己
        .addNextIntentWithParentStack(upIntent)
        .getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(pendingIntent);
...
```

產生的PendingIntent不僅指定了啓動哪個activity(被 detailsIntent 所定義)還指定了要插入任務(所有被 detailsIntent 定義的 DetailsActivity )的後退棧。所以當 DetailsActivity 啓動時，點擊Back向後導航至每一個 DetailsActivity 類的父activity。

Note:為了使addNextIntentWithParentStack()方法起作用，像上面所說那樣，你必須在你的manifest文件中使用android:parentActivityName(和相應的元素 <meta-data> )屬性聲明每個activity的邏輯父activity。

## 為Fragment實現向後導航

當在app中使用fragment時，個別的FragmentTransaction對象可以代表要加入後退棧中變化的內容。例如，如果你要在手機上通過交換fragment實現一個master/detail flow(主/詳細流程)，你就要保證點擊Back按鈕可以從detail screen返回到master screen。要這麼做，你可以在提交事務(transaction)之前調用addToBackStack():

```
// 使用framework FragmentManager
// 或support package FragmentManager (getSupportFragmentManager).
getSupportFragmentManager().beginTransaction()
    .add(detailFragment, "detail")
    // 提交這一任務到後退棧中
    .addToBackStack()
    .commit();
```

當後退棧中有`FragmentTransaction`對象並且用戶點擊Back按鈕時,`FragmentManager`會從後退棧中彈出最近的事務，然後執行反向操作(例如如果事務添加了一個fragment，那麼就刪除一個fragment)。

Note:當事務用作水平導航(例如切換tab)或者修改內容外觀(例如在調整filter時)時，不要將這個事務添加到後退棧中。更多關於向後導航的恰當時機的信息，詳見[Navigation design guide](#)。

如果你的應用更新了別的UI元素來反應當前的fragment狀態，例如action bar，記得當你提交事務時更新UI。除了在提交事務的時候，在後退棧發生變化時也要更新你的UI。你可以設置一個`FragmentManager.OnBackStackChangedListener`來監聽`FragmentTransaction`什麼時候復原：

```
getSupportFragmentManager().addOnBackStackChangedListener(  
    new FragmentManager.OnBackStackChangedListener() {  
        public void onBackStackChanged() {  
            // 在這裏更新你的UI  
        }  
    });
```

## 為WebView實現向後導航

如果你的應用的一部分包含在[WebView](#)中，可以通過瀏覽器歷史使用Back。要這麼做，如果[WebView](#)有歷史記錄，你可以重寫`onBackPressed()`並代理給[WebView](#):

```
@Override  
public void onBackPressed() {  
    if (mWebView.canGoBack()) {  
        mWebView.goBack();  
        return;  
    }  
  
    // 否則遵從系統的默認操作。  
    super.onBackPressed();  
}
```

要注意當使用這一機制時，高動態化的頁面會產生大量歷史。會生成大量歷史的頁面，例如經常改變文件散列(document hash)的頁面，當要退出你的[activity](#)時，這會使你的用戶感到繁瑣。

更多關於使用[WebView](#)的信息，詳見[Building Web Apps in WebView](#)。

# 實現向下的導航

編寫:Lin-H - 原文:<http://developer.android.com/training/implementing-navigation/descendant.html>

Descendant Navigation是用來向下導航至應用的信息層次。在[Designing Effective Navigation](#)和[Android Design: Application Structure](#)中說明。

Descendant navigation通常使用[Intent](#)和[startActivity\(\)](#)實現，或使用[FragmentTransaction](#)對象添加fragment到一個[activity](#)中。這節課程涵蓋了在實現Descendant navigation時遇到的其他有趣的情況。

## 在手機和平板(Tablet)上實現Master/Detail Flow

在master/detail導航流程(navigation flow)中，master screen(主屏幕)包含一個集閭中item的列表，detail screen(詳細屏幕)顯示集閭中特定item的詳細信息。實現從master screen到detail screen的導航是Descendant Navigation的一種形式。

手機觸摸屏非常適合一次顯示一種屏幕(master screen或detail screen)；這一想法在[Planning for Multiple Touchscreen Sizes](#)中進一步說明。在這種情況下，一般使用[Intent](#)啓動detail screen來實現activity Descendant navigation。另一方面，平板的顯示，特別是用橫屏來瀏覽時，最適合一次顯示多個內容窗格，master內容在左邊，detail在右邊。在這裏一般就使用[FragmentTransaction](#)實現descendant navigation。[FragmentTransaction](#)用來添加、刪除或用新內容替換detail窗格(pane)。

實現這一模式的基礎內容在Designing for Multiple Screens的[Implementing Adaptive UI Flows](#)課程中說明。課程中說明了如何在手機上使用兩個[activity](#)，在平板上使用一個[activity](#)來實現master/detail flow。

## 導航至外部Activities

有很多情況，是從別的應用下降(descend)至你的應用信息層次(application's information hierarchy)再到[activity](#)。例如，當正在瀏覽手機通訊錄中聯繫信息的details screen，子屏幕詳細顯示由社交網絡聯繫提供的最近文章，子屏幕可就可以屬於一個社交網絡應用。

當啓動另一個應用的[activity](#)來允許用戶說話，發郵件或選擇一個照片附件，如果用戶是從啓動器(設備的home屏幕)重啓你的應用，你一般不會希望用戶返回到別的[activity](#)。如果點擊你的應用圖標又回到“發郵件”的屏幕，這會使用戶感到很迷惑。

為防止這種情況的發生，只需要添加[FLAG\\_ACTIVITY\\_CLEAR\\_WHEN\\_TASK\\_RESET](#)標記到用來啓動外部[activity](#)的intent中，就像：

```
Intent externalActivityIntent = new Intent(Intent.ACTION_PICK);
externalActivityIntent.setType("image/*");
externalActivityIntent.addFlags(
    Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
startActivity(externalActivityIntent);
```

# 通知提示用戶

---

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user/index.html>

- Notification是一種在你APP常規UI外展示、用來指示某個事件發生的用戶交互元素。用戶可以在使用其它apps時查看notification，並在方便的時候做出迴應。
- [Notification設計指導](#)向你展示如何設計實用的notifications以及何時使用它們。這節課將會教你實現大多數常用的notification設計。
- 完整的Demo示例：[NotifyUser.zip](#)

## Lessons

---

- [建立一個Notification](#)

學習如何創建一個notification [Builder](#)，設置需要的特徵，以及發佈notification。

- [當Activity啓動時保留導航](#)

學習如何為一個從notification啓動的[Activity](#)執行適當的導航。

- [更新notifications](#)

學習如何更新與移除notifications

- [使用BigView風格](#)

學習用擴展的notification來創建一個BigView，並且維持老版本的兼容性。

- [顯示notification進度](#)

學習在notification中顯示某個操作的進度，既可以用於那些你可以估算已經完成多少（確定進度，determinate）的操作，也可以用於那些你無法知道完成了多少（不確定進度，indefinite）的操作

# 建立一個Notification

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user/build-notification.html>

- 這節課向你說明如何創建與發佈一個Notification。
- 這節課的例子是基於[NotificationCompat.Builder](#)類的，[NotificationCompat.Builder](#)在[Support Library](#)中。為了給許多各種不同的平臺提供最好的notification支持，你應該使用[NotificationCompat](#)以及它的子類，特別是[NotificationCompat.Builder](#)。

## 創建Notification Buider

- 創建Notification時，可以用[NotificationCompat.Builder](#)對象指定Notification的UI內容與行爲。一個Builder至少包含以下內容：
  - 一個小的icon，用[setSmallIcon\(\)](#)方法設置
  - 一個標題，用[setContentTitle\(\)](#)方法設置。
  - 詳細的文本，用[setContentText\(\)](#)方法設置

例如：

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```

## 定義Notification的Action（行爲）

- 儘管在Notification中Actions是可選的，但是你應該至少添加一種Action。一種Action可以讓用戶從Notification直接進入你應用內的Activity，在這個activity中他們可以查看引起Notification的事件或者做下一步的處理。在Notification中，action本身是由PendingIntent定義的，PendingIntent包含了一個啓動你應用內Activity的Intent。
- 如何構建一個PendingIntent取決於你要啓動的activity的類型。當從Notification中啓動一個activity時，你必須保存用戶的導航體驗。在下面的代碼片段中，點擊Notification啓動一個新的activity，這個activity有效地擴展了Notification的行爲。在這種情形下，就沒必要人為地去創建一個返回棧（更多關於這方面的信息，請查看[Preserving Navigation when Starting an Activity](#)）

```
Intent resultIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity, there's
// no need to create an artificial back stack.
PendingIntent resultPendingIntent =
    PendingIntent.getActivity(
        this,
        0,
        resultIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
   );
```

## 設置Notification的點擊行爲

可以通過調用[NotificationCompat.Builder](#)中合適的方法，將上一步創建的[PendingIntent](#)與一個手勢產生關聯。比方說，當點擊Notification抽屜裏的Notification文本時，啓動一個[activity](#)，可以通過調用[setContentIntent\(\)](#)方法把[PendingIntent](#)添加進去。

例如：

```
PendingIntent resultPendingIntent;
...
mBuilder.setContentIntent(resultPendingIntent);
```

## 發佈Notification

為了發佈notification：

```
* 獲取一個[NotificationManager](http://www.baidu.com/baidu?wd=NotificationManager.&tn=monline_4_dg)實例
* 使用[notify()](developer.android.com/reference/java/lang/Object.html#notify())方法發佈Notification。當你調用[notify()](deve
* 調用[build()](developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#build())方法，會返回-
```

舉個例子：

```
NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

# 啓動Activity時保留導航

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user/navigation.html>

部分設計一個notification的目的是為了保持用戶的導航體驗。為了詳細討論這個課題，請看 [Notifications API引導](#)，分為下列兩種主要情況：

```
* 常規的activity  
你啓動的是你application工作流中的一部分[Activity](developer.android.com/reference/android/app/Activity.html)。  
* 特定的activity  
用戶只能從notification中啓動，才能看到這個[Activity](http://developer.android.com/intl/zh-cn/reference/android/app/Activity.h
```

## 設置一個常規的Activity PendingIntent

設置一個直接啓動的入口Activity的PendingIntent，遵循以下步驟：

1 在manifest中定義你application的Activity層次，最終的manifest文件應該像這個：

```
<activity  
    android:name=".MainActivity"  
    android:label="@string/app_name" >  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
</activity>  
<activity  
    android:name=".ResultActivity"  
    android:parentActivityName=".MainActivity">  
    <meta-data  
        android:name="android.support.PARENT_ACTIVITY"  
        android:value=".MainActivity"/>  
</activity>
```

2 在基於啓動Activity的Intent中創建一個返回棧，比如：

```
int id = 1;  
...  
Intent resultIntent = new Intent(this, ResultActivity.class);  
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);  
// Adds the back stack  
stackBuilder.addParentStack(ResultActivity.class);  
// Adds the Intent to the top of the stack  
stackBuilder.addNextIntent(resultIntent);  
// Gets a PendingIntent containing the entire back stack  
PendingIntent resultPendingIntent =  
    stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);  
...  
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);  
builder.setContentIntent(resultPendingIntent);  
NotificationManager mNotificationManager =  
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);  
mNotificationManager.notify(id, builder.build());
```

# 設置一個特定的Activity PendingIntent

一個特定的Activity不需要一個返回棧，所以你不需要在manifest中定義Activity的層次，以及你不需要調用addParentStack()方法去構建一個返回棧。作為代替，你需要用manifest設置Activity任務選項，以及調用getActivity()創建PendingIntent

1. manifest中，在Activity的標籤中增加下列屬性： android:name="activityclass" activity的完整的類名。  
    android:taskAffinity="" 結合你在代碼裏設置的FLAG\_ACTIVITY\_NEW\_TASK標識，確保這個Activity不會進入application的默認任務。任何與application的默認任務有密切關係的任務都不會受到影響。  
    android:excludeFromRecents="true" 將新任務從最近列表中排除，目的是為了防止用戶不小心返回到它。
2. 建立以及發佈notification：  
    a.創建一個啓動Activity的Intent.  
    b.通過調用setFlags())方法並設置標識FLAG\_ACTIVITY\_NEW\_TASK 與 FLAG\_ACTIVITY\_CLEAR\_TASK，來設置Activity在一個新的，空的任務中啓動。  
    c.在Intent中設置其他你需要的選項。  
    d.通過調用 getActivity()方法從Intent中創建一個PendingIntent，你可以把這個PendingIntent當做 setContentIntent()的參數來使用。下面的代碼片段演示了這個過程：

```
// Instantiate a Builder object.
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
// Creates an Intent for the Activity
Intent notifyIntent =
    new Intent(new ComponentName(this, ResultActivity.class));
// Sets the Activity to start in a new, empty task
notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
    Intent.FLAG_ACTIVITY_CLEAR_TASK);
// Creates the PendingIntent
PendingIntent notifyIntent =
    PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```

# 更新Notification

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user/managing.html>

當你需要對同一事件發佈多次Notification時，你應該避免每次都生成一個全新的Notification。相反，你應該考慮去更新先前的Notification，或者改變它的值，或者增加一些值，或者兩者同時進行。

下面的章節描述瞭如何更新Notifications，以及如何移除它們。

## 改變一個Notification

想要設置一個可以被更新的Notification，需要在發佈它的時候調用[NotificationManager.notify\(ID, notification\)](#)方法為它指定一個notification ID。更新一個已經發佈的Notification，需要更新或者創建一個[NotificationCompat.Builder](#)對象，並從這個對象創建一個[Notification](#)對象，然後用與先前一樣的ID去發佈這個[Notification](#)。

下面的代碼片段演示了更新一個notification來反映事件發生的次數，它把notification堆積起來，顯示一個總數。

```
mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
// Because the ID remains unchanged, the existing notification is
// updated.
mNotificationManager.notify(
    notifyID,
    mNotifyBuilder.build());
...
```

## 移除Notification

Notifications 將持續可見，除非下面任何一種情況發生。

- \* 用戶清除Notification單獨地或者使用“清除所有”（如果Notification能被清除）。
- \* 你在創建notification時調用了 `setAutoCancel(developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder#setAutoCancel(boolean))`。
- \* 你為一個指定的 notification ID調用了`[cancel()]`([developer.android.com/reference/android/app/NotificationManager.html#cancel\(int\)](http://developer.android.com/reference/android/app/NotificationManager.html#cancel(int)))方法。
- \* 你調用了`[cancelAll()]`([developer.android.com/reference/android/app/NotificationManager.html#cancelAll\(\)](http://developer.android.com/reference/android/app/NotificationManager.html#cancelAll()))方法，它將會移除你先前發佈的所有Notification。

# 使用BigView樣式

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user-expanded.html>

Notification抽屜中的Notification主要有兩種視覺展示形式，normal view（平常的視圖，下同）與 big view（大視圖，下同）。Notification的 big view樣式只有當Notification被擴展時才能出現。當Notification在Notification抽屜的最上方或者用戶點擊Notification時纔會展現大視圖。

Big views在Android4.1被引進的，它不支持老版本設備。這節課叫你如何讓把big view notifications合並進你的APP，同時提供normal view的全部功能。更多信息請見[Notifications API guide](#)。

這是一個 normal view的例子

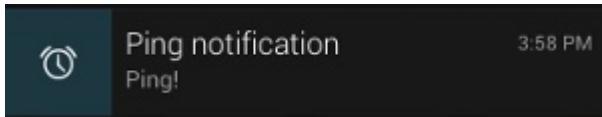


圖1 Normal view notification.

這是一個 big view的例子

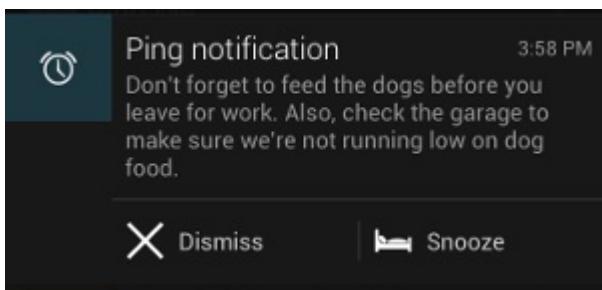


圖2 Big view notification.

在這節課的例子應用中，normal view 與 big view給用戶相同的功能：

- 繼續小睡或者消除Notification
- 一個查看用戶設置的類似計時器的提醒文字的方法，
- normal view 通過當用戶點擊Notification來啓動一個新的activity的方式提供這些特性，記住當你設計你的notifications時，首先在normal view 中提供這些功能，因為很多用戶會與notification交互。

## 設置Notification用來登陸一個新的Activity

這個例子應用用[IntentService](#)的子類（PingService）來構造以及發佈notification。在這個代碼片段中，[IntentService](#)中的方法[onHandleIntent\(\)](#) 指定了當用戶點擊notification時啓動一個新的activity。方法[setContentIntent\(\)](#)定義了pending intent在用戶點擊notification時被激發，因此登陸這個activity。

```
Intent resultIntent = new Intent(this, ResultActivity.class);
resultIntent.putExtra(CommonConstants.EXTRA_MESSAGE, msg);
resultIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
    Intent.FLAG_ACTIVITY_CLEAR_TASK);
```

```
// Because clicking the notification launches a new ("special") activity,  
// there's no need to create an artificial back stack.  
PendingIntent resultPendingIntent =  
    PendingIntent.getActivity(  
        this,  
        0,  
        resultIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT  
);  
  
// This sets the pending intent that should be fired when the user clicks the  
// notification. Clicking the notification launches a new activity.  
builder.setContentIntent(resultPendingIntent);
```

## 構造big view

這個代碼片段展示瞭如何在big view中設置buttons

```
// Sets up the Snooze and Dismiss action buttons that will appear in the  
// big view of the notification.  
Intent dismissIntent = new Intent(this, PingService.class);  
dismissIntent.setAction(CommonConstants.ACTION_DISMISS);  
PendingIntent piDismiss = PendingIntent.getService(this, 0, dismissIntent, 0);  
  
Intent snoozeIntent = new Intent(this, PingService.class);  
snoozeIntent.setAction(CommonConstants.ACTION_SNOOZE);  
PendingIntent piSnooze = PendingIntent.getService(this, 0, snoozeIntent, 0);
```

這個代碼片段展示瞭如何構造一個Builder對象，它設置了big view 的樣式為"big text",同時設置了它的內容為提醒文字。它使用addAction()方法來添加將要在big view中出現的Snooze與Dismiss按鈕（以及它們相關聯的pending intents）。

```
// Constructs the Builder object.  
NotificationCompat.Builder builder =  
    new NotificationCompat.Builder(this)  
        .setSmallIcon(R.drawable.ic_stat_notification)  
        .setContentTitle(getString(R.string.notification))  
        .setContentText(getString(R.string.ping))  
        .setDefaults(Notification.DEFAULT_ALL) // requires VIBRATE permission  
    /*  
     * Sets the big view "big text" style and supplies the  
     * text (the user's reminder message) that will be displayed  
     * in the detail area of the expanded notification.  
     * These calls are ignored by the support library for  
     * pre-4.1 devices.  
     */  
    .setStyle(new NotificationCompat.BigTextStyle()  
        .bigText(msg))  
    .addAction (R.drawable.ic_stat_dismiss,  
        getString(R.string.dismiss), piDismiss)  
    .addAction (R.drawable.ic_stat_snooze,  
        getString(R.string.snooze), piSnooze);
```

# 顯示Notification進度

編寫:[fastcome1985](#) - 原文:<http://developer.android.com/training/notify-user/display-progress.html>

Notifications可以包含一個展示用戶正在進行的操作狀態的動畫進度指示器。如果你可以在任何時候估算這個操作得花多少時間以及當前已經完成多少，你可以用“determinate（確定的，下同）”形式的指示器（一個進度條）。如果你不能估算這個操作的長度，使用“indeterminate（不確定，下同）”形式的指示器（一個活動的指示器）。

進度指示器用[ProgressBar](#)平臺實現類來顯示。

使用進度指示器，可以調用 [setProgress\(\)](#)方法。determinate 與 indeterminate形式將在下面的章節中介紹。

## 展示固定長度的進度指示器

為了展示一個確定長度的進度條，調用 [setProgress\(max, progress, false\)](#)方法將進度條添加進notification，然後發佈這個notification，第三個參數是個boolean類型，決定進度條是 indeterminate (true) 還是 determinate (false)。在你操作進行時，增加progress，更新notification。在操作結束時，progress應該等於max。一個常用的調用 [setProgress\(\)](#)的方法是設置max為100，然後增加progress就像操作的“完成百分比”。

當操作完成的時候，你可以選擇或者讓進度條繼續展示，或者移除它。無論哪種情況下，記得更新notification的文字來顯示操作完成。移除進度條，調用[setProgress\(0, 0, false\)](#)方法.比如：

```
int id = 1;
...
mNotifyManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value, the
                // current completion percentage, and "determinate"
                // state
                mBuilder.setProgress(100, incr, false);
                // Displays the progress bar for the first time.
                mNotifyManager.notify(id, mBuilder.build());
                // Sleeps the thread, simulating an operation
                // that takes time
                try {
                    // Sleep for 5 seconds
                    Thread.sleep(5*1000);
                } catch (InterruptedException e) {
                    Log.d(TAG, "sleep failure");
                }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
            .setProgress(0,0,false);
            mNotifyManager.notify(id, mBuilder.build());
        }
    }
}
```

```

    }
    // Starts the thread by calling the run() method in its Runnable
).start();

```

結果notifications顯示在圖1中，左邊是操作正在進行中的notification的快照，右邊是操作已經完成的notification的快照。

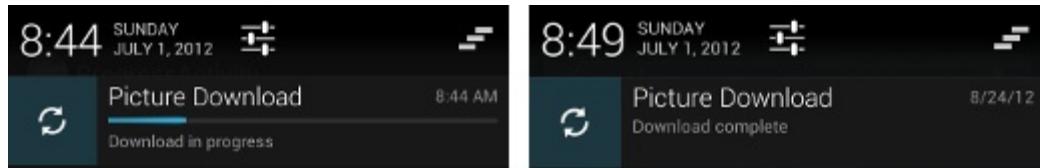


圖1 操作正在進行

中與完成時的進度條

## 展示持續的活動的指示器

為了展示一個持續的(indeterminate)活動的指示器，用`setProgress(0, 0, true)`方法把指示器添加進notification，然後發佈這個notification。前兩個參數忽略，第三個參數決定indicator 還是 indeterminate。結果是指示器與進度條有同樣的樣式，除了它的動畫正在進行。

在操作開始的時候發佈notification，動畫將會一直進行直到你更新notification。當操作完成時，調用 `setProgress(0, 0, false)` 方法，然後更新notification來移除這個動畫指示器。一定要這麼做，否責即使你操作完成了，動畫還是會在那運行。同時也要記得更新notification的文字來顯示操作完成。

為了觀察持續的活動的指示器是如何工作的，看前面的代碼。定位到下面的幾行：

```

// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(id, mBuilder.build());

```

將你找到的代碼用下面的幾行代碼代替，注意 `setProgress()`方法的第三個參數設置成了true,表示進度條是indeterminate類型的。

```

// Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(id, mBuilder.build());

```

結果顯示在圖2中：

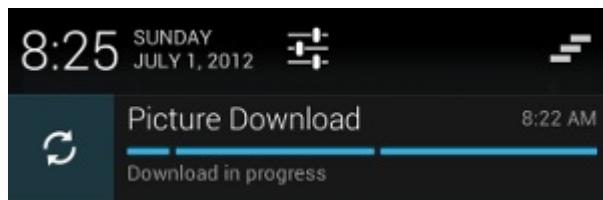


圖2 正在進行的活動的指示器

# 增加搜索功能

---

編寫:Lin-H - 原文:<http://developer.android.com/training/search/index.html>

Android的內置搜索功能，能夠在app中方便地為所有用戶提供一個統一的搜索體驗。根據設備所運行的Android版本，有兩種方式可以在你的app中實現搜索。本節課程涵蓋如何像Android 3.0中介紹的那樣用SearchView添加搜索，使用系統提供的默認搜索框來向下兼容舊版本Android。

## Lessons

---

- [建立搜索界面](#)

學習如何向你的app中添加搜索界面，如何設置activity去處理搜索請求

- [保存並搜索數據](#)

學習在SQLite虛擬數據庫表中用簡單的方法儲存和搜索數據

- [保持向下兼容](#)

通過使用搜索功能來學習如何向下兼容舊版本設備

# 建立搜索界面

編寫:Lin-H - 原文:<http://developer.android.com/training/search/setup.html>

從Android 3.0開始，在action bar中使用SearchView作為item，是在你的app中提供搜索的一種更好方法。像其他所有在action bar中的item一樣，你可以定義SearchView在有足夠空間的時候總是顯示，或設置為一個摺疊操作(collapseable action)，一開始SearchView作為一個圖標顯示，當用戶點擊圖標時再顯示搜索框佔據整個action bar。

Note:在本課程的後面，你會學習對那些不支持SearchView的設備，如何使你的app向下兼容至Android 2.1(API level 7)版本。

## 添加Search View到action bar中

為了在action bar中添加SearchView，在你的工程目錄 res/menu/ 中創建一個名為 options\_menu.xml 的文件，再把下列代碼添加到文件中。這段代碼定義瞭如何創建search item，比如使用的圖標和item的標題。collapseActionView 屬性允許你的SearchView佔據整個action bar，在不使用的時候摺疊成普通的action bar item。由於在手持設備中action bar的空間有限，建議使用 collapseableActionView 屬性來提供更好的用戶體驗。

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/search"
          android:title="@string/search_title"
          android:icon="@drawable/ic_search"
          android:showAsAction="collapseActionView|ifRoom"
          android:actionViewClass="android.widget.SearchView" />
</menu>
```

Note:如果你的menu items已經有一個XML文件，你可以只把 <item> 元素添加入文件。

要在action bar中顯示SearchView，在你的activity中onCreateOptionsMenu()方法內填充XML菜單資源( res/menu/options\_menu.xml ):

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

    return true;
}
```

如果你立即運行你的app，SearchView就會顯示在你app的action bar中，但還無法使用。你現在需要定義SearchView如何運行。

## 創建一個檢索配置

檢索配置(searchable configuration)在 res/xml/searchable.xml 文件中定義了SearchView如何運行。檢索配置中至少要包含一個 android:label 屬性，與Android manifest中的 <application> 或 <activity> android:label 屬性值相同。但我們還是建議添加 android:hint 屬性來告訴用戶應該在搜索框中輸入什麼內容:

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

<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
    android:hint="@string/search_hint" />
```

在你的應用的manifest文件中，聲明一個指向 res/xml/searchable.xml 文件的 `<meta-data>` 元素，來告訴你的應用在哪裏能找到檢索配置。在你想要顯示SearchView的 `<activity>` 中聲明 `<meta-data>` 元素：

```
<activity ... >
    ...
    <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable" />
</activity>
```

在你之前創建的`onCreateOptionsMenu()`方法中，調用`setSearchableInfo(SearchableInfo)`把SearchView和檢索配置關聯在一起：

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

    // 關聯檢索配置和SearchView
    SearchManager searchManager =
        (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView =
        (SearchView) menu.findItem(R.id.search).getActionView();
    searchView.setSearchableInfo(
        searchManager.getSearchableInfo(getApplicationContext()));

    return true;
}
```

調用`getSearchableInfo()`返回一個`SearchableInfo`由檢索配置XML文件創建的對象。檢索配置與SearchView正確關聯後，當用戶提交一個搜索請求時，SearchView會以`ACTION_SEARCH` intent啓動一個activity。所以你現在需要一個能過濾這個intent和處理搜索請求的activity。

## 創建一個檢索activity

當用戶提交一個搜索請求時，SearchView會嘗試以`ACTION_SEARCH`啓動一個activity。檢索activity會過濾`ACTION_SEARCH` intent並在某種數據集中根據請求進行搜索。要創建一個檢索activity，在你選擇的activity中聲明對`ACTION_SEARCH` intent過濾：

```
<activity android:name=".SearchResultsActivity" ... >
    ...
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    ...
</activity>
```

在你的檢索activity中，通過在`onCreate()`方法中檢查`ACTION_SEARCH` intent來處理它。

Note:如果你的檢索activity在single top mode下啓動( android:launchMode="singleTop" )，也要在onNewIntent()方法中處理ACTION\_SEARCH intent。在single top mode下你的activity只有一個會被創建，而隨後啓動的activity將不會在棧中創建新的activity。這種啓動模式很有用，因為用戶可以在當前activity中進行搜索，而不用在每次搜索時都創建一個activity實例。

```
public class SearchResultsActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        handleIntent(getIntent());
    }

    @Override
    protected void onNewIntent(Intent intent) {
        ...
        handleIntent(intent);
    }

    private void handleIntent(Intent intent) {

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUERY);
            //通過某種方法，根據請求檢索你的數據
        }
    }
    ...
}
```

如果你現在運行你的app，SearchView就能接收用戶的搜索請求，以ACTION\_SEARCH intent啓動你的檢索activity。現在就由你來解決如何依據請求來儲存和搜索數據。

# 保存並搜索數據

編寫:Lin-H - 原文:<http://developer.android.com/training/search/search.html>

有很多方法可以儲存你的數據，比如儲存在線上的數據庫，本地的SQLite數據庫，甚至是文本文件。你自己來選擇最適合你應用的存儲方式。本節課程會向你展示如何創建一個健壯的可以提供全文搜索的SQLite虛擬表。並從一個每行有一組單詞-解釋對的文件中將數據填入。

## 創建虛擬表

虛擬表與SQLite表的運行方式類似，但虛擬表是通過回調來向內存中的對象進行讀取和寫入，而不是通過數據庫文件。要創建一個虛擬表，首先為該表創建一個類：

```
public class DatabaseTable {  
    private final DatabaseOpenHelper mDatabaseOpenHelper;  
  
    public DatabaseTable(Context context) {  
        mDatabaseOpenHelper = new DatabaseOpenHelper(context);  
    }  
}
```

在 `DatabaseTable` 類中創建一個繼承 `SQLiteOpenHelper` 的內部類。你必須重寫類 `SQLiteOpenHelper` 中定義的 `abstract` 方法，才能在必要的時候創建和更新你的數據庫表。例如，下面一段代碼聲明瞭一個數據庫表，用來儲存字典app所需的單詞。

```
public class DatabaseTable {  
  
    private static final String TAG = "DictionaryDatabase";  
  
    //字典的表中將要包含的列項  
    public static final String COL_WORD = "WORD";  
    public static final String COL_DEFINITION = "DEFINITION";  
  
    private static final String DATABASE_NAME = "DICTIONARY";  
    private static final String FTS_VIRTUAL_TABLE = "FTS";  
    private static final int DATABASE_VERSION = 1;  
  
    private final DatabaseOpenHelper mDatabaseOpenHelper;  
  
    public DatabaseTable(Context context) {  
        mDatabaseOpenHelper = new DatabaseOpenHelper(context);  
    }  
  
    private static class DatabaseOpenHelper extends SQLiteOpenHelper {  
  
        private final Context mHelperContext;  
        private SQLiteDatabase mDatabase;  
  
        private static final String FTS_TABLE_CREATE =  
            "CREATE VIRTUAL TABLE " + FTS_VIRTUAL_TABLE +  
            " USING fts3 (" +  
            COL_WORD + ", " +  
            COL_DEFINITION + ")";  
  
        DatabaseOpenHelper(Context context) {  
            super(context, DATABASE_NAME, null, DATABASE_VERSION);  
            mHelperContext = context;  
        }  
    }  
}
```

```

@Override
public void onCreate(SQLiteDatabase db) {
    mDatabase = db;
    mDatabase.execSQL(FTS_TABLE_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
        + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS " + FTS_VIRTUAL_TABLE);
    onCreate(db);
}
}
}

```

## 填入虛擬表

現在，表需要數據來儲存。下面的代碼會向你展示如何讀取一個內容為單詞和解釋的文本文件(位於 `res/raw/definitions.txt`)，如何解析文件與如何將文件中的數據按行插入虛擬表中。為防止UI鎖死這些操作會在另一條線程中執行。將下面的一段代碼添加到你的 `DatabaseOpenHelper` 內部類中。

**Tip:**你也可以設置一個回調來通知你的UI `activity`線程的完成結果。

```

private void loadDictionary() {
    new Thread(new Runnable() {
        public void run() {
            try {
                loadWords();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }).start();
}

private void loadWords() throws IOException {
    final Resources resources = mHelperContext.getResources();
    InputStream inputStream = resources.openRawResource(R.raw.definitions);
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

    try {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] strings = TextUtils.split(line, "-");
            if (strings.length < 2) continue;
            long id = addWord(strings[0].trim(), strings[1].trim());
            if (id < 0) {
                Log.e(TAG, "unable to add word: " + strings[0].trim());
            }
        }
    } finally {
        reader.close();
    }
}

public long addWord(String word, String definition) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(COL_WORD, word);
    initialValues.put(COL_DEFINITION, definition);

    return mDatabase.insert(FTS_VIRTUAL_TABLE, null, initialValues);
}

```

任何恰當的地方，都可以調用 `loadDictionary()` 方法向表中填入數據。一個比較好的地方是 `DatabaseOpenHelper` 類

的`onCreate()`方法中，緊隨創建表之後：

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    mDatabase = db;  
    mDatabase.execSQL(FTS_TABLE_CREATE);  
    loadDictionary();  
}
```

## 搜索請求

當你的虛擬表創建好並填入數據後，根據`SearchView`提供的請求搜索數據。將下面的方法添加到`DatabaseTable`類中，用來創建搜索請求的SQL語句：

```
public Cursor getWordMatches(String query, String[] columns) {  
    String selection = COL_WORD + " MATCH ?";  
    String[] selectionArgs = new String[] {query+"*"};  
  
    return query(selection, selectionArgs, columns);  
}  
  
private Cursor query(String selection, String[] selectionArgs, String[] columns) {  
    SQLiteQueryBuilder builder = new SQLiteQueryBuilder();  
    builder.setTables(FTS_VIRTUAL_TABLE);  
  
    Cursor cursor = builder.query(mDatabaseOpenHelper.getReadableDatabase(),  
        columns, selection, selectionArgs, null, null, null);  
  
    if (cursor == null) {  
        return null;  
    } else if (!cursor.moveToFirst()) {  
        cursor.close();  
        return null;  
    }  
    return cursor;  
}
```

調用`getWordMatches()`來搜索請求。任何符合的結果返回到`Cursor`中，可以直接遍歷或是建立一個`ListView`。這個例子是在檢索`activity`的`handleIntent()`方法中調用`getWordMatches()`。請記住，因為之前創建的intent filter，檢索`activity`會在`ACTION_SEARCH` intent中額外接收請求作為變量存儲：

```
DatabaseTable db = new DatabaseTable(this);  
  
...  
  
private void handleIntent(Intent intent) {  
  
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {  
        String query = intent.getStringExtra(SearchManager.QUERY);  
        Cursor c = db.getWordMatches(query, null);  
        //執行Cursor並顯示結果  
    }  
}
```

# 保持向下兼容

編寫:Lin-H - 原文:<http://developer.android.com/training/search/backward-compat.html>

`SearchView`和action bar只在Android 3.0及以上版本可用。為了支持舊版本平臺，你可以回到搜索對話框。搜索框是系統提供的UI，在調用時會覆蓋在你的應用的最頂端。

## 設置最小和目標API級別

要設置搜索對話框，首先在你的manifest中聲明你要支持舊版本設備，並且目標平臺為Android 3.0或更新版本。當你這麼做之後，你的應用會自動地在Android 3.0或以上使用action bar，在舊版本的設備使用傳統的目錄系統：

```
<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="15" />

<application>
    ...

```

## 為舊版本設備提供搜索對話框

要在舊版本設備中調用搜索對話框，可以在任何時候，當用戶從選項目錄中選擇搜索項時，調用`onSearchRequested()`。因為Android 3.0或以上會在action bar中顯示`SearchView`(就像在第一節課中演示的那樣)，所以當用戶選擇目錄的搜索項時，只有Android 3.0以下版本的會調用`onOptionsItemSelected()`。

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.search:
            onSearchRequested();
            return true;
        default:
            return false;
    }
}
```

## 在運行時檢查Android的構建版本

在運行時，檢查設備的版本可以保證在舊版本設備中，不使用不支持的`SearchView`。在我們這個例子中，這一操作在`onCreateOptionsMenu()`方法中：

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

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        SearchManager searchManager =
            (SearchManager) getSystemService(Context.SEARCH_SERVICE);
        SearchView searchView =
            (SearchView) menu.findItem(R.id.search).getActionView();
        searchView.setSearchableInfo(
            searchManager.getSearchableInfo(getComponentName()));
    }
}
```

```
        searchView.setIconifiedByDefault(false);
    }
    return true;
}
```

# 使得你的App內容可被Google搜索

編寫:Lin-H - 原文:<http://developer.android.com/training/app-indexing/index.html>

隨着移動app變得越來越普遍，用戶不僅僅從網站上查找相關信息，也在他們安裝的app上查找。你可以使Google能夠抓取你的app內容，當內容與你自己的網頁一致時，Google搜索的結果會將你的app作為結果展示給用戶。

通過為你的activity提供intent filter，可以使Google搜索展示你的app中特定的內容。Google搜索應用索引(Google Search app indexing)通過在用戶搜索結果的網頁鏈接旁附上相關的app內容鏈接，補充了這一功能。使用移動設備的用戶可以在他們的搜索結果中點擊鏈接來打開你的app，使他們能夠直接瀏覽你的app中的內容，而不需要打開網頁。

要啓用Google搜索應用索引，你需要把有關app與網頁之間聯繫的信息提供給Google。這個過程包括下面幾個步驟：

1. 通過在你的app manifest中添加intent filter來開啓鏈接到你的app中指定內容的深度鏈接。
2. 在你的網站中的相關頁面或Sitemap文件中為這些鏈接添加註解。
3. 選擇允許谷歌爬蟲(Googlebot)在Google Play store中通過APK抓取，建立app內容索引。在早期採用者計劃(early adopter program)中作為參與者加入時，會自動選擇允許。

這節課程，會向你展示如何啓用深度鏈接和建立應用內容索引，使用戶可以從移動設備搜索結果直接打開此內容。

## Lessons

- [為App內容開啓深度鏈接](#)

演示如何添加intent filter來啓用鏈接app內容的深度鏈接

- [為索引指定App內容](#)

演示如何給網站的metadata添加註解，使Google的算法能為app內容建立索引

# 為App內容開啓深度鏈接

編寫:Lin-H - 原文:<http://developer.android.com/training/app-indexing/deep-linking.html>

為使Google能夠抓取你的app內容，並允許用戶從搜索結果進入你的app，你必須給你的app manifest中相關的activity添加intent filter。這些intent filter能使深度鏈接與你的任何activity相連。例如，用戶可以在購物app中，點擊一條深度鏈接來瀏覽一個介紹了自己所搜索的產品的頁面。

## 為你的深度鏈接添加Intent filter

要創建一條與你的app內容相連的深度鏈接，添加一個包含了以下這些元素和屬性值的intent filter到你的manifest中：

<action>

指定ACTION\_VIEW的操作，使得Google搜索可以觸及intent filter。

<data>

添加一個或多個 <data> 標籤，每一個標籤代表一種activity對URI格式的解析，<data> 必須至少包含android:scheme屬性。

你可以添加額外的屬性來改善activity所接受的URI類型。例如，你或許有幾個activity可以接受相似的URI，它們僅僅是路徑名不同。在這種情況下，使用android:path屬性或它的變形( pathPattern 或 pathPrefix )，使系統能辨別對不同的URI路徑應該啓動哪個activity。

<category>

包括BROWSABLE category。BROWSABLE category對於使intent filter能被瀏覽器訪問是必要的。沒有這個category，在瀏覽器中點擊鏈接無法解析到你的app。DEFAULT category是可選的，但建議添加。沒有這個category，activity只能夠使用app組件名稱以顯示(explicit)intent啓動。

下面的一段XML代碼向你展示，你應該如何在manifest中為深度鏈接指定一個intent filter。URI “example://gizmos” 和 “<http://www.example.com/gizmos>” 都能夠解析到這個activity。

```
<activity
    android:name="com.example.android.GizmosActivity"
    android:label="@string/title_gizmos" >
    <intent-filter android:label="@string/filter_title_viewgizmos">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- 接受以"example://gizmos"開頭的 URIs -->
        <data android:scheme="example"
            android:host="gizmos" />
        <!-- 接受以"http://www.example.com/gizmos"開頭的 URIs -->
        <data android:scheme="http"
            android:host="www.example.com"
            android:pathPrefix="gizmos" />
    </intent-filter>
</activity>
```

當你把包含有指定activity內容的URI的intent filter添加到你的app manifest後，Android就可以在你的app運行時，為app與匹配URI的Intent建立路徑。

Note: 對一個URI pattern，intent filter可以只包含一個單一的 `data` 元素，創建不同的intent filter來匹配額外的URI pattern。

學習更多關於定義intent filter，見[Allow Other Apps to Start Your Activity](#)

## 從傳入的intent讀取數據

一旦系統通過一個intent filter啓動你的activity，你可以使用由Intent提供的數據來決定需要處理什麼。調用`getData()`和`getAction()`方法來取出傳入Intent中的數據與操作。你可以在activity生命週期的任何時候調用這些方法，但一般情況下你應該在前期回調如`onCreate()`或`onStart()`中調用。

這個是一段代碼，展示如何從Intent中取出數據：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    Intent intent = getIntent();  
    String action = intent.getAction();  
    Uri data = intent.getData();  
}
```

遵守下面這些慣例來提高用戶體驗：

- 深度鏈接應直接為用戶打開內容，不需要任何提示，插播式廣告頁和登錄頁面。要確保用戶能看到app的內容，即使之前從沒打開過這個應用。當用戶從啓動器打開app時，可以在操作結束後給出提示。這個準則也同樣適用於網站的first click free體驗。
- 遵循Navigation with Back and Up中的設計指導，來使你的app能夠滿足用戶通過深度鏈接進入app後，向後導航的需求。

## 測試你的深度鏈接

你可以使用Android Debug Bridge和activity管理(am)工具來測試你指定的intent filter URI，能否正確解析到正確的app activity。你可以在設備或者模擬器上運行adb命令。

測試intent filter URI的一般adb語法是：

```
$ adb shell am start  
-W -a android.intent.action.VIEW  
-d <URI> <PACKAGE>
```

例如，下面的命令試圖瀏覽與指定URI相關的目標app activity。

```
$ adb shell am start  
-W -a android.intent.action.VIEW  
-d "example://gizmos" com.example.android
```

# 為索引指定App內容

編寫:Lin-H - 原文: <http://developer.android.com/training/app-indexing/enabling-app-indexing.html>

Google的網頁爬蟲機器(Googlebot)會抓取頁面，併為Google搜索引擎建立索引，也能為你的Android app內容建立索引。通過選擇加入這一功能，你可以允許Googlebot通過抓取在Google Play Store中的APK內容，為你的app內容建立索引。要指出哪些app內容你想被Google索引，只需要添加鏈接元素到現有的Sitemap文件，或添加到你的網站中每個頁面的 `<head>` 元素中，以相同的方式為你的頁面添加。

你所共享給Google搜索的深度鏈接必須按照下面的URI格式:

```
android-app://<package_name>/<scheme>/<host_path>
```

構成URI的各部分是:

- `package_name` 代表在Google Play Developer Console中所列出來的你的APK的包名。
- `scheme` 匹配你的intent filter的URI方案。
- `host_path` 找出你的應用中所指定的內容。

下面的幾節敘述如何添加一個深度鏈接URI到你的Sitemap或網頁中。

## 添加深度鏈接(Deep link)到你的Sitemap

要在你的Sitemap中為Google搜索app索引(Google Search app indexing)添加深度鏈接的註解，使用 `<xhtml:link>` 標籤，並指定用作替代URI的深度鏈接。

例如，下面一段XML代碼向你展示如何使用 `<loc>` 標籤指定一個鏈接到你的頁面的鏈接，以及如何使用 `<xhtml:link>` 標籤指定鏈接到你的Android app的深度鏈接。

```
<?xml version="1.0" encoding="UTF-8" ?>
<urlset
  xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <url>
    <loc>example://gizmos</loc>
    <xhtml:link
      rel="alternate"
      href="android-app://com.example.android/example/gizmos" />
  </url>
  ...
</urlset>
```

## 添加深度鏈接到你的網頁中

除了在你的Sitemap文件中，為Google搜索app索引指定深度鏈接外，你還可以在你的HTML標記網頁中給深度鏈接添加註解。你可以在 `<head>` 標籤內這麼做，為每一個頁面添加一個 `<link>` 標籤，並指定用作替代URI的深度鏈接。

例如，下面的一段HTML代碼向你展示如何在頁面中指定一個URL為 `example://gizmos` 的相應的深度鏈接。

```
<html>
<head>
  <link rel="alternate"
    href="android-app://com.example.android/example/gizmos" />
  ...
</head>
<body> ... </body>
```

## 允許Google通過你的app抓取URL請求

一般來說，你可以通過使用[robots.txt](#)文件，來控制Googlebot如何抓取你網站上的公開訪問的URL。當Googlebot為你的app內容建立索引後，你的app可以把HTTP請求當做一般操作。但是，這些請求會被視為從Googlebot發出，發送到你的服務器上。因此，你必須正確配置你的服務器上的 `robots.txt` 文件來允許這些請求。

例如，下面的 `robots.txt` 指示向你展示，如何允許你網站上的特定目錄(如 `/api/`)能被你的app訪問，並限制Googlebot訪問你的網站上的其他目錄。

```
User-Agent: Googlebot
Allow: /api/
Disallow: /
```

學習更多關於如何修改 `robots.txt`，來控制頁面抓取，詳見[Controlling Crawling and Indexing Getting Started](#)。

# Android界面設計

---

These classes teach you how to build a user interface using Android layouts for all types of devices. Android provides a flexible framework for UI design that allows your app to display different layouts for different devices, create custom UI widgets, and even control aspects of the system UI outside your app's window.

## Designing for Multiple Screens

How to build a user interface that's flexible enough to fit perfectly on any screen and how to create different interaction patterns that are optimized for different screen sizes.

## Creating Custom Views

How to build custom UI widgets that are interactive and smooth.

## Creating Backward-Compatible UIs

How to use UI components and other APIs from the more recent versions of Android while remaining compatible with older versions of the platform.

## Implementing Accessibility

How to make your app accessible to users with vision impairment or other physical disabilities.

## Managing the System UI

How to hide and show status and navigation bars across different versions of Android, while managing the display of other screen components.

## Creating Apps with Material Design

How to implement material design on Android.

# 為多屏幕設計

編寫:riverfeng - 原文:<http://developer.android.com/training/multiscreen/index.html>

從小屏手機到大屏電視，android擁有數百種不同屏幕尺寸的設備。因此，設計兼容不同屏幕尺寸的應用程序滿足不同的用戶體驗就變得非常重要。

但是，只是單純的兼容不同的設備類型是遠遠不夠的。每個不同的屏幕尺寸都給用戶體驗帶來不同的可能性和挑戰。所以，為了充分的滿足和打動用戶，你的應用不僅要支持多屏幕，更要針對每個屏幕配置優化你的用戶體驗。

這個課程就將教你如何針對不同屏幕配置來優化你的UI。

本課程提供了一個簡單的示例[NewsReader](#)。這個示例中每節課的代碼展示瞭如何更好的優化多屏幕適配，你也可以將這個示例中的代碼運用到你自己的項目中。

Note：這節課中相關的例子爲了兼容android 3.0以下的版本使用了support library中的Fragment相關APIs。在使用該示例前，請先確定support library已經添加到你的應用中。

## Lessons

- [支持不同屏幕尺寸](#)

這節課程將引導你如何設計適配多種不同尺寸的佈局（通過使用靈活的尺寸規格guige（dimensions），相對佈局（RelativeLayout），屏幕尺寸和方向限定（qualifiers），別名過濾器（alias filter）和點9圖片）。

- [支持不同的屏幕密度](#)

這節課程將演示如何支持不同像素密度的屏幕（使用密度獨立像素（dip）以及爲不同的密度提供合適的位圖（bitmap））。

- [實現自適應UI流（Flows）](#)

這節課將演示如何以UI流（flow）的方式來適配一些屏幕大小/密度組合（動態佈局運行時檢測，響應當前佈局，處理屏幕配置變化）。

# 支持不同的屏幕大小

編寫:riverfeng - 原文:<http://developer.android.com/training/multiscreen/screensizes.html>

這節課教你如何通過以下幾種方式支持多屏幕：

- 1、確保你的佈局能自適應屏幕
- 2、根據你的屏幕配置提供合適的UI佈局
- 3、確保正確的佈局適合正確的屏幕。
- 4、提供縮放正確的位圖（bitmap）

## 使用“wrap\_content”和“match\_parent”

為了確保你的佈局能靈活的適應不同的屏幕尺寸，針對一些view組件，你應該使用wrap\_content和match\_parent來設置他們的寬和高。如果你使用了wrap\_content，view的寬和高會被設置為該view所包含的內容的大小值。如果是match\_parent（在API 8之前是fill\_parent）則會匹配該組件的父控件的大小。

通過使用wrap\_content和match\_parent尺寸值代替硬編碼的尺寸，你的視圖將分別隻使用控件所需要的空間或者被拓展以填充所有有效的空間。比如：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <LinearLayout android:layout_width="match_parent"  
        android:id="@+id/LinearLayout1"  
        android:gravity="center"  
        android:layout_height="50dp">  
        <ImageView android:id="@+id/imageView1"  
            android:layout_height="wrap_content"  
            android:layout_width="wrap_content"  
            android:src="@drawable/logo"  
            android:paddingRight="30dp"  
            android:layout_gravity="left"  
            android:layout_weight="0" />  
        <View android:layout_height="wrap_content"  
            android:id="@+id/view1"  
            android:layout_width="wrap_content"  
            android:layout_weight="1" />  
        <Button android:id="@+id/categorybutton"  
            android:background="@drawable/button_bg"  
            android:layout_height="match_parent"  
            android:layout_weight="0"  
            android:layout_width="120dp"  
            style="@style/CategoryButtonStyle"/>  
    </LinearLayout>  
  
    <fragment android:id="@+id/headlines"  
        android:layout_height="fill_parent"  
        android:name="com.example.android.newsreader.HeadlinesFragment"  
        android:layout_width="match_parent" />  
</LinearLayout>
```

注意上面的例子使用wrap\_content和match\_parent來指定組件尺寸而不是使用固定的尺寸。這樣就能使你的佈局正確的適配不同的屏幕尺寸和屏幕方向（這裏的配置主要是指屏幕的橫豎屏切換）。

例如，下圖演示的就是該佈局在豎屏和橫屏模式下的效果，注意組件的尺寸是自動適應寬和高的。

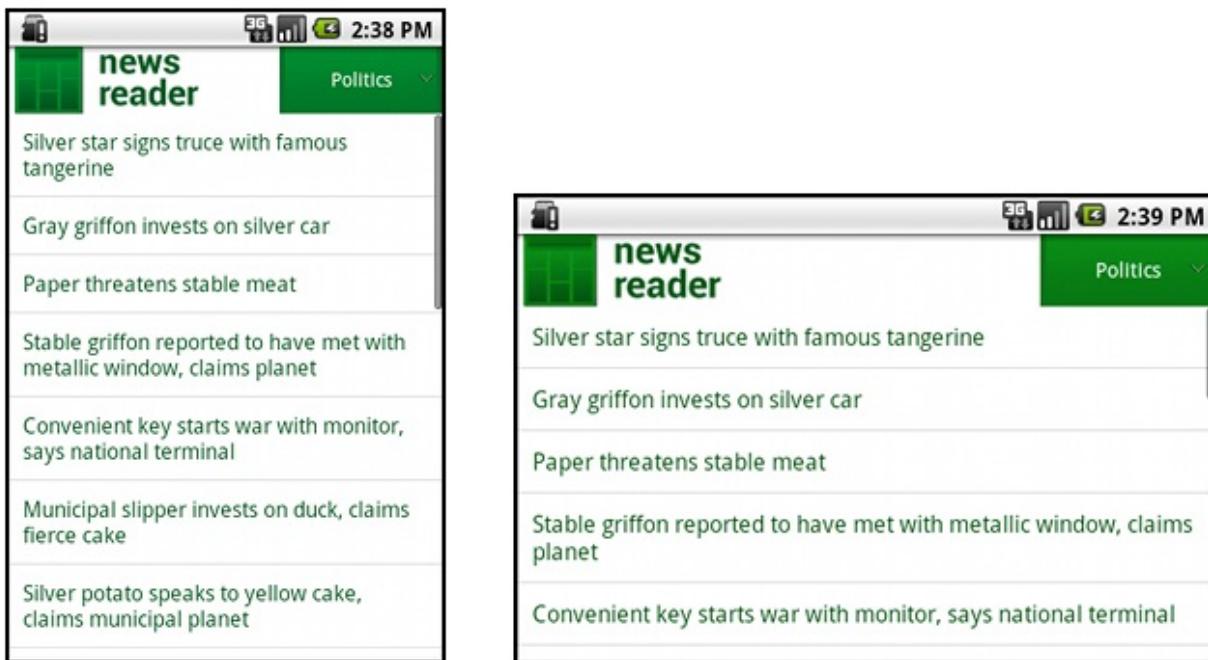


圖1：News Reader示例app（左邊豎屏，右邊橫屏）。

## 使用相對佈局 (RelativeLayout)

你可以使用LinearLayout以及wrap\_content和match\_parent組合來構建複雜的佈局，但是LinearLayout卻不允許你精準的控制它子view的關係，子view在LinearLayout中只能簡單一個接一個的排成行。如果你需要你的子view不只是簡單簡單的排成行的排列，更好的方法是使用RelativeLayout，它允許你指定你佈局中控件與控件之間的關係，比如，你可以指定一個子view在左邊，另一個則在屏幕的右邊。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dp"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@+id/ok"
        android:layout_alignTop="@+id/ok"
        android:text="Cancel" />
```

```
</RelativeLayout>
```

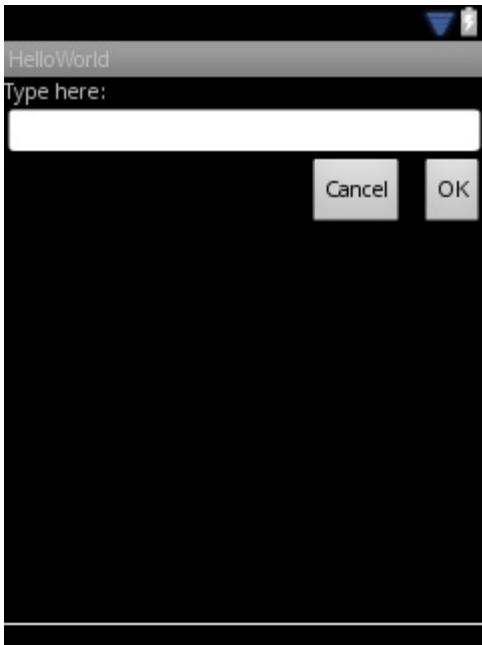


圖2：QVGA（小尺寸屏幕）屏幕下截圖



圖3：WSVGA（大尺寸屏幕）屏幕下截圖

注意：儘管組件的尺寸發生了變化，但是它的子view之間的空間關係還是通過`RelativeLayout.LayoutParams`已經指定好了。

## 使用尺寸限定詞

（譯者注：這裏的限定詞主要是指在編寫佈局文件時，將佈局文件放在加上類似`large`，`sw600dp`等這樣限定詞的文件

夾中，以此來告訴系統根據屏幕選擇對應的佈局文件，比如下面例子的layout-large文件夾）

從上一節的學習里程中，我們知道如何編寫靈活的佈局或者相對佈局，它們都能通過拉伸或者填充控件來適應不同的屏幕，但是它們卻不能為每個不同屏幕尺寸提供最好的用戶體驗。因此，你的應用不應該只是實現靈活的佈局，同時也應該為不同的屏幕配置提供幾種不同的佈局方式。你可以通過配置限定（configuration qualifiers）來做這件事情，它能在運行時根據你當前設備的配置（比如不同的屏幕尺寸設計了不同的佈局）來選擇合適的佈局資源。

比如，很多應用都為大屏幕實現了“兩個窗格”模式（應用可能在一個窗格中實現一個list的item，另外一個則實現list的content），平板和電視都是大到能在一個屏幕上適應兩個窗格，但是手機屏幕卻只能分別顯示。所以，如果你想實現這些佈局，你就需要以下文件：

res/layout/main.xml,單個窗格（默認）佈局：

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="match_parent" />

</LinearLayout>
```

res/layout-large/main.xml,兩個窗格佈局：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>
```

注意第二個佈局文件的目錄名字“large qualifier”，在大尺寸的設備屏幕時（比如7寸平板或者其他大屏幕的設備）就會選擇該佈局文件，而其他比較小的設備則會選擇沒有限定詞的另一個佈局（也就是第一個佈局文件）。

## 使用最小寬度限定詞

在Android 3.2之前，開發者還有一個困難，那就是Android設備的“large”屏幕尺寸，其中包括Dell Streak（設備名稱），老版Galaxy Tab和一般的7寸平板，有很多的應用都想針對這些不同的設備（比如5和7寸的設備）定義不同的佈局，但是這些設備都被定義為了large尺寸屏幕。也是因為這個，所以Android在3.2的時候開始使用最小寬度限定詞。

最小寬度限定詞允許你根據設備的最小寬度（dp單位）來指定不同佈局。比如，傳統的7寸平板最小寬度為600dp，如果你希望你的UI能夠在這樣的屏幕上顯示兩個窗格（不是一個窗格顯示在小屏幕上），你可以使用上節中提到的使用同樣的兩個佈局文件。不同的是，使用sw600來指定兩個方框的佈局使用在最小寬度為600dp的設備上。

res/layout/main.xml,單個窗格（默認）佈局：

```

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="match_parent" />
</LinearLayout>

```

res/layout-sw600dp/main.xml,兩個方框佈局：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>

```

這樣意味着當你的設備的最小寬度等於600dp或者更大時，系統選擇layout-sw600dp/main.xml（兩個窗格）的佈局，而小一點的屏幕則會選擇layout/main.xml（單個窗格）的佈局。然而，在3.2之前的設備上，這樣做並不是很好的選擇。因為3.2之前還沒有將sw600dp作為一個限定詞出現，所以，你還是需要使用large限定詞來做。因此，你還是應該要有一個佈局文件名為res/layout-large/main.xml，和res/layout-sw600dp/main.xml一樣。在下一節中，你將學到如何避免像這樣出現重複的佈局文件。

## 使用佈局別名

最小寬度限定詞只能在Android3.2或者更高的版本上使用。因此，你還是需要使用抽象尺寸（small，normal，large，xlarge）來兼容以前的版本。比如，你想要將你的UI設計為在手機上只顯示一個方框的佈局，而在7寸平板或電視，或者其他大屏幕設備上顯示多個方框的佈局，你可能得提供這些文件：

- res/layout/main.xml：單個窗格佈局
- res/layout-large：多個窗格佈局
- res/layout-sw600dp：多個窗格佈局

最後兩個文件都是一樣的，因為其中一個將會適配Android3.2的設備，而另外一個則會適配其他Android低版本的平板或者電視。為了避免這些重複的文件（維護讓人感覺頭痛就是因為這個），你可以使用別名文件。比如，你可以定義如下佈局：

- res/layout/main.xml，單個方框佈局
- res/layout/main\_twopans.xml，兩個方框佈局

然後添加這兩個文件：

- res/values-large/layout.xml：

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

- res/values-sw600dp/layout.xml :

```
<resources>
    <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

最後兩個文件擁有相同的內容，但它們並沒有真正意義上的定義佈局。它們只是將main\_twopanes設置成爲了別名main，它們分別處在large和sw600dp選擇器中，所以它們能適配Android任何版本的平板和電視（在3.2之前平板和電視可以直接匹配large，而3.2或者以上的則匹配sw600dp）。

## 使用方向限定詞

有一些佈局不管是在橫向還是縱向的屏幕配置中都能顯示的非常好，但是更多的時候，適當的調整一下會更好。在News Reader應用例子中，以下是佈局在不同屏幕尺寸和方向的行爲：

- 小屏幕，縱向：一個窗格加logo
- 小屏幕，橫向：一個窗格加logo
- 7寸平板，縱向：一個窗格加action bar
- 7寸平板，橫向：兩個寬窗格加action bar
- 10寸平板，縱向：兩個窄窗格加action bar
- 10寸平板，橫向：兩個寬窗格加action bar
- 電視，橫向：兩個寬窗格加action bar

這些每個佈局都會在res/layout目錄下定義一個xml文件，如此，應用就能根據屏幕配置的變化根據別名匹配到對應的佈局來適應屏幕。

res/layout/onepane.xml :

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

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="match_parent" />

</LinearLayout>
```

res/layout/onepane\_with\_bar.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:layout_width="match_parent"
        android:id="@+id/LinearLayout1"
        android:gravity="center"
        android:layout_height="50dp">
        <ImageView android:id="@+id/imageView1"
            android:layout_height="wrap_content"
```

```

        android:layout_width="wrap_content"
        android:src="@drawable/logo"
        android:paddingRight="30dp"
        android:layout_gravity="left"
        android:layout_weight="0" />
    <View android:layout_height="wrap_content"
          android:id="@+id/view1"
          android:layout_width="wrap_content"
          android:layout_weight="1" />
    <Button android:id="@+id/categorybutton"
            android:background="@drawable/button_bg"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:layout_width="120dp"
            style="@style/CategoryButtonStyle"/>
</LinearLayout>

<fragment android:id="@+id/headlines"
          android:layout_height="fill_parent"
          android:name="com.example.android.newsreader.HeadlinesFragment"
          android:layout_width="match_parent" />
</LinearLayout>

```

res/layout/twopanes.xml:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="400dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>

```

res/layout/twopanes\_narrow.xml:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.HeadlinesFragment"
              android:layout_width="200dp"
              android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsreader.ArticleFragment"
              android:layout_width="fill_parent" />
</LinearLayout>

```

現在所有可能的佈局我們都已經定義了，唯一剩下的問題是使用方向限定詞來匹配對應的佈局給屏幕。這時候，你就可以使用佈局別名的功能了：

res/values/layouts.xml :

```
<resources>
```

```
<item name="main_layout" type="layout">@layout/onepane_with_bar</item>
<bool name="has_two_panes">false</bool>
</resources>
```

res/values-sw600dp-land/layouts.xml:

```
<resources>
<item name="main_layout" type="layout">@layout/twopanes</item>
<bool name="has_two_panes">true</bool>
</resources>
```

res/values-sw600dp-port/layouts.xml:

```
<resources>
<item name="main_layout" type="layout">@layout/onepane</item>
<bool name="has_two_panes">false</bool>
</resources>
```

res/values-large-land/layouts.xml:

```
<resources>
<item name="main_layout" type="layout">@layout/twopanes</item>
<bool name="has_two_panes">true</bool>
</resources>
```

res/values-large-port/layouts.xml:

```
<resources>
<item name="main_layout" type="layout">@layout/twopanes_narrow</item>
<bool name="has_two_panes">true</bool>
</resources>
```

## 使用.9.png圖片

支持不同的屏幕尺寸同時也意味着你的圖片資源也必須能兼容不同的屏幕尺寸。比如，一個button的背景圖片就必須要適應該button的各種形狀。

如果你在使用組件時可以改變圖片的大小，你很快就會發現這是一個不明確的選擇。因為運行的時候，圖片會被拉伸或者壓縮（這樣容易造成圖片失真）。避免這種情況的解決方案就是使用點9圖片，這是一種能夠指定哪些區域能夠或者不能夠拉伸的特殊png文件。

因此，在設計的圖片需要與組件一起變大變小時，一定要使用點9。若要將位圖轉換為點9，你可以用一個普通的圖片開



始（下圖，是在4倍變焦情況下的圖片顯示）。

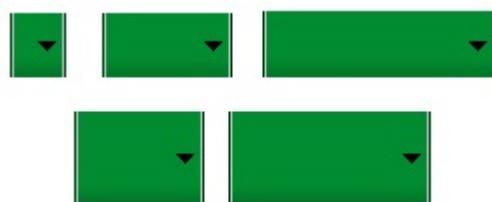
你可以通過sdk中的draw9patch程序（位於tools/directory目錄下）來畫點9圖片。通過沿左側和頂部邊框繪製像素來標記應該被拉伸的區域。也可以通過沿右側和底部邊界繪製像素來標記。就像下圖所示一樣：



請注意，上圖沿邊界的黑色像素。在頂部邊框和左邊框的那些表明圖像的可拉伸區域，右邊和底部邊框則表示內容應該放置的地方。

此外，注意.9.png這個格式，你也必須用這個格式，因為系統會檢測這是一個點9圖片而不是一個普通PNG圖片。

當你將這個應用到組件的背景的時候（通過設置`android:background="@drawable/button"`），`android`框架會自動正確的拉伸圖像以適應按鈕的大小，下圖就是各種尺寸中的顯示效果：



# 兼容不同的屏幕密度

編寫:riverfeng - 原文:<http://developer.android.com/training/multiscreen/screendensities.html>

這節課將教你如何通過提供不同的資源和使用獨立分辨率（dp）來支持不同的屏幕密度。

## 使用密度獨立像素（dp）

設計佈局時，要避免使用絕對像素（absolutepixels）定義距離和尺寸。使用像素單位來定義佈局大小是有問題的。因為，不同的屏幕有不同的像素密度，所以，同樣單位的像素在不同的設備上會有不同的物理尺寸。因此，在指定單位的時候，通常使用dp或者sp。一個dp代表一個密度獨立像素，也就相當於在160 dpi的一個像素的物理尺寸，sp也是一個基本的單位，不過它主要是用在文本尺寸上（它也是一種尺寸規格獨立的像素），所以，你在定義文本尺寸的時候應該使用這種規格單位（不要使用在布尺寸上）。

例如，當你是定義兩個view之間的空間時，應該使用dp而不是px：

```
<Button android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/clickme"  
        android:layout_marginTop="20dp" />
```

當指定文本尺寸時，始終應該使用sp：

```
<TextView android:layout_width="match_parent"  
         android:layout_height="wrap_content"  
         android:textSize="20sp" />
```

## 提供可供選擇的圖片

因為Android能運行在很多不同屏幕密度的設備上，所以，你應該針對不同的設備密度提供不同的bitmap資源：小屏幕（low），medium（中），high（高）以及超高（extra-high）密度。這將能幫助你在所有的屏幕密度中得到非常好的圖形質量和性能。

為了提供更好的用戶體驗，你應該使用以下幾種規格來縮放圖片大小，為不同的屏幕密度提供相應的位圖資源：

```
xhdpi:2.0  
hdpi:1.5  
mdpi:1.0(標準線)  
ldpi:0.75
```

這也就意味着如果在xhdpi設備上你需要一個200x200的圖片，那麼你則需要一張150x150的圖片用於hdpi，100x100的用於mdpi以及75x75的用於ldpi設備。

然後將這些圖片資源放到res/對應的目錄下面，系統會自動根據當前設備屏幕密度自動去選擇合適的資源進行加載：

```
MyProject/  
    res/
```

```
drawable-xhdpi/  
    awesomeimage.png  
drawable-hdpi/  
    awesomeimage.png  
drawable-mdpi/  
    awesomeimage.png  
drawable-ldpi/  
    awesomeimage.png
```

這樣放置圖片資源後，不論你什麼時候使用@drawable/awesomeimage，系統都會給予屏幕的dp來選擇合適的圖片。

如果你想知道更多關於如何為你的應用程序創建icon資源，你可以看看Icon設計指南[Icon Design Guidelines](#).

# 實現自適應UI流（Flows）

編寫:riverfeng - 原文:<http://developer.android.com/training/multiscreen/adoptui.html>

根據當前你的應用顯示的佈局，它的UI流可能會不一樣。比如，當你的應用是雙窗格模式，點擊左邊窗格的條目（item）時，內容（content）顯示在右邊窗格中。如果是單窗格模式中，當你點擊某個item的時候，內容則顯示在一個新的activity中。

## 確定當前佈局

由於每種佈局的實現會略有差別，首先你可能要確定用戶當前可見的佈局是哪一個。比如，你可能想知道當前用戶到底是處於“單窗格”的模式還是“雙窗格”的模式。你可以通過檢查指定的視圖（view）是否存在和可見來實現：

```
public class NewsReaderActivity extends FragmentActivity {
    boolean mIsDualPane;

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

        View articleView = findViewById(R.id.article);
        mIsDualPane = articleView != null &&
                      articleView.getVisibility() == View.VISIBLE;
    }
}
```

注意：使用代碼查詢id為“article”的view是否可見比直接硬編碼查詢指定的佈局更加的靈活。

另一個關於如何適配不同組件是否存在的例子，是在組件執行操作之前先檢查它是否是可用的。比如，在News Reader示例中，有一個按鈕點擊後打開一個菜單，但是這個按鈕僅僅只在Android3.0之後的版本中才能顯示（因為這個功能被ActionBar代替，在API 11+中定義）。所以，在給這個按鈕添加事件之間，你可以這樣做：

```
Button catButton = (Button) findViewById(R.id.categorybutton);
OnClickListener listener = /* create your listener here */;
if (catButton != null) {
    catButton.setOnClickListener(listener);
}
```

## 根據當前佈局響應

一些操作會根據當前的佈局產生不同的效果。比如，在News Reader示例中，當你點擊標題（headlines）列表中的某一條headline時，如果你的UI是雙窗格模式，內容會顯示在右邊的窗格中，如果你的UI是單窗格模式，會啓動一個分開的Activity並顯示：

```
@Override
public void onHeadlineSelected(int index) {
    mArtIndex = index;
    if (mIsDualPane) {
        /* display article on the right pane */
        mArticleFragment.displayArticle(mCurrentCat.getArticle(index));
    } else {
```

```

    /* start a separate activity */
    Intent intent = new Intent(this, ArticleActivity.class);
    intent.putExtra("catIndex", mCatIndex);
    intent.putExtra("artIndex", index);
    startActivity(intent);
}
}

```

同樣，如果你的應用處於多窗格模式，那麼它應該在導航欄中設置帶有選項卡的action bar。而如果是單窗格模式，那麼導航欄應該設置為spinner widget。所以，你的代碼應該檢查哪個方案是最合適的：

```

final String CATEGORIES[] = { "Top Stories", "Politics", "Economy", "Technology" };

public void onCreate(Bundle savedInstanceState) {
    ...
    if (mIsDualPane) {
        /* use tabs for navigation */
        actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_TABS);
        int i;
        for (i = 0; i < CATEGORIES.length; i++) {
            actionBar.addTab(actionBar.newTab().setText(
                CATEGORIES[i]).setTabListener(handler));
        }
        actionBar.setSelectedNavigationItem(selTab);
    }
    else {
        /* use list navigation (spinner) */
        actionBar.setNavigationMode(android.app.ActionBar.NAVIGATION_MODE_LIST);
        SpinnerAdapter adap = new ArrayAdapter(this,
            R.layout.headline_item, CATEGORIES);
        actionBar.setListNavigationCallbacks(adap, handler);
    }
}

```

## 在其他Activity中複用Fragment

在多屏幕設計時經常出現的情況是：在一些屏幕配置上設計一個窗格，而在其他屏幕配置上啟動一個獨立的Activity。例如，在News Reader中，新聞內容文字在大屏幕上顯示在屏幕右邊的方框中，而在小屏幕中，則是由單獨的activity顯示的。

像這樣的情況，你就應該在不同的activity中使用同一個Fragment，以此來避免代碼的重複，而達到代碼複用的效果。比如，ArticleFragment在雙窗格模式下是這樣用的：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.HeadlinesFragment"
        android:layout_width="400dp"
        android:layout_marginRight="10dp"/>
    <fragment android:id="@+id/article"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.ArticleFragment"
        android:layout_width="fill_parent" />
</LinearLayout>

```

在小屏幕中，它又是如下方式被複用的（沒有佈局文件）：

```
ArticleFragment frag = new ArticleFragment();
getSupportFragmentManager().beginTransaction().add(android.R.id.content, frag).commit();
```

當然，如果將這個fragment定義在XML佈局文件中，也有同樣的效果，但是在這個例子中，則沒有必要，因為這個article fragment是這個activity的唯一組件。

當你在設計fragment的時候，非常重要的一點：不要為某個特定的activity設計耦合度高的fragment。通常的做法是，通過定義抽象接口，並在接口中定義需要與該fragment進行交互的activity的抽象方法，然後與該fragment進行交互的activity實現這些抽象接口方法。

例如，在News Reader中，HeadlinesFragment就很好的詮釋了這一點：

```
public class HeadlinesFragment extends ListFragment {
    ...
    OnHeadlineSelectedListener mHeadlineSelectedListener = null;

    /* Must be implemented by host activity */
    public interface OnHeadlineSelectedListener {
        public void onHeadlineSelected(int index);
    }
    ...

    public void setOnHeadlineSelectedListener(OnHeadlineSelectedListener listener) {
        mHeadlineSelectedListener = listener;
    }
}
```

然後，當用戶選擇了一個headline item之後，fragment將通知對應的activity指定監聽事件（而不是通過硬編碼的方式去通知）：

```
public class HeadlinesFragment extends ListFragment {
    ...
    @Override
    public void onItemClick(AdapterView<?> parent,
                           View view, int position, long id) {
        if (null != mHeadlineSelectedListener) {
            mHeadlineSelectedListener.onHeadlineSelected(position);
        }
    }
    ...
}
```

這種技術在[支持平板與手持設備\(Supporting Tablets and Handsets\)](#)有更加詳細的介紹。

## 處理屏幕配置變化

如果使用的是單獨的activity來實現你界面的不同部分，你需要注意的是，屏幕變化（如旋轉變化）的時候，你也應該根據屏幕配置的變化來保持你的UI佈局的一致性。

例如，在傳統的Android3.0或以上版本的7寸平板上，News Reader示例在豎屏的時候使用獨立的activity顯示文章內容，而在橫屏的時候，則使用兩個窗格模式（即內容顯示在右邊的方框中）。這也就意味着，當用戶在豎屏模式下觀看文章的時候，你需要檢測屏幕是否變成了橫屏，如果改變了，則結束當前activity並返回到主activity中，這樣，content就能顯示在雙窗格模式佈局中。

```
public class ArticleActivity extends FragmentActivity {
    int mCatIndex, mArtIndex;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCatIndex = getIntent().getExtras().getInt("catIndex", 0);
        mArtIndex = getIntent().getExtras().getInt("artIndex", 0);

        // If should be in two-pane mode, finish to return to main activity
        if (getResources().getBoolean(R.bool.has_two_panes)) {
            finish();
            return;
        }
        ...
    }
}
```

# 創建自定義View

---

編寫:[kesenhoo](#) - 原文:<http://developer.android.com/training/custom-views/index.html>

Android的framework有大量的Views用來與用戶進行交互並顯示不同種類的數據。但是有時候你的程序有個特殊的需要，而Android內置的views組件並不能實現。這一章節會演示如何創建你自己的views，並使得它們是robust與reusable的。

依賴和要求

Android 2.1 (API level 7) 或更高

你也可以看

- [Custom Components](#)
- [Input Events](#)
- [Property Animation](#)
- [Hardware Acceleration](#)
- [Accessibility developer guide](#)

## Sample

---

[CustomView.zip](#)

## Lesson

---

- [創建一個View類](#)

創建一個像內置的view，有自定義屬性並支持ADT layout編輯器。

- [自定義Drawing](#)

使用Android graphics系統使你的view擁有獨特的視覺效果。

- [使得View是可交互的](#)

用戶期望view對操作反應流暢自然。這節課會討論如何使用gesture detection, physics, 和 animation使你的用戶界面有專業的水準。

- [優化View](#)

不管你的UI如何的漂亮，如果不能以高幀率流暢運行，用戶也不會喜歡。學習如何避免一般的性能問題，和如何使用硬件加速來使你的自定義圖像運行更流暢。

# 創建自定義的View類

編寫:kesenhoo - 原文:<http://developer.android.com/training/custom-views/create-view.html>

設計良好的類總是相似的。它使用一個好用的接口來封裝一個特定的功能，它有效的使用CPU與內存，等等。為了成為一個設計良好的類，自定義的view應該：

- 遵守Android標準規則。
- 提供自定義的風格屬性值並能夠被Android XML Layout所識別。
- 發出可訪問的事件。
- 能夠兼容Android的不同平臺。

Android的framework提供了許多基類與XML標籤用來幫助你創建一個符合上面要求的View。這節課會介紹如何使用Android framework來創建一個view的核心功能。

## 繼承一個View

Android framework裏面定義的view類都繼承自View。你自定義的view也可以直接繼承View，或者你可以通過繼承既有的一個子類(例如Button)來節約一點時間。

為了讓Android Developer Tools能夠識別你的view，你必須至少提供一個constructor，它包含一個Content與一個AttributeSet對象作為參數。這個constructor允許layout editor創建並編輯你的view的實例。

```
class PieChart extends View {  
    public PieChart(Context context, AttributeSet attrs) {  
        super(context, attrs);  
    }  
}
```

## 定義自定義屬性

為了添加一個內置的View到你的UI上，你需要通過XML屬性來指定它的樣式與行為。良好的自定義views可以通過XML添加和改變樣式，為了讓你的自定義的view也有如此的行為，你應該：

- 為你的view在資源標簽下定義自設的屬性
- 在你的XML layout中指定屬性值
- 在運行時獲取屬性值
- 把獲取到的屬性值應用在你的view上

這一節討論如何定義自定義屬性以及指定屬性值，下一節將會實現在運行時獲取屬性值並將它應用。

為了定義自設的屬性，添加 資源到你的項目中。放置於res/values/attrs.xml文件中。下面是一個attrs.xml文件的示例：

```
<resources>  
    <declare-styleable name="PieChart">  
        <attr name="showText" format="boolean" />  
        <attr name="labelPosition" format="enum">  
            <enum name="left" value="0"/>  
            <enum name="right" value="1"/>  
        </attr>
```

```
</declare-styleable>
</resources>
```

上面的代碼聲明瞭2個自設的屬性，`showText`與`labelPosition`，它們都歸屬於`PieChart`的項目下的`styleable`實例。`styleable`實例的名字，通常與自定義的view名字一致。儘管這並沒有嚴格規定要遵守這個convention，但是許多流行的代碼編輯器都依靠這個命名規則來提供statement completion。

一旦你定義了自設的屬性，你可以在layout XML文件中使用它們，就像內置屬性一樣。唯一不同的是你自設的屬性是歸屬於不同的命名空間。不是屬於 `http://schemas.android.com/apk/res/android` 的命名空間，它們歸屬於 `http://schemas.android.com/apk/res/[your package name]`。例如，下面演示瞭如何為`PieChart`使用上面定義的屬性：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res/com.example.customviews">
    <com.example.customviews.charting.PieChart
        custom:showText="true"
        custom:labelPosition="left" />
</LinearLayout>
```

為了避免輸入長串的namespace名字，示例上面使用了 `xmlns` 指令，這個指令可以指派 `custom` 為 `http://schemas.android.com/apk/res/com.example.customviews` namespace的別名。你也可以選擇其他的別名作為你的namespace。

請注意，如果你的view是一個inner class，你必須指定這個view的outer class。同樣的，如果`PieChart`有一個inner class叫做`PieView`。為了使用這個類中自設的屬性，你應該使用 `com.example.customviews.charting.PieChart$PieView`.

## 應用自定義屬性

當view從XML layout被創建的時候，在xml標簽下的屬性值都是從resource下讀取出來並傳遞到view的constructor作為一個`AttributeSet`參數。儘管可以從`AttributeSet`中直接讀取數值，可是這樣做有些弊端：

- 擁有屬性的資源並沒有經過解析
- Styles並沒有運用上

翻譯註：通過 `attrs` 的方法是可以直接獲取到屬性值的，但是不能確定值類型，如：

```
String title = attrs.getAttributeValue(null, "title");
int resId = attrs.getAttributeResourceValue(null, "title", 0);
title = context.getText(resId);
```

都能獲取到 "title" 屬性，但你不知道值是字符串還是resId，處理起來就容易出問題，下面的方法則能在編譯時就發現問題

取而代之的是，通過`obtainStyledAttributes()`來獲取屬性值。這個方法會傳遞一個`TypedArray`對象，它是間接 referenced並且styled的。

Android資源編譯器幫你做了許多工作來使調用`obtainStyledAttributes()`更簡單。對res目錄裏的每一個 `<declare-styleable>` 資源，自動生成的R.java文件定義了存放屬性ID的數組和常量，常量用來索引數組中每個屬性。你可以使用這些預先定義的常量來從`TypedArray`中讀取屬性。這裏就是 `PieChart` 類如何讀取它的屬性：

```
public PieChart(Context context, AttributeSet attrs) {
    super(context, attrs);
    TypedArray a = context.getTheme().obtainStyledAttributes(
        attrs,
        R.styleable.PieChart,
        0, 0);

    try {
        mShowText = a.getBoolean(R.styleable.PieChart_showText, false);
        mTextPos = a.getInteger(R.styleable.PieChart_labelPosition, 0);
    } finally {
        a.recycle();
    }
}
```

請注意TypedArray對象是一個共享資源，必須被在使用後進行回收。

## 添加屬性和事件

Attributes是一個強大的控制view的行爲與外觀的方法，但是他們僅僅能夠在view被初始化的時候被讀取到。為了提供一個動態的行爲，需要暴露出一些合適的getter 與setter方法。下面的代碼演示瞭如何使用這個技巧：

```
public boolean isShowText() {
    return mShowText;
}

public void setShowText(boolean showText) {
    mShowText = showText;
    invalidate();
    requestLayout();
}
```

請注意，在setShowText方法裏面有調用`invalidate()` and `requestLayout()`。這兩個調用是確保穩定運行的關鍵。當view的某些內容發生變化的時候，需要調用invalidate來通知系統對這個view進行redraw，當某些元素變化會引起組件大小變化時，需要調用requestLayout方法。調用時若忘了這兩個方法，將會導致hard-to-find bugs。

自定義的view也需要能夠支持響應事件的監聽器。例如，`PieChart` 暴露了一個自定義的事件 `OnCurrentItemChanged` 來通知監聽器，用戶已經切換了焦點到一個新的組件上。

我們很容易忘記了暴露屬性與事件，特別是當你是這個view的唯一用戶時。請花費一些時間來仔細定義你的view的交互。一個好的規則是總是暴露任何屬性與事件。

## 設計可訪問性

自定義view應該支持廣泛的用戶羣體，包含一些不能看到或使用觸屏的殘障人士。為了支持殘障人士，我們應該：

- 使用 `android:contentDescription` 屬性標記輸入字段。
- 在適當的時候通過調用 `sendAccessibilityEvent()` 發送訪問事件。
- 支持備用控制器，如方向鍵（D-pad）和軌跡球（trackball）等。

對於創建使用的 views的更多消息，請參見Android Developers Guide中的 [Making Applications Accessible](#) 。

# 實現自定義View的繪製

編寫:kesenhoo - 原文:<http://developer.android.com/training/custom-view/custom-draw.html>

自定義view最重要的一個部分是自定義它的外觀。根據你的程序的需求，自定義繪製可能簡單也可能很複雜。這節課會演示一些最常見的操作。

## Override onDraw()

重繪一個自定義的view最重要的步驟是重寫onDraw()方法。onDraw()的參數是一個Canvas對象。Canvas類定義了繪製文本，線條，圖像與許多其他圖形的方法。你可以在onDraw方法裏面使用那些方法來創建你的UI。

在你調用任何繪製方法之前，你需要創建一個Paint對象。

## 創建繪圖對象

android.graphics framework把繪制定義為下面兩類：

- 繪製什麼，由Canvas處理
- 如何繪製，由Paint處理

例如Canvas提供繪製一條直線的方法，Paint提供直線顏色。Canvas提供繪製矩形的方法，Paint定義是否使用顏色填充。簡單來說：Canvas定義你在屏幕上畫的圖形，而Paint定義顏色，樣式，字體，

所以在繪製之前，你需要創建一個或多個Paint對象。在這個PieChart的例子，是在 `init()` 方法實現的，由 `constructor` 調用。

```
private void init() {
    mTextPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    mTextPaint.setColor(mTextColor);
    if (mTextHeight == 0) {
        mTextHeight = mTextPaint.getTextSize();
    } else {
        mTextPaint.setTextSize(mTextHeight);
    }

    mPiePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    mPiePaint.setStyle(Paint.Style.FILL);
    mPiePaint.setTextSize(mTextHeight);

    mShadowPaint = new Paint(0);
    mShadowPaint.setColor(0xff101010);
    mShadowPaint.setMaskFilter(new BlurMaskFilter(8, BlurMaskFilter.Blur.NORMAL));
}

...
```

剛開始就創建對象是一個重要的優化技巧。Views會被頻繁的重新繪製，初始化許多繪製對象需要花費昂貴的代價。在onDraw方法裏面創建繪製對象會嚴重影響到性能並使得你的UI顯得卡頓。

## 處理佈局事件

為了正確的繪製你的view，你需要知道view的大小。複雜的自定義view通常需要根據在屏幕上的大小與形狀執行多次

layout計算。而不是假設這個view在屏幕上的顯示大小。即使只有一個程序會使用你的view，仍然是需要處理屏幕大小不同，密度不同，方向不同所帶來的影響。

儘管view有許多方法是用來計算大小的，但是大多數是不需要重寫的。如果你的view不需要特別的控制它的大小，唯一需要重寫的方法是[onSizeChanged\(\)](#)。

[onSizeChanged\(\)](#)，當你的view第一次被賦予一個大小時，或者你的view大小被更改時會被執行。在[onSizeChanged](#)方法裏面計算位置，間距等其他與你的view大小值。

當你的view被設置大小時，layout manager(佈局管理器)假定這個大小包括所有的view的內邊距(padding)。當你計算你的view大小時，你必須處理內邊距的值。這段 `PieChart.onSizeChanged()` 中的代碼演示該怎麼做：

```
// Account for padding
float xpad = (float)(getPaddingLeft() + getPaddingRight());
float ypad = (float)(getPaddingTop() + getPaddingBottom());

// Account for the label
if (mShowText) xpad += mTextWidth;

float ww = (float)w - xpad;
float hh = (float)h - ypad;

// Figure out how big we can make the pie.
float diameter = Math.min(ww, hh);
```

如果你想更加精確的控制你的view的大小，需要重寫[onMeasure\(\)](#)方法。這個方法的參數是View.MeasureSpec，它會告訴你的view的父控件的大小。那些值被包裝成int類型，你可以使用靜態方法來獲取其中的信息。

這裏是一個實現[onMeasure\(\)](#)的例子。在這個例子中 `PieChart` 試着使它的區域足夠大，使pie可以像它的label一樣大：

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    // Try for a width based on our minimum
    int minw = getPaddingLeft() + getPaddingRight() + getSuggestedMinimumWidth();
    int w = resolveSizeAndState(minw, widthMeasureSpec, 1);

    // Whatever the width ends up being, ask for a height that would let the pie
    // get as big as it can
    int minh = MeasureSpec.getSize(w) - (int)mTextWidth + getPaddingBottom() + getPaddingTop();
    int h = resolveSizeAndState(MeasureSpec.getSize(w) - (int)mTextWidth, heightMeasureSpec, 0);

    setMeasuredDimension(w, h);
}
```

上面的代碼有三個重要的事情需要注意：

- 計算的過程有把view的padding考慮進去。這個在後面會提到，這部分是view所控制的。
- 幫助方法[resolveSizeAndState\(\)](#)是用來創建最終的寬高值的。這個方法會通過比較view的需求大小與spec值，返回一個合適的View.MeasureSpec值，並傳遞到onMeasure方法中。
- [onMeasure\(\)](#)沒有返回值。它通過調用[setMeasuredDimension\(\)](#)來獲取結果。調用這個方法是強制執行的，如果你遺漏了這個方法，會出現運行時異常。

## 繪圖！

每個view的onDraw都是不同的，但是有下面一些常見的操作：

- 繪製文字使用drawText()。指定字體通過調用setTypeface(), 通過setColor()來設置文字顏色。
- 繪製基本圖形使用drawRect(), drawOval(), drawArc(). 通過setStyle()來指定形狀是否需要filled, outlined。
- 繪製一些複雜的圖形，使用Path類. 通過給Path對象添加直線與曲線，然後使用drawPath()來繪製圖形. 和基本圖形一樣，paths也可以通過setStyle來設置是outlined, filled, both.
- 通過創建LinearGradient對象來定義漸變。調用setShader()來使用LinearGradient。
- 通過使用drawBitmap來繪製圖片。

```

protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Draw the shadow
    canvas.drawOval(
        mShadowBounds,
        mShadowPaint
    );

    // Draw the label text
    canvas.drawText(mData.get(mCurrentItem).mLabel, mTextX, mTextY, mTextPaint);

    // Draw the pie slices
    for (int i = 0; i < mData.size(); ++i) {
        Item it = mData.get(i);
        mPiePaint.setShader(it.mShader);
        canvas.drawArc(mBounds,
            360 - it.mEndAngle,
            it.mEndAngle - it.mStartAngle,
            true, mPiePaint);
    }

    // Draw the pointer
    canvas.drawLine(mTextX, mPointerY, mPointerX, mPointerY, mTextPaint);
    canvas.drawCircle(mPointerX, mPointerY, mPointerSize, mTextPaint);
}

```

# 使得View可交互

編寫:kesenhoo - 原文:<http://developer.android.com/training/custom-view/make-interactive.html>

繪製UI僅僅是創建自定義View的一部分。你還需要使得你的View能夠以模擬現實世界的方式來進行反饋。對象應該總是與現實情景能夠保持一致。例如，圖片不應該突然消失又從另外一個地方出現，因為在現實世界裏面不會發生那樣的事情。正確的應該是，圖片從一個地方移動到另外一個地方。

用戶應該可以感受到UI上的微小變化，並對模仿現實世界的細微之處反應強烈。例如，當用戶fling(迅速滑動)一個對象時，應該在開始時感到摩擦帶來的阻力，在結束時感到fling帶動的動力。應該在滑動開始與結束的時候給用戶一定的反饋。

這節課會演示如何使用Android framework的功能來為自定義的View添加那些現實世界中的行爲。

## 處理輸入的手勢

像許多其他UI框架一樣，Android提供一個輸入事件模型。用戶的動作會轉換成觸發一些回調函數的事件，你可以重寫這些回調方法來定製你的程序應該如何響應用戶的輸入事件。在Android中最常用的輸入事件是touch，它會觸發`onTouchEvent(android.view.MotionEvent)`的回調。重寫這個方法來處理touch事件：

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    return super.onTouchEvent(event);  
}
```

Touch事件本身並不是特別有用。如今的touch UI定義了touch事件之間的相互作用，叫做gestures。例如tapping,pulling,flinging與zooming。為了把那些touch的源事件轉換成gestures，Android提供了`GestureDetector`。

通過傳入`GestureDetector.OnGestureListener`的一個實例構建一個`GestureDetector`。如果你只是想要處理幾種gestures(手勢操作)你可以繼承`GestureDetector.SimpleOnGestureListener`，而不用實現`GestureDetector.OnGestureListener`接口。例如，下面的代碼創建一個繼承`GestureDetector.SimpleOnGestureListener`的類，並重寫`onDown(MotionEvent)`。

```
class mListener extends GestureDetector.SimpleOnGestureListener {  
    @Override  
    public boolean onDown(MotionEvent e) {  
        return true;  
    }  
}  
mDetector = new GestureDetector(PieChart.this.getContext(), new mListener());
```

不管你是否使用`GestureDetector.SimpleOnGestureListener`，你必須總是實現`onDown()`方法，並返回true。這一步是必須的，因為所有的gestures都是從`onDown()`開始的。如果你在`onDown()`裏面返回false，系統會認為你想要忽略後續的gesture，那麼`GestureDetector.OnGestureListener`的其他回調方法就不會被執行到了。一旦你實現了`GestureDetector.OnGestureListener`並且創建了`GestureDetector`的實例，你可以使用你的`GestureDetector`來中止你在`onTouchEvent`裏面收到的touch事件。

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    boolean result = mDetector.onTouchEvent(event);
```

```
    if (!result) {
        if (event.getAction() == MotionEvent.ACTION_UP) {
            stopScrolling();
            result = true;
        }
    }
    return result;
}
```

當你傳遞一個touch事件到onTouchEvent()時，若這個事件沒有被辨認出是何種gesture，它會返回false。你可以執行自定義的gesture-deception代碼。

## 創建基本合理的物理運動

Gestures是控制觸摸設備的一種強有力的方式，但是除非你能夠產出一個合理的觸摸反饋，否則將是違反用戶直覺的。一個很好的例子是fling手勢，用戶迅速的在屏幕上移動手指然後擡手離開屏幕。這個手勢應該使得UI迅速的按照fling的方向進行滑動，然後慢慢停下來，就像是用戶旋轉一個飛輪一樣。

但是模擬這個飛輪的感覺並不簡單，要想得到正確的飛輪模型，需要大量的物理，數學知識。幸運的是，Android有提供幫助類來模擬這些物理行爲。[Scroller](#)是控制飛輪式的fling的基類。

要啓動一個fling，需調用 `fling()`，並傳入啓動速率、x、y的最小值和最大值，對於啓動速度值，可以使用[GestureDetector](#)計算得出。

```
@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
    mScroller.fling(currentX, currentY, velocityX / SCALE, velocityY / SCALE, minX, minY, maxX, maxY);
    postInvalidate();
}
```

Note: 儘管速率是通過GestureDetector來計算的，許多開發者感覺使用這個值使得fling動畫太快。通常把x與y設置爲4到8倍的關係。

調用`fling()`時會爲fling手勢設置物理模型。然後，通過調用定期調用 [Scroller.computeScrollOffset\(\)](#)來更新Scroller。[computeScrollOffset\(\)](#)通過讀取當前時間和使用物理模型來計算x和y的位置更新Scroller對象的內部狀態。調用[getCurX\(\)](#)和[getCurY\(\)](#)來獲取這些值。

大多數view通過Scroller對象的x,y的位置直接到[scrollTo\(\)](#)，PieChart例子稍有不同，它使用當前滾動y的位置設置圖表的旋轉角度。

```
if (!mScroller.isFinished()) {
    mScroller.computeScrollOffset();
    setPieRotation(mScroller.getCurY());
}
```

Scroller 類會爲你計算滾動位置，但是他不會自動把哪些位置運用到你的view上面。你有責任確保View獲取並運用到新的座標。你有兩種方法來實現這件事情：

- 在調用`fling()`之後執行`postInvalidate()`，這是爲了確保能強制進行重畫。這個技術需要每次在`onDraw`裏面計算過 scroll offsets(滾動偏移量)之後調用`postInvalidate()`。
- 使用[ValueAnimator](#)在fling是展現動畫，並且通過調用`addUpdateListener()`增加對fling過程的監聽。

這個PieChart 的例子使用了第二種方法。這個方法使用起來會稍微複雜一點，但是它更有效率並且避免了不必要的重畫

的view進行重繪。缺點是ValueAnimator是從API Level 11纔有的。因此他不能運用到3.0的系統之前的版本上。

Note: ValueAnimator雖然是API 11纔有的，但是你還是可以在最低版本低於3.0的系統上使用它，做法是在運行時判斷當前的API Level，如果低於11則跳過。

```
mScroller = new Scroller(getContext(), null, true);
mScrollAnimator = ValueAnimator.ofFloat(0,1);
mScrollAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator valueAnimator) {
        if (!mScroller.isFinished()) {
            mScroller.computeScrollOffset();
            setPieRotation(mScroller.getCurrY());
        } else {
            mScrollAnimator.cancel();
            onScrollFinished();
        }
    }
});
```

## 使過渡平滑

用戶期待一個UI之間的切換是能夠平滑過渡的。UI元素需要做到漸入淡出來取代突然出現與消失。Android從3.0開始有提供[property animation framework](#),用來使得平滑過渡變得更加容易。

使用這套動畫系統時，任何時候屬性的改變都會影響到你的視圖，所以不要直接改變屬性的值。而是使用ValueAnimator來實現改變。在下面的例子中，在PieChart 中更改選擇的部分將導致整個圖表的旋轉，以至選擇的進入選擇區內。ValueAnimator在數百毫秒內改變旋轉量，而不是突然地設置新的旋轉值。

```
mAutoCenterAnimator = ObjectAnimator.ofInt(PieChart.this, "PieRotation", 0);
mAutoCenterAnimator.setIntValues(targetAngle);
mAutoCenterAnimator.setDuration(AUTOCENTER_ANIM_DURATION);
mAutoCenterAnimator.start();
```

如果你想改變的是view的某些基礎屬性，你可以使用[ViewPropertyAnimator](#) ,它能夠同時執行多個屬性的動畫。

```
animate().rotation(targetAngle).setDuration(ANIM_DURATION).start();
```

# 優化自定義View

編寫:kesenhoo - 原文:<http://developer.android.com/training/custom-views/optimizing-view.html>

前面的課程學習到瞭如何創建設計良好的View，並且能夠使之在手勢與狀態切換時得到正確的反饋。下面要介紹的是如何使得view能夠執行更快。為了避免UI顯得卡頓，你必須確保動畫能夠保持在60fps。

## Do Less, Less Frequently

爲了加速你的view，對於頻繁調用的方法，需要儘量減少不必要的代碼。先從onDraw開始，需要特別注意不應該在這裏做內存分配的事情，因爲它會導致GC，從而導致卡頓。在初始化或者動畫間隙期間做分配內存的動作。不要在動畫正在執行的時候做內存分配的事情。

你還需要儘可能的減少onDraw被調用的次數，大多數時候導致onDraw都是因爲調用了invalidate().因此請儘量減少調用invalidate()的次數。如果可能的話，儘量調用含有4個參數的invalidate()方法而不是沒有參數的invalidate()。沒有參數的invalidate會強制重繪整個view。

另外一個非常耗時的操作是請求layout。任何時候執行requestLayout()，會使得Android UI系統去遍歷整個View的層級來計算出每一個view的大小。如果找到有衝突的值，它會需要重新計算好幾次。另外需要儘量保持View的層級是扁平化的，這樣對提高效率很有幫助。

如果你有一個複雜的UI，你應該考慮寫一個自定義的ViewGroup來執行他的layout操作。與內置的view不同，自定義的view可以使得程序僅僅測量這一部分，這避免了遍歷整個view的層級結構來計算大小。這個PieChart 例子展示瞭如何繼承ViewGroup作爲自定義view的一部分。PieChart 有子views，但是它從來不測量它們。而是根據他自身的layout法則，直接設置它們的大小。

## 使用硬件加速

從Android 3.0開始，Android的2D圖像系統可以通過GPU (Graphics Processing Unit)來加速。GPU硬件加速可以提高許多程序的性能。但是這並不是說它適合所有的程序。Android framework讓你能過隨意控制你的程序的各個部分是否啓用硬件加速。

參考 Android Developers Guide 中的[Hardware Acceleration](#) 來學習如何在application, [activity](#), 或 window 層啓用加速。注意除了 Android Guide 的指導之外，你必須要設置你的應用的target API爲11，或更高，通過在你的AndroidManifest.xml 文件中增加 < uses-sdk android:targetSdkVersion = "11" /> 。

一旦你開啓了硬件加速，性能的提示並不一定可以明顯察覺到。移動設備的GPU在某些例如scaling,rotating與translating的操作中表現良好。但是對其他一些任務，比如畫直線或曲線，則表現不佳。爲了充分發揮GPU加速，你應該最大化GPU擅長的操作的數量，最小化GPU不擅長操作的數量。

在下面的例子中，繪製pie是相對來說比較費時的。解決方案是把pie放到一個子view中，並設置View使用LAYER\_TYPE\_HARDWARE來進行加速。

```
private class PieView extends View {  
  
    public PieView(Context context) {  
        super(context);  
        if (!isInEditMode()) {  
            setLayerType(View.LAYER_TYPE_HARDWARE, null);  
        }  
    }  
}
```

```
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    for (Item it : mData) {
        mPiePaint.setShader(it.mShader);
        canvas.drawArc(mBounds,
            360 - it.mEndAngle,
            it.mEndAngle - it.mStartAngle,
            true, mPiePaint);
    }
}

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    mBounds = new RectF(0, 0, w, h);
}

RectF mBounds;
}
```

通過這樣的修改以後，PieChart.PieView.onDraw()只會在第一次現實的時候被調用。之後，pie chart會被緩存為一張圖片，並通過GPU來進行重畫不同的角度。GPU特別擅長這類的事情，並且表現效果突出。

緩存圖片到hardware layer會消耗video memory，而video memory又是有限的。基於這樣的考慮，僅僅在用戶觸發scrolling的時候使用 `LAYER_TYPE_HARDWARE`，在其他時候，使用 `LAYER_TYPE_NONE`。

# 創建向後兼容的UI

---

編寫:spencer198711 - 原文:<http://developer.android.com/training/backward-compatible-ui/index.html>

這一課展示瞭如何以向後兼容的方式使用在新版本的Android上可用的UI組件和API，確保你的應用在之前的版本上依然能夠運行。

貫穿整個課程，在Android 3.0被新引入的Action Bar Tabs功能在本課程中作為指導例子，但是你可以在其他UI組件和API功能上運用這種方式。

## Sample

---

<http://developer.android.com/shareables/training/TabCompat.zip>

## Lessons

---

- [抽象出新的APIs](#)

決定你的應用需要的功能和接口。學習如何為你的應用定義面向特定應用的、作為中間媒介並抽象出UI組件具體實現的java接口。

- [代理至新的APIs](#)

學習如何創建使用新的APIs的接口的具體實現

- [使用舊的APIs實現新API的效果](#)

學習如何創建使用老的APIs的自定義的接口實現

- [使用能感知版本的組件](#)

學習如何在運行的時候去選擇一個具體的實現，並且開始在你的應用中使用接口。

# 抽象出新的APIs

編寫:spencer198711 - 原文:<http://developer.android.com/training/backward-compatible-ui/abstracting.html>

假如你想使用Action Bar Tabs作為你的應用的頂層導航的主要形式。不幸的是，ActionBar APIs只在Android 3.0（API等級11）之後才能使用。因此，如果你想要在運行之前版本的Android平臺的設備上分發你的應用，你需要提供一個支持新的API的實現，同時提供一個回退機制，使得能夠使用舊的APIs。

在本課程中，使用了具有面向特定版本實現的抽象類去構建一個tab頁形式的用戶界面，並以此提供向後兼容性。這一課描述瞭如何為新的tab API創建一個抽象層，並以此作為構建tab組件的第一步。

## 為抽象做準備

在Java編程語言中，抽象包含了創建一個或者多個接口或抽象類去隱藏具體的實現細節。在新版本的Android API的情況中，你可以使用抽象去構建能感知版本的組件，這個組件會在新版本的設備上使用當前的APIs，當回退到老的設備上同時存在兼容的APIs。

當使用這種方法時，你首先需要決定哪些要使用的類需要提供向後兼容，然後去根據新類中的public接口去創建抽象類。在創建抽象接口的過程中，你應該儘可能多的為新APIs創建鏡像。這會最大化前向兼容性，使得在將來當這些接口不再需要的時候，廢棄這些接口會更加容易。

在為新的APIs創建抽象類之後，任何數量的實現都可以在運行的過程中去創建和選擇使用哪種。出於後向兼容的目的，這些實現可以通過所需的API級別而有所變化。一個實現可能會使用最新發佈的APIs，而其他的則會去使用比較老的APIs。

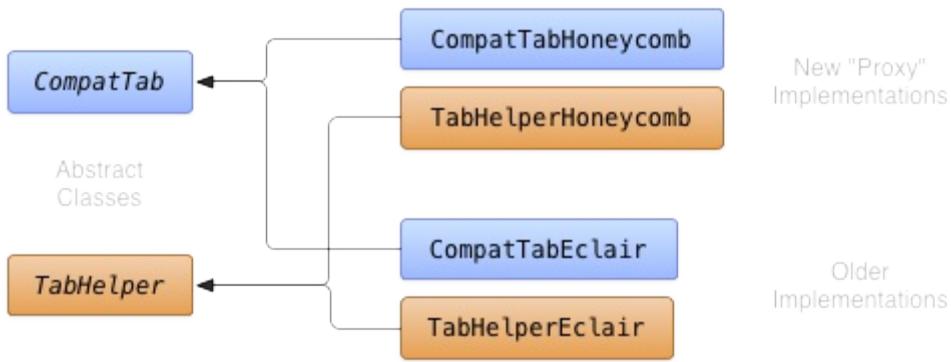
## 創建抽象的Tab接口

為了能夠創建一個向後兼容的tabs，你首先需要決定你的應用需要哪些功能和哪些特定的APIs接口。在頂層分節tabs的情況下，假設你有以下功能需求：

1. 顯示圖標和文本的Tab指示器
2. Tabs可以跟一個Fragment實例向關聯
3. Activity可以監聽到Tab變化

提前準備這些需求能夠讓你控制抽象層的範圍。這意味着你可以花更少的時間去創建抽象層的多個具體實現，並很快就能使用這些新的後向兼容的實現。

Tabs的關鍵APIs是ActionBar和ActionBar.Tab，為了能夠使得tab能夠感知Android版本，這些是需要抽象出來的APIs。這個示例項目的需求要求同Eclair(API等級5)保持一致性，同時能夠利用Honeycomb(API等級11)中新的tab功能。一張展示能夠支持這兩種實現的類結構和它們的抽象父類的圖顯示如下：



- 圖1.抽象基類和版本相關的子類實現類結構圖

## Abstract ActionBar.Tab

通過創建一個代表tab的抽象類來開始着手構建tab抽象層，這個類是ActionBar.Tab接口的鏡像：

```

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();
    ...
}
    
```

在這裏，為了簡化諸如tab對象和Activity的聯繫（未在代碼片段中顯示）等公共的功能，你可以使用一個抽象類而不是去使用接口。

## 抽象出Action Bar Tab的方法

下一步，定義一個能夠允許你往Activity中創建和添加tab抽象類，並定義類似ActionBar.newTab()和ActionBar.addTab()的方法。

```

public abstract class TabHelper {
    ...
    public CompatTab newTab(String tag) {
        // This method is implemented in a later lesson.
    }
    public abstract void addTab(CompatTab tab);
    ...
}
    
```

在下一課程中，你將會創建TabHelper和CompatTab的實現，它能夠在新舊不同的平臺版本上都能工作。

# 代理至新的APIs

編寫: spencer198711 - 原文:<http://developer.android.com/training/backward-compatible-ui/new-implementation.html>

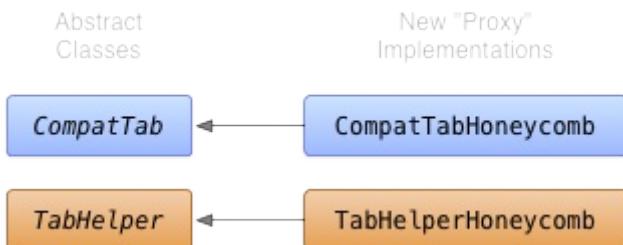
這一課展示瞭如何編寫CompatTab和TabHelper等抽象類的子類，並且使用了較新的APIs。你的應用可以在支持這些新的APIs的平臺版本的設備上使用這種實現方式。

## 使用較新的APIs實現Tabs

CompatTab和TabHelper抽象類的具體子類是一種代理實現，它們使用了使用較新的APIs。由於抽象類在之前的課程中定義並且是對新APIs接口（類結構、方法簽名等等）的鏡像，使用新APIs的具體子類只是簡單的代理方法調用和方法調用的結果。

你可以在這些具體子類中直接使用較新的APIs，由於使用延遲類加載的方式，在早期版本的設備上並不會發生崩潰現象。這些類在首次次被訪問（實例化類對象或者訪問類的靜態屬性或靜態方法）的時候纔會去加載並初始化。因此，只要你不Honeycomb之前的設備上實例化Honeycomb相關的實現，dalvik虛擬機都不會拋出`VerifyError`異常。

對於本實現，一個比較好的命名約定是把具體子類需要的API等級或者版本名字附加在APIs接口的後邊。例如，本地tab實現可以由 `CompatTabHoneycomb` 和 `abHelperHoneycomb` 這兩個類提供，名字後面附加Honeycomb是由於它們都依賴於Android 3.0 ( API等級11 ) 之後版本的APIs。



• 圖1. Honeycomb上tabs實現的類關係圖.

## 實現CompatTabHoneycomb

`CompatTabHoneycomb` 是 `CompatTab` 抽象類的具體實現並用來引用單獨的tabs。 `CompatTabHoneycomb` 只是簡單的代理`ActionBar.Tab`對象的方法調用。 開始使用`ActionBar.Tab`的APIs實現`CompatTabHoneycomb`：

```
public class CompatTabHoneycomb extends CompatTab {
    // The native tab object that this CompatTab acts as a proxy for.
    ActionBar.Tab mTab;

    ...
    protected CompatTabHoneycomb(FragmentActivity activity, String tag) {
        ...
        // Proxy to new ActionBar.newTab API
        mTab = activity.getActionBar().newTab();
    }
    public CompatTab setText(int resId) {
        // Proxy to new ActionBar.Tab.setText API
        mTab.setText(resId);
        return this;
    }
    ...
}
```

```
// Do the same for other properties (icon, callback, etc.)  
}
```

## 實現TabHelperHoneycomb

TabHelperHoneycomb 是 TabHelper 抽象類的具體實現，TabHelperHoneycomb 代理方法調用到ActionBar對象，而這個ActionBar對象是從包含他的Activity中獲取的。

實現 TabHelperHoneycomb，代理其方法調用到ActionBar的API：

```
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 is a CompatTabHoneycomb instance, so its  
        // native tab object is an ActionBar.Tab.  
        mActionBar.addTab((ActionBar.Tab) tab.getTab());  
    }  
  
    // The other important method, newTab() is part of  
    // the base implementation.  
}
```

# 使用舊的APIs實現新API的效果

編寫: spencer198711 - 原文:<http://developer.android.com/training/backward-compatible-ui/older-implementation.html>

這一課討論瞭如何創建一個支持舊的設備並且與新的APIs接口相同的實現。

## 決定一個替代方案

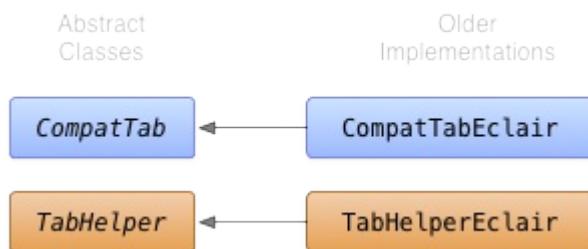
在以向後兼容的方式使用較新的UI功能的時候，最具挑戰的任務是為舊的平臺版本決定一個解決方案。在很多情況下，使用舊的UI框架中的功能是有可能完成這些新的UI組件的。例如：

- Action Bar可以使用水平的包含圖片按鈕的LinearLayout來實現，這個在Activity中的LinearLayout作為自定義標題欄或者僅僅作為視圖。下拉功能行為可以使用設備的菜單按鈕來實現。
- Action Bar的tab頁可以使用包含按鈕的水平的LinearLayout，或者使用TabWidget UI控件來實現。
- NumberPicker和Switch控件可以分別通過使用Spinner和ToggleButton控件來實現。
- ListPopupWindow和PopupMenu控件可以通過使用PopupWindow來實現。

為了往老的設備上向後移植UI組件，這些一般不是一刀切的解決方案。注意用戶體驗：在老的設備上，用戶可能不熟悉新的界面設計模式和UI組件，思考一下如何使用熟悉的控件去實現相同的功能。在很多種情況下，這些通常不會被注意到，特別是在如果新的UI組件在應用程序的生態系統中是突出的（比如Action Bar），或者交互模型是非常簡單和直觀的（比如使用ViewPager去滑動界面）。

## 使用舊的APIs實現Tabs

你可以使用TabWidget和TabHost（儘管其中一個也可以使用水平方向的Button控件）去創建Action Bar Tabs的老的實現。可以在TabHelperEclair和CompatTabEclair的類中去實現，因為這些實現使用了不遲於Android 2.0（Eclair）的APIs。



● 圖1. Eclair版本上實現tabs的類圖

CompatTabEclair 在實例變量中保存了諸如tab文本和tab圖標等tab屬性，因為在老的版本中沒有ActionBar.Tab對象去處理這些數據存儲。

```
public class CompatTabEclair extends CompatTab {  
    // Store these properties in the instance,  
    // as there is no ActionBar.Tab object.  
    private CharSequence mText;  
    ...  
  
    public CompatTab setText(int resId) {  
        // Our older implementation simply stores this  
        ...  
    }  
}
```

```

    // information in the object instance.
    mText = mActivity.getResources().getText(resId);
    return this;
}

...
// Do the same for other properties (icon, callback, etc.)
}

```

`TabHelperEclair` 利用了`TabHost`控件的方法去創建`TabHost.TabSpec`對象和tab的頁面指示效果：

```

public class TabHelperEclair extends TabHelper {
    private TabHost mTabHost;
    ...

    protected void setUp() {
        if (mTabHost == null) {
            // Our activity layout for pre-Honeycomb devices
            // must contain a TabHost.
            mTabHost = (TabHost) mActivity.findViewById(
                android.R.id.tabhost);
            mTabHost.setup();
        }
    }

    public void addTab(CompatTab tab) {
        ...
        TabSpec spec = mTabHost
            .newTabSpec(tag)
            .setIndicator(tab.getText()); // And optional icon
        ...
        mTabHost.addTab(spec);
    }

    // The other important method, newTab() is part of
    // the base implementation.
}

```

現在你已經有了兩種 `CompatTab` 和 `TabHelper` 的實現，一種是使用了新的APIs為了能夠在Android 3.0或其後版本設備上能夠運行，另一種則是使用了舊的APIs為了在Android 2.0或之前的設備上能夠運行。下一課討論在應用中使用這兩種實現。

# 使用能感知版本的組件

編寫:spencer198711 - 原文:<http://developer.android.com/training/backward-compatible-ui/using-component.html>

既然對 TabHelper 和 CompatTab 你已經有了兩種具體實現，一個為Android 3.0和其後版本，一個為Android 3.0之前的版本。現在，該使用這些實現做些事情了。這一課討論了創建在這兩種實現之前切換的邏輯，創建能夠感知版本的界面佈局，最終使用我們創建的後向兼容的UI組件。

## 添加切換邏輯

TabHelper 抽象類基於當前設備的平臺版本，是用來創建適當版本的 TabHelper 和 CompatTab 實例的工廠類：

```
public abstract class TabHelper {  
    ...  
    // Usage is 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);  
        }  
    }  
  
    // Usage is 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);  
        }  
    }  
    ...  
}
```

## 創建能感知版本的Activity佈局

下一步是提供能夠支持兩種tab實現的Activity界面佈局。對於老的實現（TabHelperEclair），你需要確保你的界面佈局包含TabWidget和TabHost，同時存在一個包含tab內容的佈局容器。

res/layout/main.xml:

```
<!-- This layout is for API level 5-10 only. -->  
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"  
        android:id="@+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="@+id/tabs"  
            android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content" />

    <FrameLayout
        android:id="@+id/tabcontent"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

</LinearLayout>
</TabHost>
```

對於 `TabHelperHoneycomb` 的實現，你唯一要做的就是一個包含tab內容的`FrameLayout`，這是由於`ActionBar`已經提供了 tab相關的頁面。

`res/layout-v11/main.xml`:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabcontent"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

在運行的時候，Android將會根據平臺版本去決定使用哪個版本的 `main.xml` 佈局文件。這根上一節中選擇哪一個版本的 `TabHelper` 所展示的邏輯是相同的。

## 在Activity中使用TabHelper

在Activity的`onCreate()`方法中，你可以獲得一個 `TabHelper` 對象，並且使用以下代碼添加tabs：

```
@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);
}
```

當運行這個應用的時候，代碼會自動顯示對應的界面佈局和實例化對應的 `TabHelperHoneycomb` 或 `TabHelperEclair` 對象，而實際使用的類對於Actvity來說是不透明的，因為它們擁有共同的 `TabHelper` 接口。

以下是這種實現運行在Android 2.3和Android 4.0上的界面截圖：



7:44

## Tab Demo

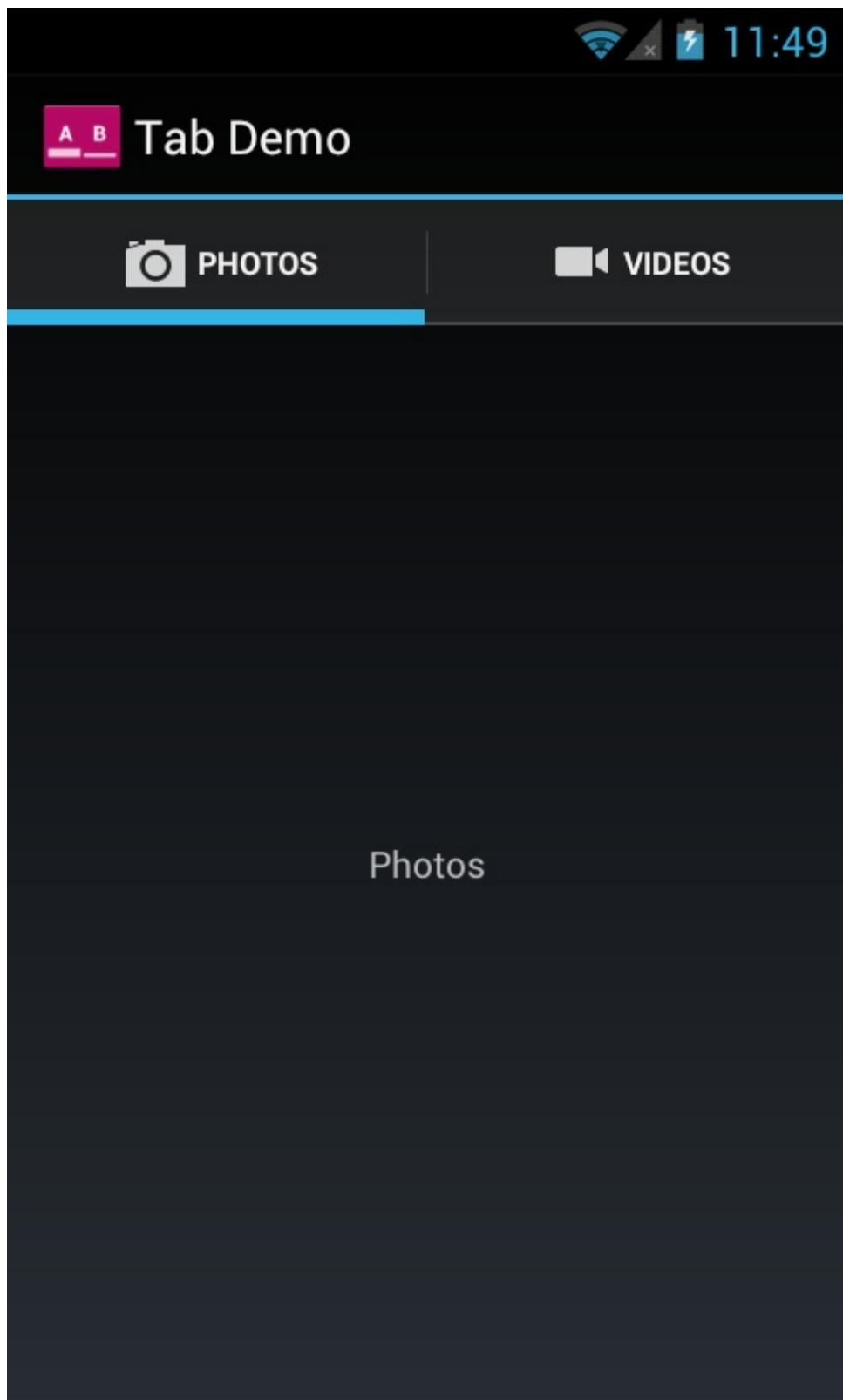


Photos



Videos

Photos



● 圖1.向後兼容的tabs運行在Android 2.3設備上（使用TabHelperEclair）和運行在Android 4.0設備上的截圖

# 實現輔助功能

---

編寫:KOST - 原文:<http://developer.android.com/training/accessibility/index.html>

當我們需要儘可能擴大我們用戶的基數的時候，就要開始注意我們軟件的可達性了(Accessibility 易接近，可親性)。在界面中展示提示對大多數用戶而言是可行的，比如說當按鈕被按下時視覺上的變化，但是對於那些視力上有些缺陷的用戶而言效果就不是那麼理想了。

本章將給您演示如何最大化利用Android框架中的Accessibility特性。包括如何利用焦點導航(focus navigation)與內容描述(content description)對你的應用的可達性進行優化。也包括了創建Accessibility Service，使用戶與應用（不僅僅是你自己的應用）之間的交互更加容易。

## Lessons

---

- [開發Accessibility應用](#)

學習如何讓你的程序更易用，具有可達性。允許使用鍵盤或者十字鍵(directional pad)來進行導航，利用Accessibility Service特性設置標籤或執行事件來打造更舒適的用戶體驗。

- [編寫 Accessibility Services](#)

編寫一個Accessibility Service來監聽可達性事件，利用這些不同類型的事件和內容描述來幫助用戶與應用的交互。本例將會實現利用一個TTS引擎來向用戶發出語音提示的功能。

# 開發輔助程序

編寫:KOST - 原文:<http://developer.android.com/training/accessibility/accessible-app.html>

本課程將教您：

1. 添加內容描述(Content Descriptions)
2. 設計焦點導航 (Focus Navigation)
3. 觸發可達性事件(Accessibility Events)
4. 測試你的程序

Android平臺本身有一些專注可達性的特性，這些特性可以幫助你專門為那些視覺上或生理上有缺陷的用戶在應用上做特別的優化。然而，正確的優化方式或最簡單利用這個特性的方法往往不是那麼顯而易見的。本課程將給您演示如何利用和實現這些策略和平臺的特性功能，構建一個更友好的具有可達性的Android應用。

## 添加內容描述

一個好的交互界面上的元素通常不需要特別使用一個標籤來表明這個元素的作用。例如對於一個任務型應用來說，一個項目旁邊的勾選框表達的意思就非常明確，或者對於一個文件管理應用，垃圾桶的圖標表達的意思也非常清楚。然而對於具有視覺障礙的用戶來說，其他類型的UI交互提示是有必要的。

幸運的是，我們可以很輕鬆的給一個UI元素加上標籤，這樣類似於TalkBack這樣的基於語音的Accessibility Service就可以將標籤的內容朗讀出來。如果你的標簽在整個應用的生命週期中不太可能會發生變化(比如‘停止’或者‘購買’)，你就可以在XML佈局文件中對`android:contentDescription`屬性進行設置。代碼如下：

```
<Button  
    android:id="@+id/pause_button"  
    android:src="@drawable/pause"  
    android:contentDescription="@string/pause"/>
```

然而，在很多情況下描述的內容是基於上下文環境的，比如說一個開關按鈕的狀態，或者在list中一片可選的數據項。在運行時編輯內容描述可以使用`setContentDescription()`方法，代碼如下：

```
String contentDescription = "Select " + strValues[position];  
label.setContentDescription(contentDescription);
```

將以上功能添加進您的代碼是提高您應用可達性的最簡單的方法。嘗試着將那些有用的地方都加入內容描述，但同時要避免像web開發者那樣將所有的元素都標註，那樣會產生大量的無用信息。比如說，不要將應用圖標的內容描述設置為‘應用圖標’。這只會對用戶的瀏覽產生干擾。

來試試吧！下載TalkBack(谷歌開發的一款可達性應用)，在Settings > Accessibility > TalkBack將它開啟。然後使用你的應用聽聽看TalkBack發出的語音提示。

## 設計焦點導航

你的應用除了支持觸摸操作外，更應該支持其他的導航方式。很多Android設備不僅僅提供了觸摸屏，還提供了其他的導航硬件比如說十字鍵、方向鍵、軌跡球等等。除此之外，最新的Android發行版本也支持藍牙或USB的外接設備，比如鍵盤等等。

為了實現這種方式的導航，一切用戶可以用來可導航的元素(navigational elements)都需要設置為focusable（聚焦），它可以在運行時通過View.setFocusable()方法來進行設定，或者也可以在XML佈局文件中使用android:focusable來設置。

每個UI控件有四個屬

性，`android:nextFocusUp`,`android:nextFocusDown`,`android:nextFocusLeft`,`android:nextFocusRight`,用戶在導航時可以利用這些屬性來指定下一個焦點的位置。系統會自動根據佈局的方向來確定導航的順序，如果在您的應用中系統提供的方案並不合適，您可以用這些屬性來進行自定義的修改。

比如說，下面就是一個關於按鈕和標籤的例子，他們都是可聚焦的(focusable)，按向下鍵會將焦點從按鈕移到文字上，按向上會重新將焦點移到按鈕上。

```
<Button android:id="@+id/doSomething"
    android:focusable="true"
    android:nextFocusDown="@+id/label"
    ... />
<TextView android:id="@+id/label"
    android:focusable="true"
    android:text="@string/labelText"
    android:nextFocusUp="@+id/doSomething"
    ... />
```

證實您的應用運行正確的直觀方法，最簡單的方式就是在Android虛擬機裏運行您的應用，然後使用虛擬器的方向鍵來在各個元素之間導航，使用OK按鈕來代替觸摸操作。

## 觸發可達性事件

如果你在你的Android框架中使用了View組件，當你選中了一個View或者是焦點變化的時候，可達性事件(AccessibilityEvent)都會產生。這些事件會被傳遞到Accessibility Service中進行處理，實現一些輔助功能，如語音提示等。

如果你寫了一個自定義的View，請確保它在合適的時候產生事件。使用sendAccessibilityEvent(int)函數可以產生可達性事件，其中的參數表示事件的類型。完整的可達性事件類型可查閱[AccessibilityEvent](#)參考文檔。

比如說，你拓展了一個圖片的View，你希望在它聚焦的時候使用鍵盤打字可以在其中插入題注，這時候發送一個TYPE\_VIEW\_TEXT\_CHANGED事件就非常合適，儘管它不是本身就構建在這個圖片View中的。產生事件的代碼如下：

```
public void onTextChanged(String before, String after) {
    ...
    if (AccessibilityManager.getInstance(mContext).isEnabled()) {
        sendAccessibilityEvent(AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED);
    }
    ...
}
```

## 測試你的程序

請確保您在添加可達性功能後測試它的有效性。為了測試內容描述可達性事件，請安裝並啓用一個Accessibility Service。比如說使用TalkBack，它是一個免費的開源的屏幕讀取軟件，可在Google Play上進行下載。Service啓動後，請測試您應用中所有的功能，同時聽聽TalkBack的語音反饋。

同時，嘗試着用一個方向控制器來控制你的應用，而非使用直接觸摸的方式。你可以使用一個物理設備，比如十字鍵、軌跡球等。如果沒有條件，可以使用android虛擬器，它提供了虛擬的按鍵控制。

在測試導航與反饋的同時，和在沒有任何視覺提示的情況下，應該對你的應用大概是一個什麼樣子有所認識。出現問題就修復優化它們，最終就會開發出一個更易用可達的Android程序。

# 開發輔助服務

編寫:[KOST - 原文: http://developer.android.com/training/accessibility/service.html](http://developer.android.com/training/accessibility/service.html)

本課程將教您：

1. 創建可達性服務(Accessibility Service)
2. 配置可達性服務(Accessibility Service)
3. 響應可達性事件(AccessibilityEvents)
4. 從View層級中提取更多信息

Accessibility Service是Android系統框架提供給安裝在設備上應用的一個可選的導航反饋特性。Accessibility Service可以替代應用與用戶交流反饋，比如將文本轉化為語音提示，或是用戶的手指懸停在屏幕上一個較重要的區域時的觸摸反饋等。本課程將教您如何創建一個Accessibility Service，同時處理來自應用的信息，並將這些信息反饋給用戶。

## 創建Accessibility Service

Accessibility Service可以綁定在一個正常的應用中，或者是單獨的一個Android項目都可以。創建一個Accessibility Service的步驟與創建普通Service的步驟相似，在你的項目中創建一個繼承於[AccessibilityService](#)的類：

```
package com.example.android.apis.accessibility;

import android.accessibilityservice.AccessibilityService;

public class MyAccessibilityService extends AccessibilityService {
    ...
    @Override
    public void onAccessibilityEvent(AccessibilityEvent event) {
    }

    @Override
    public void onInterrupt() {
    }

    ...
}
```

與其他Service類似，你必須在manifest文件當中聲明這個Service。記得標明它監聽處理了 `android.accessibilityservice` 事件，以便Service在其他應用產生[AccessibilityEvent](#)的時候被調用。

```
<application ...>
    ...
    <service android:name=".MyAccessibilityService">
        <intent-filter>
            <action android:name="android.accessibilityservice.AccessibilityService" />
        </intent-filter>
        ...
    </service>
    ...
</application>
```

如果你為這個Service創建了一個新項目，且僅僅是一個Service而不準備做成一個應用，那麼你就可以移除啓動

的Activity(一般為>MainActivity.java)，同樣也記得在manifest中將這個Activity聲明移除。

## 配置Accessibility Service

設置Accessibility Service的配置變量會告訴系統如何讓Service運行與何時運行。你希望響應哪種類型的事件？Service是否對所有的應用有效還是對部分指定包名的應用有效？使用哪些不同類型的反饋？

你有兩種設置這些變量屬性的方法，一種向下兼容的辦法是通過代碼來進行設定，使用 `setServiceInfo` (`android.accessibilityservice.AccessibilityServiceInfo`)。你需要重寫 `(override) onServiceConnected()` 方法，並在這裏進行Service的配置。

```
@Override  
public void onServiceConnected() {  
    // Set the type of events that this service wants to listen to. Others  
    // won't be passed to this service.  
    info.eventTypes = AccessibilityEvent.TYPE_VIEW_CLICKED |  
        AccessibilityEvent.TYPE_VIEW_FOCUSED;  
  
    // If you only want this service to work with specific applications, set their  
    // package names here. Otherwise, when the service is activated, it will listen  
    // to events from all applications.  
    info.packageNames = new String[]  
        {"com.example.android.myFirstApp", "com.example.android.mySecondApp"};  
  
    // Set the type of feedback your service will provide.  
    info.feedbackType = AccessibilityServiceInfo.FEEDBACK_SPOKEN;  
  
    // Default services are invoked only if no package-specific ones are present  
    // for the type of AccessibilityEvent generated. This service *is*  
    // application-specific, so the flag isn't necessary. If this was a  
    // general-purpose service, it would be worth considering setting the  
    // DEFAULT flag.  
  
    // info.flags = AccessibilityServiceInfo.DEFAULT;  
  
    info.notificationTimeout = 100;  
  
    this.setServiceInfo(info);  
}
```

在Android 4.0之後，就用另一種方式來設置了：通過設置XML文件來進行配置。一些特性的選項比如 `canRetrieveWindowContent` 僅僅可以在XML可以配置。對於上面所示的相應的配置，利用XML配置如下：

```
<accessibility-service  
    android:accessibilityEventTypes="typeViewClicked|typeViewFocused"  
    android:packageNames="com.example.android.myFirstApp, com.example.android.mySecondApp"  
    android:accessibilityFeedbackType="feedbackSpoken"  
    android:notificationTimeout="100"  
    android:settingsActivity="com.example.android.apis.accessibility.TestBackActivity"  
    android:canRetrieveWindowContent="true"  
/>
```

如果你確定是通過XML進行配置，那麼請確保在manifest文件中通過`< meta-data >`標籤指定這個配置文件。假設此配置文件存放的地址為：`res/xml/serviceconfig.xml`，那麼標籤應該如下：

```
<service android:name=".MyAccessibilityService">  
    <intent-filter>  
        <action android:name="android.accessibilityservice.AccessibilityService" />  
    </intent-filter>
```

```
<meta-data android:name="android.accessibilityservice"
    android:resource="@xml/serviceconfig" />
</service>
```

## 響應Accessibility Event

現在你的Service已經配置好並可以監聽Accessibility Event了，來寫一些響應這些事件的代碼吧！首先就是要重寫onAccessibilityEvent(AccessibilityEvent)方法，在這個方法中，使用getEventType()來確定事件的類型，使用getContentDescription()來提取產生事件的View的相關的文本標籤。

```
@Override
public void onAccessibilityEvent(AccessibilityEvent event) {
    final int eventType = event.getEventType();
    String eventText = null;
    switch(eventType) {
        case AccessibilityEvent.TYPE_VIEW_CLICKED:
            eventText = "Focused: ";
            break;
        case AccessibilityEvent.TYPE_VIEW_FOCUSED:
            eventText = "Focused: ";
            break;
    }

    eventText = eventText + event.getContentDescription();

    // Do something nifty with this text, like speak the composed string
    // back to the user.
    speakToUser(eventText);
    ...
}
```

## 從View層級中提取更多信息

這一步並不是必要步驟，但是卻非常有用。Android 4.0版本中增加了一個新特性，就是能夠用AccessibilityService來遍歷View層級，並從產生Accessibility事件的組件與它的父子組件中提取必要的信息。為了實現這個目的，你需要在XML文件中進行如下的配置：

```
android:canRetrieveWindowContent="true"
```

一旦完成，使用getSource()獲取一個AccessibilityNodeInfo對象，如果觸發事件的窗口是活動窗口，該調用只返回一個對象，如果不是，它將返回null，做出相應的反響。下面的示例是一個代碼片段，當它接收到一個事件時，執行以下步驟：

1. 立即獲取到產生這個事件的Parent
2. 在這個Parent中尋找文本標籤或勾選框
3. 如果找到，創建一個文本內容來反饋給用戶，提示內容和是否已勾選。
4. 如果當遍歷View的時候某處返回了null值，那麼就直接結束這個方法。

```
// Alternative onAccessibilityEvent, that uses AccessibilityNodeInfo

@Override
public void onAccessibilityEvent(AccessibilityEvent event) {

    AccessibilityNodeInfo source = event.getSource();
    if (source == null) {
        return;
    }
```

```
// Grab the parent of the view that fired the event.
AccessibilityNodeInfo rowNode = getListItemNodeInfo(source);
if (rowNode == null) {
    return;
}

// Using this parent, get references to both child nodes, the label and the checkbox.
AccessibilityNodeInfo labelNode = rowNode.getChild(0);
if (labelNode == null) {
    rowNode.recycle();
    return;
}

AccessibilityNodeInfo completeNode = rowNode.getChild(1);
if (completeNode == null) {
    rowNode.recycle();
    return;
}

// Determine what the task is and whether or not it's complete, based on
// the text inside the label, and the state of the check-box.
if (rowNode.getChildCount() < 2 || !rowNode.getChild(1).isCheckable()) {
    rowNode.recycle();
    return;
}

CharSequence taskLabel = labelNode.getText();
final boolean isComplete = completeNode.isChecked();
String completeStr = null;

if (isComplete) {
    completeStr = getString(R.string.checked);
} else {
    completeStr = getString(R.string.not_checked);
}
String reportStr = taskLabel + completeStr;
speakToUser(reportStr);
}
```

現在你已經實現了一個完整可運行的Accessibility Service。嘗試着調整它與用戶的交互方式吧！比如添加語音引擎，或者添加震動來提供觸覺上的反饋都是不錯的選擇！

# 管理系統UI

編寫:KOST - 原文:<http://developer.android.com/training/system-ui/index.html>

**System Bar**是用來展示通知、表現設備狀態和完成設備導航的屏幕區域。通常上來說，系統欄(System bar)包括狀態欄和導航欄(Figure 1)，他們一般都是與程序同時顯示在屏幕上的。而照片、視頻等這類沉浸式的應用可以臨時弱化系統欄圖標來創造一個更加專注的體驗環境，甚至可以完全隱藏系統Bar。

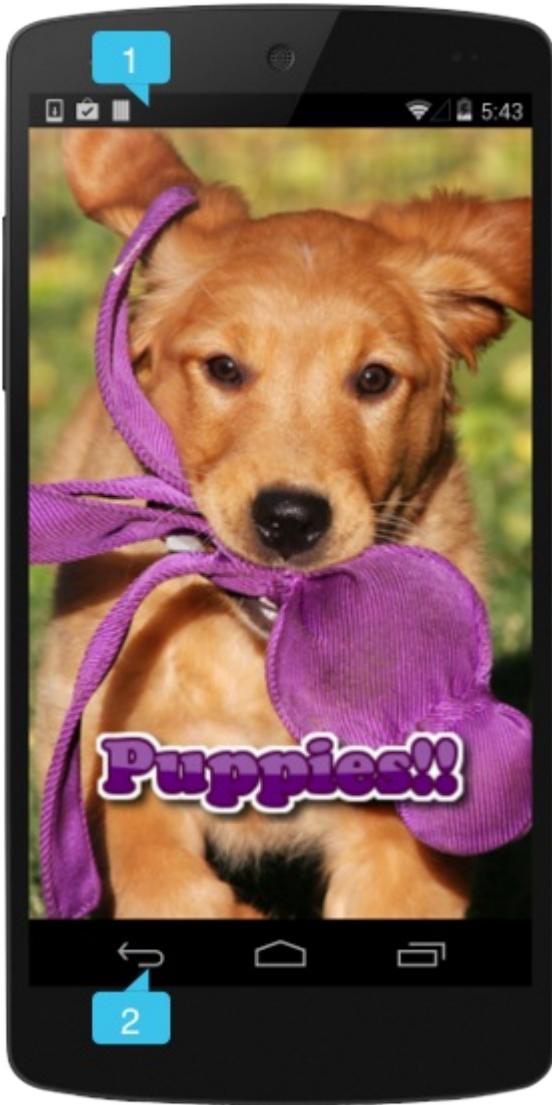


Figure 1. System bars，包含[1]狀態欄，和[2]導航欄。

如果你對Android Design Guide很熟悉，你應該已經知道遵照標準的Android UI Guideline與遵循模式來設計App的重要性。在你修改系統欄之前，你應該仔細的考慮一下用戶的需求與預期，因為它們是操作設備和觀察設備狀態的常規途徑。

這節課描述瞭如何在不同版本的Android上隱藏或淡化系統欄，來營造一個沉浸式的用戶體驗，同時做到快速的訪問與操作系統欄。

## Sample

ImmersiveMode - <http://developer.android.com/samples/ImmersionMode/index.html>

## Lessons

---

- [淡化系統欄](#)

學習如何淡化和隱藏狀態欄與導航欄。

- [隱藏狀態欄](#)

學習如何在不同版本的Android上隱藏狀態欄。

- [隱藏導航欄](#)

學習如何隱藏導航欄。

- [全屏沉浸式應用](#)

學習如何在你的App中創建沉浸模式。

- [響應UI可見性的變化](#)

學習如何註冊一個監聽器來監聽系統UI可見性的變化，以便於相應的調整App的UI。

# 淡化系統Bar

編寫:[KOST](#) - 原文:<http://developer.android.com/training/system-ui/dim.html>

本課程將向你講解如何在Android 4.0(API level 14)與更高的的系統版本上淡化系統欄(System bar,狀態欄與導航欄)。早期版本的Android沒有提供一個自帶的方法來淡化系統欄。

當你使用這個方法的時候，內容區域並不會發生大小的變化，只是系統欄的圖標會收起來。一旦用戶觸摸狀態欄或者是導航欄的時候，這兩個系統欄就又都會完全顯示（無透明度）。這種方法的優勢是系統欄仍然可見，但是它們的細節被隱藏掉了，因此可以在不犧牲快捷訪問系統欄的情況下創建一個沉浸式的體驗。

這節課將教您

1. 淡化狀態欄和導航欄
2. 顯示狀態欄和導航欄

同時您應該閱讀

- [Action Bar API 指南](#)
- [Android Design Guide](#)

## 淡化狀態欄和系統欄

如果要淡化狀態和通知欄，在版本為4.0以上的Android系統上，你可以像如下使用 `SYSTEM_UI_FLAG_LOW_PROFILE` 這個標籤。

```
// This example uses decor view, but you can use any visible view.  
View decorView = getActivity().getWindow().getDecorView();  
int uiOptions = View.SYSTEM_UI_FLAG_LOW_PROFILE;  
decorView.setSystemUiVisibility(uiOptions);
```

一旦用戶觸摸到了狀態欄或者是系統欄，這個標籤就會被清除，使系統欄重新顯現（無透明度）。在標籤被清除的情況下，如果你想重新淡化系統欄就必須重新設定這個標籤。

圖1展示了一個圖庫中的圖片，界面的系統欄都已被淡化（需要注意的是圖庫應用完全隱藏狀態欄，而不是淡化它）；注意導航欄（圖片的右側）上變暗的白色的小點，他們代表了被隱藏的導航操作。



圖1.淡化的系統欄

圖2展示的是同一張圖片，系統欄處於顯示的狀態。

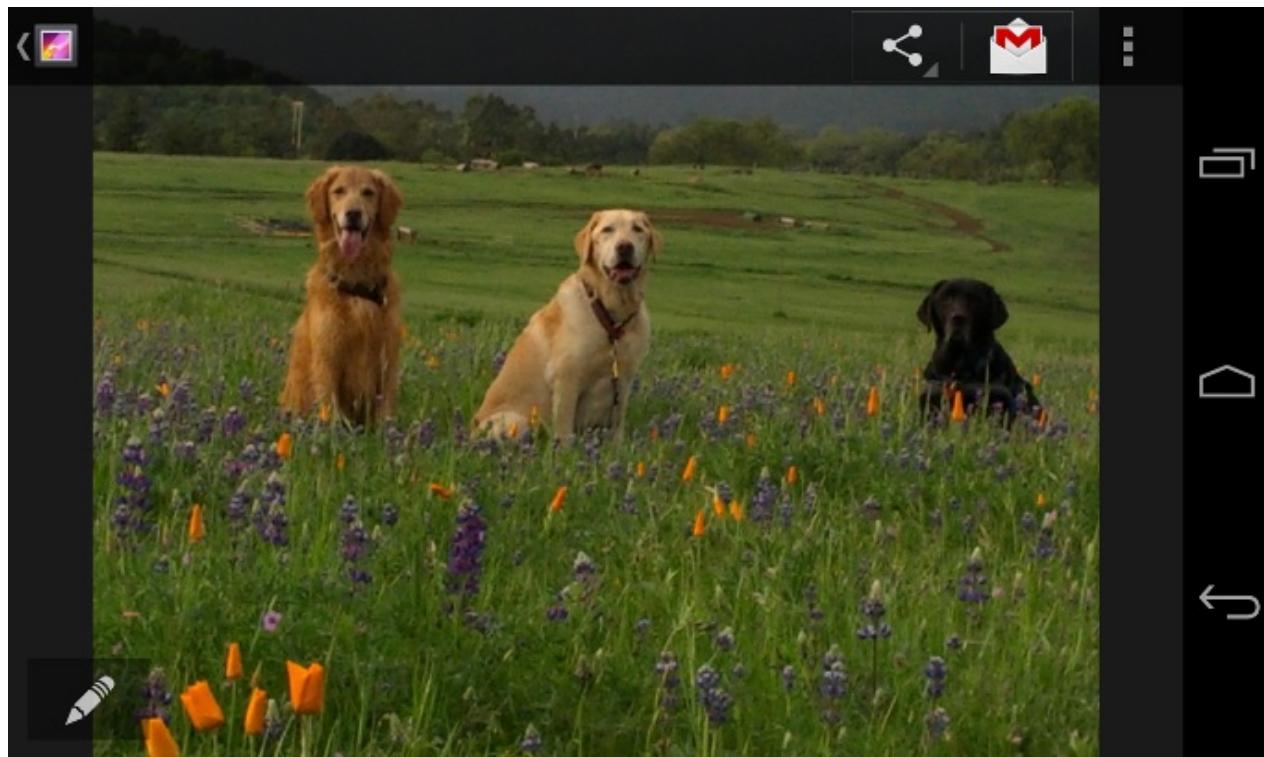


圖2.顯示的系統欄

## 顯示狀態欄與導航欄

如果你想動態的清除顯示標籤，你可以使用 `setSystemUiVisibility()` 方法：

```
View decorView = getActivity().getWindow().getDecorView();
// Calling setSystemUiVisibility() with a value of 0 clears
// all flags.
decorView.setSystemUiVisibility(0);
```

# 隱藏狀態欄

編寫:KOST - 原文:<http://developer.android.com/training/system-ui/status.html>

這節課將教您

1. 在4.0及以下版本中隱藏狀態欄
2. 在4.1及以上版本中隱藏狀態欄
3. 在4.4及以上版本中隱藏狀態欄
4. 讓內容顯示在狀態欄之後
5. 同步狀態欄與ActionBar的變化

同時您應該閱讀

- [ActionBar API 指南](#)
- [Android Design Guide](#)

本課程將教您如何在不同版本的Android下隱藏狀態欄。隱藏狀態欄（或者是導航欄）可以讓內容得到更多的展示空間，從而提供一個更加沉浸式的用戶體驗。

圖1展示了顯示狀態欄的界面



圖1. 顯示狀態欄.

圖2展示了隱藏狀態欄的界面。請注意，ActionBar這個時候也被隱藏了。請永遠不要在隱藏狀態欄的時候顯示ActionBar。



圖2. 隱藏狀態欄。

## 在4.0及以下版本中隱藏狀態欄

在Android 4.0及更低的版本中，你可以通過設置 `WindowManager` 來隱藏狀態欄。你可以動態的隱藏，也可以在你的 `manifest` 文件中設置 `Activity` 的主題。如果你的應用的狀態欄在運行過程中會一直隱藏，那麼推薦你使用改寫 `manifest` 設定主題的方法（嚴格上來講，即便設置了 `manifest` 你也可以動態的改變界面主題）。

```
<application
    ...
    android:theme="@android:style/Theme.Holo.NoActionBar.Fullscreen"
    ...
</application>
```

設置主題的優勢是：

- 易於維護，且不像動態設置標籤那樣容易出錯
- 有更流暢的UI轉換，因為在初始化你的 `Activity` 之前，系統已經得到了需要渲染UI的信息

另一方面我們可以選擇使用 `WindowManager` 來動態隱藏狀態欄。這個方法可以更簡單的在用戶與App進行交互式展示與隱藏狀態欄。

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // If the Android version is lower than Jellybean, use this call to hide
        // the status bar.
        if (Build.VERSION.SDK_INT < 16) {
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        }
        setContentView(R.layout.activity_main);
    }
    ...
}
```

當你設置 `WindowManager` 標籤之後（無論是通過 `Activity` 主題還是動態設置），這個標籤都會一直生效直到你清除它。

設置了 `FLAG_LAYOUT_IN_SCREEN` 之後，你可以擁有與啓用 `FLAG_FULLSCREEN` 後相同的屏幕區域。這個方法防止了狀態欄隱藏

和展示的時候內容區域的大小變化。

## 在4.1及以上版本中隱藏狀態欄

在Android 4.1(API level 16)以及更高的版本中，你可以使用`setSystemUiVisibility()`來進行動態隱藏。`setSystemUiVisibility()` 在View層面設置了UI的標籤，然後這些設置被整合到了Window層面。`setSystemUiVisibility()` 紿了你一個比設置`WindowManager` 標籤更加粒度化的操作。下面這段代碼隱藏了狀態欄：

```
View decorView = getWindow().getDecorView();
// Hide the status bar.
int uiOptions = View.SYSTEM_UI_FLAG_FULLSCREEN;
decorView.setSystemUiVisibility(uiOptions);
// Remember that you should never show the action bar if the
// status bar is hidden, so hide that too if necessary.
ActionBar actionBar = getActionBar();
actionBar.hide();
```

注意以下幾點：

- 一旦UI標籤被清除(比如跳轉到另一個Activity)，如果你還想隱藏狀態欄你就必須再次設定它。詳細可以看第五節如何監聽並響應UI可見性的變化。
- 在不同的地方設置UI標籤是有所區別的。如果你在Activity的`onCreate()`方法中隱藏系統欄，當用戶按下home鍵系統欄就會重新顯示。當用戶再重新打開Activity的時候，`onCreate()`不會被調用，所以系統欄還會保持可見。如果你想讓在不同Activity之間切換時，系統UI保持不變，你需要在`onResume()`與`onWindowFocusChanged()`裏設定UI標籤。
- `setSystemUiVisibility()`僅僅在被調用的View顯示的時候纔會生效。
- 當從View導航到別的地方時，用`setSystemUiVisibility()`設置的標籤會被清除。

## 讓內容顯示在狀態欄之後

在Android 4.1及以上版本，你可以將應用的內容顯示在狀態欄之後，這樣當狀態欄顯示與隱藏的時候，內容區域的大小就不會發生變化。要做到這個效果，我們需要用到`SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN` 這個標誌。同時，你也有可能需要`SYSTEM_UI_FLAG_LAYOUT_STABLE` 這個標誌來幫助你的應用維持一個穩定的佈局。

當使用這種方法的時候，你就需要來確保應用中特定區域不會被系統欄掩蓋（比如地圖應用中一些自帶的操作區域）。如果被覆蓋了，應用可能就會無法使用。在大多數的情況下，你可以在佈局文件中添加`android:fitsSystemWindows` 標籤，設置它為`true`。它會調整父`ViewGroup`使它留出特定區域給系統欄，對於大多數應用這種方法就足夠了。

在一些情況下，你可能需要修改默認的padding大小來獲取合適的佈局。為了控制內容區域的佈局相對系統欄（它佔據了一個叫做“內容嵌入”`content insets` 的區域）的位置，你可以重寫`fitSystemWindows(Rect insets)` 方法。當窗口的內容嵌入區域發生變化時，`fitSystemWindows()` 方法會被view的`hierarchy`調用，讓View做出相應的調整適應。重寫這個方法你就可以按你的意願處理嵌入區域與應用的佈局。

## 同步狀態欄與Action Bar的變化

在Android 4.1及以上的版本，為了防止在Action Bar隱藏和顯示的時候佈局發生變化，你可以使用Action Bar的`overlay`模式。在Overlay模式中，Activity的佈局佔據了所有可能的空間，好像Action Bar不存在一樣，系統會在佈局的上方繪製Action Bar。雖然這會遮蓋住上方的一些佈局，但是當Action Bar顯示或者隱藏的時候，系統就不需要重新改變佈局區域的大小，使之無縫的變化。

要啓用Action Bar的overlay模式，你需要創建一個繼承自Action Bar主題的自定義主題，  
將 `android:windowActionBarOverlay` 屬性設置為true。要瞭解詳細信息，請參考[添加Action Bar](#)課程中的Action Bar的覆蓋層疊。

設置 `SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN` 來讓你的activity使用的屏幕區域與設置 `SYSTEM_UI_FLAG_FULLSCREEN` 時的區域相同。  
當你需要隱藏系統UI時，使用 `SYSTEM_UI_FLAG_FULLSCREEN`。這個操作也同時隱藏了Action Bar（因為 `windowActionBarOverlay="true"`），當同時顯示與隱藏ActionBar與狀態欄的時候，使用一個動畫來讓他們相互協調。

# 隱藏導航欄

編寫:KOST - 原文:<http://developer.android.com/training/system-ui/navigation.html>

這節課將教您

1. 在4.0及以上版本中隱藏導航欄
2. 讓內容顯示在導航欄之後

本節課程將教您如何對導航欄進行隱藏，這個特性是Android 4.0（）版本中引入的。

即便本小節僅關注如何隱藏導航欄，但是在實際的開發中，你最好讓狀態欄與導航欄同時消失。在保證導航欄易於再次訪問的情況下，隱藏導航欄與狀態欄使內容區域佔據了整個顯示空間，因此可以提供一個更加沉浸式的用戶體驗。



圖1. 導航欄.

## 在4.0及以上版本中隱藏導航欄

你可以在Android 4.0及以上版本，使用 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` 標誌來隱藏導航欄。這段代碼同時隱藏了導航欄和系統欄：

```
View decorView = getWindow().getDecorView();
// Hide both the navigation bar and the status bar.
// SYSTEM_UI_FLAG_FULLSCREEN is only available on Android 4.1 and higher, but as
// a general rule, you should design your app to hide the status bar whenever you
// hide the navigation bar.
int uiOptions = View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
    | View.SYSTEM_UI_FLAG_FULLSCREEN;
decorView.setSystemUiVisibility(uiOptions);
```

注意以下幾點

- 使用這個方法時，觸摸屏幕的任何一個區域都會使導航欄（與狀態欄）重新顯示。用戶的交互會使這個標籤 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` 被清除。
- 一旦這個標籤被清除了，如果你想再次隱藏導航欄，你就需要重新對這個標籤進行設定。在下一節響應UI可見性的變化中，將詳細講解應用監聽系統UI變化來做出相應的調整操作。
- 在不同的地方設置UI標籤是有所區別的。如果你在Activity的`onCreate()`方法中隱藏系統欄，當用戶按下home鍵系統欄就會重新顯示。當用戶再重新打開activity的時候，`onCreate()`不會被調用，所以系統欄還會保持可見。如果你想讓在不同Activity之間切換時，系統UI保持不變，你需要在`onReasume()`與`onWindowFocusChaned()`裏設定UI標籤。

- `setSystemUiVisibility()`僅僅在被調用的View顯示的時候纔會生效。
- 當從View導航到別的地方時，用`setSystemUiVisibility()`設置的標籤會被清除。

## 2)讓內容顯示在導航欄之後

---

在Android 4.1與更高的版本中，你可以讓應用的內容顯示在導航欄的後面，這樣當導航欄展示或隱藏的時候內容區域就不會發生佈局大小的變化。可以使用 `SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION` 標籤來做到這個效果。同時，你也有可能需要 `SYSTEM_UI_FLAG_LAYOUT_STABLE` 這個標籤來幫助你的應用維持一個穩定的佈局。

當你使用這種方法的時候，就需要你來確保應用中特定區域不會被系統欄掩蓋。更詳細的信息可以瀏覽[隱藏狀態欄一節](#)。

# 全屏沉浸式應用

編寫:KOST - 原文:<http://developer.android.com/training/system-ui/immersive.html>

這節課將教您

1. 選擇一種沉浸方式
2. 使用非粘性沉浸模式
3. 使用粘性沉浸模式

Adnroid 4.4(API level 19)中引入為 `setSystemUiVisibility()` 引入了一個新標籤 `SYSTEM_UI_FLAG_IMMERSIVE`，它可以讓應用進入真正的全屏模式。當這個標籤與 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` 和 `SYSTEM_UI_FLAG_FULLSCREEN` 一起使用的時候，導航欄和狀態欄就會隱藏，讓你的應用可以接受屏幕上任何地方的觸摸事件。

當沉浸式全屏模式啓用的時候，你的Activity會繼續接受各類的觸摸事件。用戶可以通過在邊緣區域向內滑動來讓系統欄重新顯示。這個操作清空了 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` (和 `SYSTEM_UI_FLAG_FULLSCREEN`，如果有的話)兩個標籤，因此系統欄重新變得可見。如果設置了的話，這個操作同時也觸發了 `View.OnSystemUiVisibilityChangeListener`。然而，如果你想讓系統欄在一段時間後自動隱藏的話，你應該使用 `SYSTEM_UI_FLAG_IMMERSIVE_STICKY` 標籤。請注意，帶有'sticky'的標籤不會觸發任何的監聽器，因為在這個模式下展示的系統欄是處於暫時(transient)的狀態。

圖1展示了各種不同的“沉浸式”狀態



圖1. 沉浸模式狀態.

在上圖中：

1. 非沉浸模式 —— 展示了應用進入沉浸模式之前的狀態。也展示了設置 `IMMERSIVE` 標籤後用戶滑動展示系統欄的狀態。用戶滑動後，`SYSTEM_UI_FLAG_HIDE_NAVIGATION` 和 `SYSTEM_UI_FLAG_FULLSCREEN` 就會被清除，系統欄就會重新顯示並保持可見。請注意，最好的實踐方式就是讓所有的UI控件的變化與系統欄的顯示隱藏保持同步，這樣可以減少屏幕顯示所處的狀態，同時提供了更無縫平滑的用戶體驗。因此所有的UI控件跟隨系統欄一同顯示。一旦應用進入了沉浸模式，相應的UI控件也跟隨着系統欄一同隱藏。為了確保UI的可見性與系統欄保持一致，我們需要一個監聽器 `View.OnSystemUiVisibilityChangeListener` 來監聽系統欄的變化。這在下一節中將詳細講解。

2. 提示氣泡——第一次進入沉浸模式時，系統將會顯示一個提示氣泡，提示用戶如何再讓系統欄顯示出來。

Note：如果為了測試你想強制顯示提示氣泡，你可以先將應用設為沉浸模式，然後按下電源鍵進入鎖屏模式，並在5秒中之後打開屏幕。

3. 沉浸模式——這張圖展示了隱藏了系統欄和其他UI控件的狀態。你可以設置 `IMMERSIVE` 和 `IMMERSIVE_STICKY` 來進入這個狀態。

4. 粘性標籤——這就是你設置了 `IMMERSIVE_STICKY` 標籤時的UI狀態，用戶會向內滑動以展示系統欄。半透明的系統欄會臨時的進行顯示，一段時間後自動隱藏。滑動的操作並不會清空任何標籤，也不會觸發系統UI可見性的監聽器，因為暫時顯示的導航欄並不被認為是一種可見性狀態的變化。

Note：`immersive` 類的標籤只有在與 `SYSTEM_UI_FLAG_HIDE_NAVIGATION`，`SYSTEM_UI_FLAG_FULLSCREEN` 中一個或兩個一起使用的時候纔會生效。你可以只使用其中的一個，但是一般情況下你需要同時隱藏狀態欄和導航欄以達到沉浸的效果。

## 選擇一種沉浸方式

`SYSTEM_UI_FLAG_IMMERSIVE` 與 `SYSTEM_UI_FLAG_IMMERSIVE_STICKY` 都提供了沉浸式的體驗，但是在上面的描述中，他們是不一樣的，下面講解一下什麼時候該用哪一種標籤。

- 如果你在寫一款圖書瀏覽器、新聞雜誌閱讀器，請將 `IMMERSIVE` 標籤與 `SYSTEM_UI_FLAG_FULLSCREEN`，`SYSTEM_UI_FLAG_HIDE_NAVIGATION` 一起使用。因為用戶可能會經常訪問Action Bar和一些UI控件，又不希望在翻頁的時候有其他的東西進行干擾。`IMMERSIVE` 在該種情況下就是個很好的選擇。
- 如果你在打造一款真正的沉浸式應用，而且你希望屏幕邊緣的區域也可以與用戶進行交互，並且用戶也不會經常訪問系統UI。這個時候就要將 `IMMERSIVE_STICKY` 和 `SYSTEM_UI_FLAG_FULLSCREEN`，`SYSTEM_UI_FLAG_HIDE_NAVIGATION` 兩個標籤一起使用。比如做一款遊戲或者繪圖應用就很合適。
- 如果你在打造一款視頻播放器，並且需要少量的用戶交互操作。你可能就需要之前版本的一些方法了（從Android 4.0開始）。對於這種應用，簡單的使用 `SYSTEM_UI_FLAG_FULLSCREEN` 與 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` 就足夠了，不需要使用 `immersive` 標籤。

## 使用非粘性沉浸模式

當你使用 `SYSTEM_UI_FLAG_IMMERSIVE` 標籤的時候，它是基於其他設置過的標籤 (`SYSTEM_UI_FLAG_HIDE_NAVIGATION` 和 `SYSTEM_UI_FLAG_FULLSCREEN`)來隱藏系統欄的。當用戶向內滑動，系統欄重新顯示並保持可見。

用其他的UI標籤(如 `SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION` 和 `SYSTEM_UI_FLAG_LAYOUT_STABLE`)來防止系統欄隱藏時內容區域大小發生變化是一種很不錯的方法。你也需要確保Action Bar和其他系統UI控件同時進行隱藏。下面這段代碼展示瞭如何在不改變內容區域大小的情況下，隱藏與顯示狀態欄和導航欄。

```
// This snippet hides the system bars.
private void hideSystemUI() {
    // Set the IMMERSIVE flag.
    // Set the content to appear under the system bars so that the content
    // doesn't resize when the system bars hide and show.
    mDecorView.setSystemUiVisibility(
        View.SYSTEM_UI_FLAG_LAYOUT_STABLE
        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION // hide nav bar
        | View.SYSTEM_UI_FLAG_FULLSCREEN // hide status bar
        | View.SYSTEM_UI_FLAG_IMMERSIVE);
}
```

```
// This snippet shows the system bars. It does this by removing all the flags
// except for the ones that make the content appear under the system bars.
private void showSystemUI() {
    mDecorView.setSystemUiVisibility(
        View.SYSTEM_UI_FLAG_LAYOUT_STABLE
        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
}
```

你可能同時也希望在如下的幾種情況下使用 `IMMERSIVE` 標籤來提供更好的用戶體驗：

- 註冊一個監聽器來監聽系統UI的變化。
- 實現 `onWindowFocusChanged()` 函數。如果窗口獲取了焦點，你可能需要對系統欄進行隱藏。如果窗口失去了焦點，比如說彈出了一個對話框或菜單，你可能需要取消那些將要在 `Handler.postDelayed()` 或其他地方的隱藏操作。
- 實現一個 `GestureDetector`，它監聽了 `onSingleTapUp(MotionEvent)` 事件。可以使用戶點擊內容區域來切換系統欄的顯示狀態。單純的點擊監聽可能不是最好的解決方案，因為當用戶在屏幕上拖動手指的時候（假設點擊的內容佔據了整個屏幕），這個事件也會被觸發。

更多關於此話題的討論，可以觀看這個視頻 [DevBytes: Android 4.4 Immersive Mode](#)

## 使用粘性沉浸模式

當使用了 `SYSTEM_UI_FLAG_IMMERSIVE_STICKY` 標籤的時候，向內滑動的操作會讓系統欄臨時顯示，並處於半透明的狀態。此時沒有標籤會被清除，系統UI可見性監聽器也不會被觸發。如果用戶沒有進行操作，系統欄會在一段時間內自動隱藏。

圖2展示了當使用 `IMMERSIVE_STICKY` 標籤時，半透明的系統欄展示與又隱藏的狀態。



圖2. 自動隱藏系統欄。

下面是一段實現代碼。一旦窗口獲取了焦點，只要簡單的設置 `IMMERSIVE_STICKY` 與上面討論過的其他標籤即可。

```
@Override  
public void onWindowFocusChanged(boolean hasFocus) {  
    super.onWindowFocusChanged(hasFocus);  
    if (hasFocus) {  
        decorView.setSystemUiVisibility(  
            View.SYSTEM_UI_FLAG_LAYOUT_STABLE  
            | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION  
            | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN  
            | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION  
            | View.SYSTEM_UI_FLAG_FULLSCREEN  
            | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY);}  
}
```

Notes：如果你想實現 `IMMERSIVE_STICKY` 的自動隱藏效果，同時也需要展示你自己的UI控件。你只需要使用 `IMMERSIVE` 與 `Handler.postDelayed()` 或其他類似的東西，讓它幾秒後重新進入沉浸模式即可。

# 響應UI可見性的變化

編寫:KOST - 原文:<http://developer.android.com/training/system-ui/visibility.html>

本節課將教你如果註冊監聽器來監聽系統UI可見性的變化。這個方法在將系統欄與你自己的UI控件進行同步操作時很有用。

## 註冊監聽器

為了獲取系統UI可見性變化的通知，我們需要對View註冊 `View.OnSystemUiVisibilityChangeListener` 監聽器。通常上來說，這個View是用來控制導航的可見性的。

例如你可以添加如下代碼在onCreate中

```
View decorView = getWindow().getDecorView();
decorView.setOnSystemUiVisibilityChangeListener(
    (new View.OnSystemUiVisibilityChangeListener() {
        @Override
        public void onSystemUiVisibilityChange(int visibility) {
            // Note that system bars will only be "visible" if none of the
            // LOW_PROFILE, HIDE_NAVIGATION, or FULLSCREEN flags are set.
            if ((visibility & View.SYSTEM_UI_FLAG_FULLSCREEN) == 0) {
                // TODO: The system bars are visible. Make any desired
                // adjustments to your UI, such as showing the action bar or
                // other navigational controls.
            } else {
                // TODO: The system bars are NOT visible. Make any desired
                // adjustments to your UI, such as hiding the action bar or
                // other navigational controls.
            }
        }
    }));

```

保持系統欄和UI同步是一種很好的實踐方式，比如當狀態欄顯示或隱藏的時候進行ActionBar的顯示和隱藏等等。

# 創建使用Material Design的應用

編寫: allenlsy - 原文: <https://developer.android.com/training/material/index.html>

Material Design 是一個全面的關於視覺，動作和交互的指南，實現跨平臺的設計。要在你的 Android 應用中使用 Material Design，你需要遵從 [Material Design 規格文檔](#)，來使用 Android 5.0 中新添加的組件和功能。

本課會通過以下方面教你如何創建 Material Design 設計的應用：

- Material Design 主題
- 用於卡片和列表的小組件
- 定義Shadows與Clipping視圖
- 矢量 drawable
- 自定義動畫

本課還將告訴你在使用 Material Design 時如何兼容 Android 5.0 (API level 21) 之前的版本。

## 課程

### 開始使用Material Design

學習如何升級應用，使用 Material Design 特性

### 使用 Material Design 主題

學習如何使用 Material Design 主題

### 用於卡片和列表的小組件

學習如何創建列表和卡片視圖，使得應用和其他系統組件風格統一

### 定義Shadows與Clipping視圖

學習如何設置 evaluation 來自定義陰影，以及創建 Clipping 視圖

### 使用 Drawables

學習如何創建矢量 Drawable 以及如何給 drawable 資源着色

### 自定義動畫

學習如何為視圖和 Activity 切換創建自定義動畫

### 維護兼容性

學習如何兼容 Android 5.0 以下的版本

# 開始使用Material Design

編寫: allenlsy - 原文: <https://developer.android.com/training/material/get-started.html>

要創建一個 Material Design 應用：

1. 學習 [Material Design 規格標準](#)
2. 應用 Material Design 主題
3. 創建符合 Material Design 的 Layout 文件
4. 定義視圖的 elevation 值來修改陰影
5. 使用系統組件來創建列表和卡片
6. 自定義動畫

## 維護向下兼容性

你可以添加 Material Design 特性，同時保持對 Android 5.0 之前版本的兼容。更多信息，請參見[維護兼容性章節](#)。

## 使用 Material Design 更新現有應用

要更新現有應用，使其使用 Material Design，你需要翻新你的 layout 文件來遵從 Material Design 標準，並確保其包含了正確的元素高度，觸摸反饋和動畫。

## 使用 Material Design 創建新的應用

如果你要創建使用 Material Design 的新的應用，Material Design 指南提供了一套跨平臺統一的設計。請遵從指南，使用新功能來進行 Android 應用的設計和開發。

# 應用 Material 主題

要在應用中使用 Material 主題，需要定義一個繼承於 `android:Theme.Material` 的 style 文件：

```
<!-- res/values/styles.xml -->
<resources>
    <!-- your theme inherits from the material theme -->
    <style name="AppTheme" parent="android:Theme.Material">
        <!-- theme customizations -->
    </style>
</resources>
```

Material 主題提供了更新後的系統組件，使你可以設置調色板和在觸摸和 [Activity](#) 切換時使用默認的動畫。更多信息，請參見 [Material 主題](#) 章節。

## 設計你的 Layouts

另外，要應用自定義的 Material 主題，你的 layout 應該要符合 [Material 設計規範](#)。在設計 Layout 時，尤其要注意一下方面：

- 基準線網格
- Keyline

- 間隙
- 觸摸目標的大小
- Layout 結構

## 定義視圖的 Elevation

視圖可以投射陰影，`elevation` 值決定了陰影的大小和繪製順序。要設定 `elevation` 值，請使用 `android:elevation` 屬性：

```
<TextView  
    android:id="@+id/my_textview"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/next"  
    android:background="@color/white"  
    android:elevation="5dp" />
```

新的 `translationZ` 屬性使得你可以設計臨時變更 `elevation` 的動畫。`elevation` 變化在做觸摸反饋時很有用。

更多信息，請參見定義陰影和 Clipping 視圖章節。

## 創建列表和卡片

[RecyclerView](#) 是一個植入性更強的 `ListView`，它支持不同的 layout 類型，並可以提升性能。[CardView](#) 使得你可以在卡片內顯示一部分內容，並且和其他應用保持外觀一致。以下是一段樣例代碼展示如何在 layout 中添加 `CardView`

```
<android.support.v7.widget.CardView  
    android:id="@+id/card_view"  
    android:layout_width="200dp"  
    android:layout_height="200dp"  
    card_view:cardCornerRadius="3dp">  
...  
</android.support.v7.widget.CardView>
```

更多信息，請參見列表和卡片章節。

## 自定義動畫

Android 5.0 (API level 21) 包含了新的創建自定義動畫 API。比如，你可以在 `activity` 中定義進入和退出 `activity` 時的動畫。

```
public class MyActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // enable transitions  
        getWindow().requestFeature(Window.FEATURE_CONTENT_TRANSITIONS);  
        setContentView(R.layout.activity_my);  
    }  
  
    public void onSomeButtonClicked(View view) {  
        getWindow().setExitTransition(new Explode());  
        Intent intent = new Intent(this, MyOtherActivity.class);
```

```
        startActivity(intent,
                    ActivityOptions
                        .makeSceneTransitionAnimation(this).toBundle());
    }
}
```

當你從當前 `activity` 進入另一個 `activity` 時，退出切換動畫會被調用。

想學習更多新的動畫 API，參見[自定義動畫章節](#)。

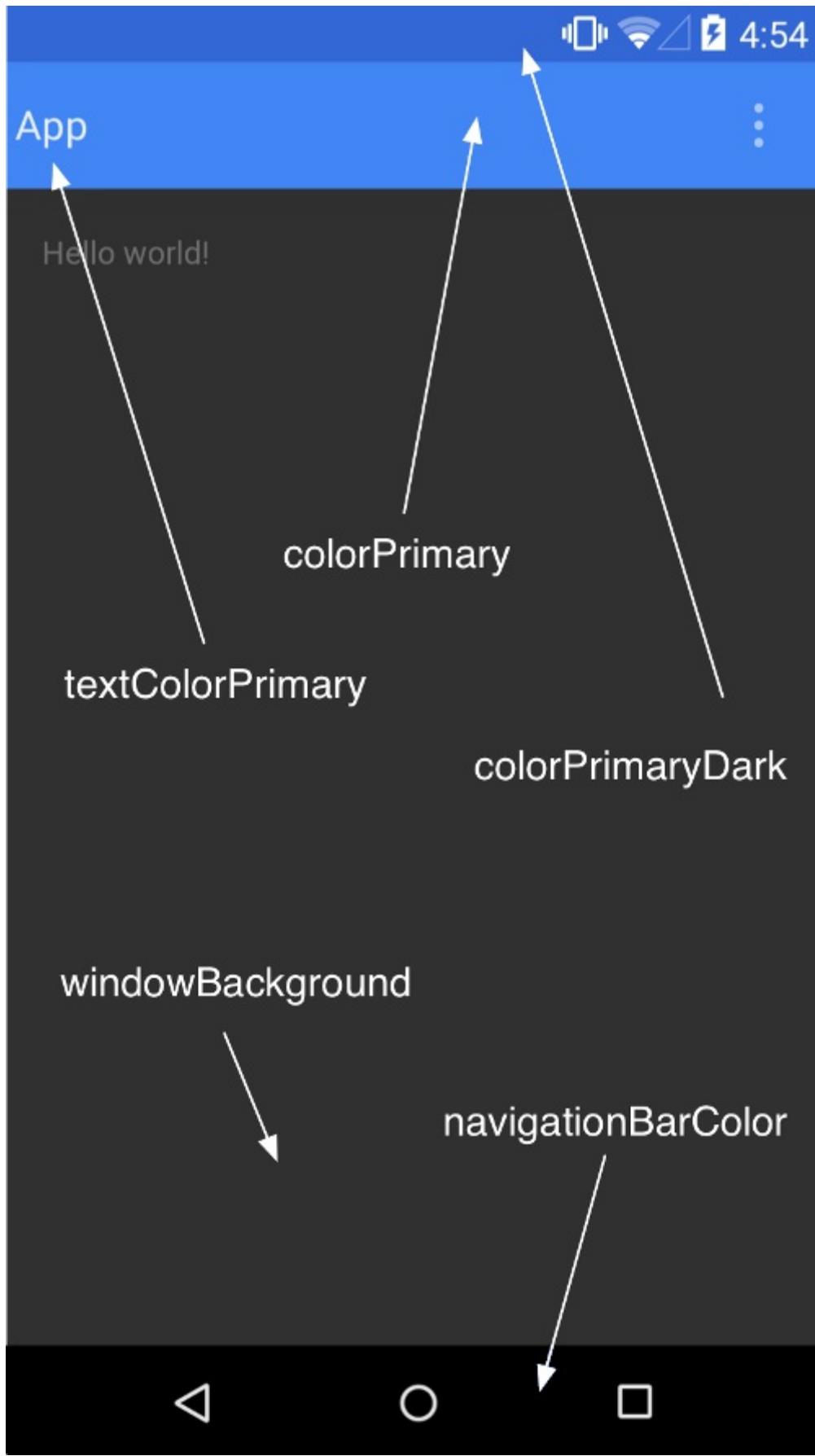
# 使用Material的主題

編寫: allenlsy - 原文: <https://developer.android.com/training/material/theme.html>

新的 Material 主題提供：

- 系統組件，用於設定調色板
- 系統組件的觸摸反饋動畫
- *Activity* 切換動畫

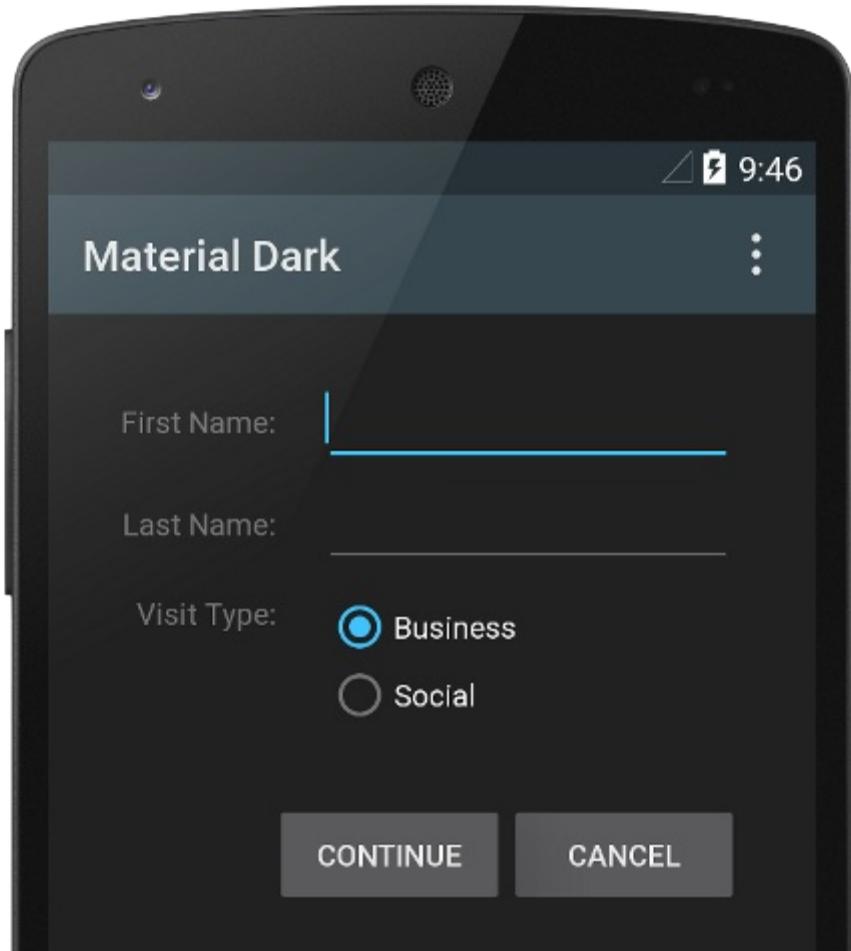
你可以根據你的品牌特徵修改調色板，從而自定義 Material 主題。你可以通過主題屬性調整 action bar 和狀態欄的顏色，就像下圖一樣：

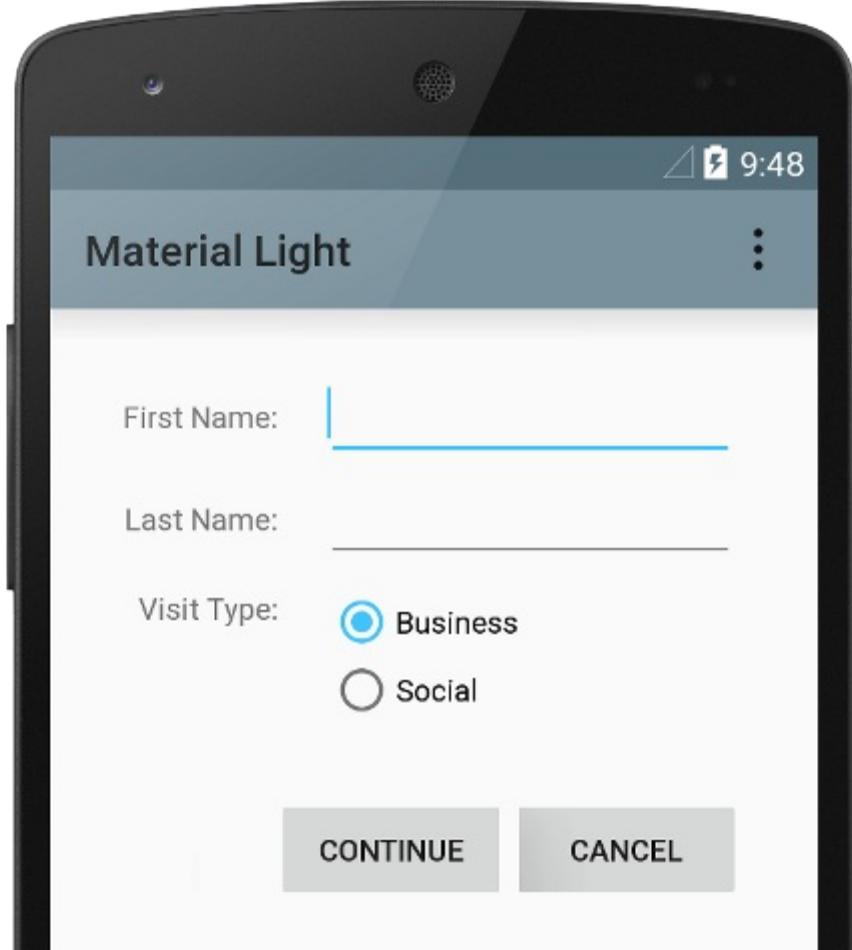


系統組件擁有新的設計和觸摸反饋動畫。你可以自定義調色板，反饋動畫和 [Activity](#) 切換動畫。

Material 主題被定義在：

- @android:style/Theme.Material (暗色版本)
- @android:style/Theme.Material.Light (亮色版本)
- @android:style/Theme.Material.Light.DarkActionBar





想知道可用的 Material style 的列表，可以在 API 文檔中參見 [R.style](#).

Note: Material 主題只支持 Android 5.0 (API level 21) 及以上版本。[v7 Support 庫](#)提供了一些組件的 Material Design 樣式，也支持自定義調色板。更多信息，請參見維護兼容性章節。

## 自定義調色板

在根據自己的品牌自定義調色板時，你需要在繼承 material 主題時定義 theme 屬性。

```
<resources>
    <!-- inherit from the material theme -->
    <style name="AppTheme" parent="android:Theme.Material">
        <!-- Main theme colors -->
        <!-- your app branding color for the app bar -->
        <item name="android:colorPrimary">@color/primary</item>
        <!-- darker variant for the status bar and contextual app bars -->
        <item name="android:colorPrimaryDark">@color/primary_dark</item>
        <!-- theme UI controls like checkboxes and text fields -->
        <item name="android:colorAccent">@color/accent</item>
    </style>
</resources>
```

## 自定義狀態欄

Material 主題使得你很容易自定義狀態欄，你可以設定適合自己品牌的顏色，並提供足夠的對比度，以顯示白色的狀態圖標。設置狀態欄顏色時，要在繼承 Material 主題時設定 `android:statsBarColor` 屬性。默認情況

下，`android:statusBarColor` 會繼承 `android:colorPrimaryDark` 的值。

你也可以在狀態欄的背景上繪畫。比如，你想讓位於照片之上的狀態欄透明，並保留一點深色漸變以確保白色圖標可見。這樣的話，設定 `android:statusBarColor` 屬性為 `@android:color/transparent` 並調整窗口的 Flag 標記。你也可以用 `Window.setStatusBarColor()` 來實現動畫或淡入淡出。

**Note:** 狀態欄必須隨時保持和 primary toolbar (即頂部ActionBar，譯者注) 的界線清晰。除了一種情況，即在狀態欄後面顯示圖片或媒體內容時之外，你都要用漸變色來確保前臺圖標仍然可見。

當你自定義導航欄和狀態欄時，要麼兩者都透明，要麼只修改狀態欄。其他情況下，導航欄應該保持黑色。

## 主題單獨視圖

---

XML layout 中的元素可以定義 `android:theme` 屬性，用於引用主題資源。這個屬性修改了自己和子元素的主題，對於要修改局部顏色主題的情況十分有用。

# 創建Lists與Cards

編寫: allenlsy - 原文: <https://developer.android.com/training/material/lists-cards.html>

要在應用中創建複雜的列表和使用 Material Design 的卡片列表，你可以使用 `RecyclerView` 和 `CardView`。

## 創建列表

`RecyclerView` 組件是一個更高級和伸縮性更強的 `ListView`。這個組件是一個顯示大量數據的容器，通過維護有限量的 View，來達到滾動時的高效。當你的數據集在運行過程中會根據用戶行為或網絡事件更新時，應該使用 `RecyclerView`。

`RecyclerView` 通過以下方式簡化顯示流程，並操作大量數據：

- 使用 Layout manager 來定位元素
- 為常用操作定義默認動畫，比如添加或移除元素

你也可以為 `RecyclerView` 自定義 Layout manager 和動畫。

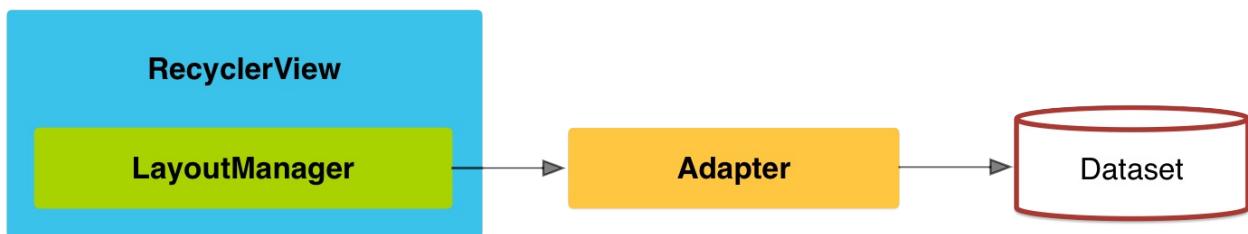
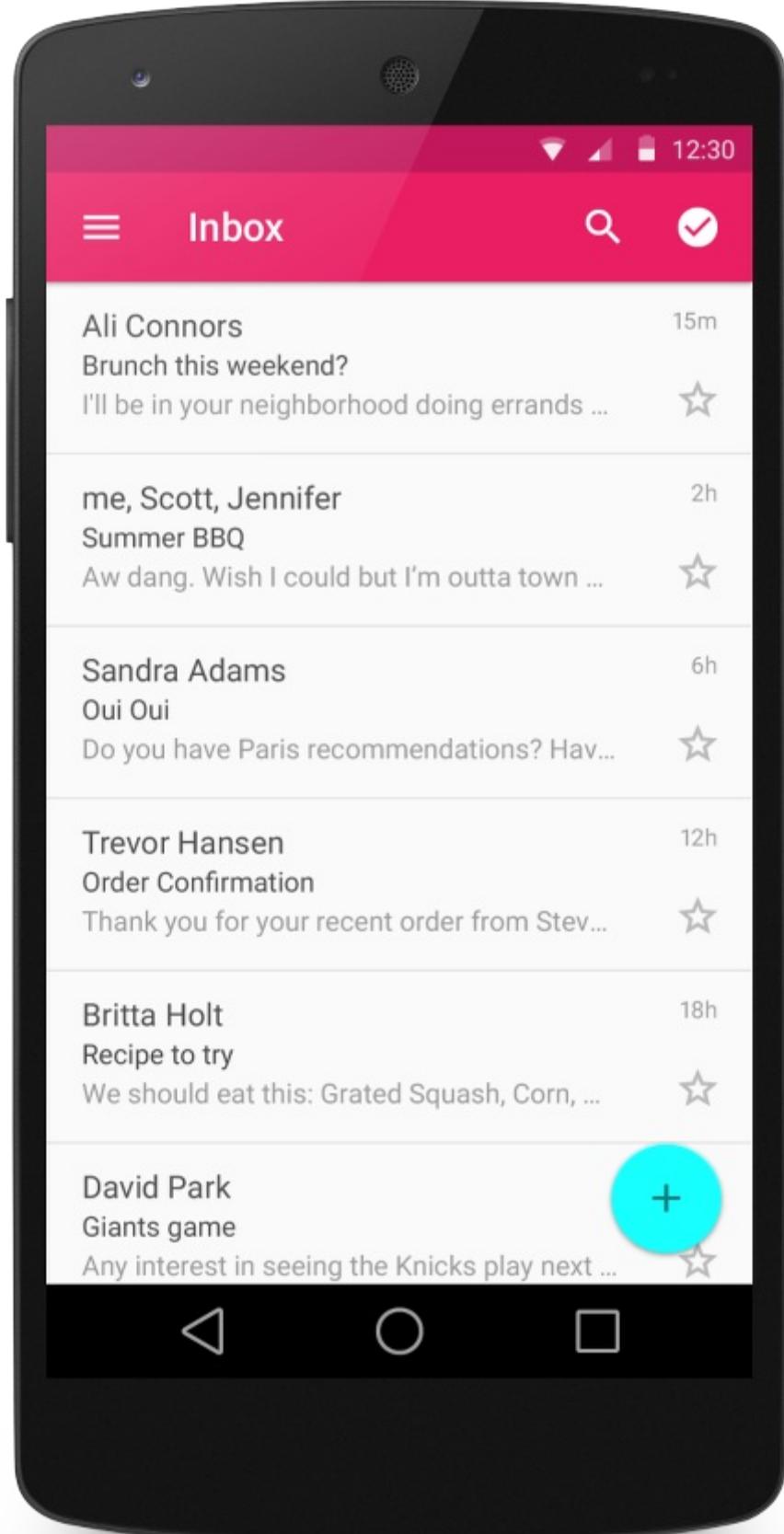


圖1. The `RecyclerView` widget.

要使用 `RecyclerView` 組件，你需要定義一個 adapter 和 layout manager。創建 adapter，要繼承 `RecyclerView.Adapter` 類。實現類的細節取決於你的數據集和視圖類型。更多信息，請看以下樣例。



Layout manager把元素視圖放在 `RecyclerView`，並決定什麼時候重用不可見的元素視圖。要重用（或回收）視圖時，layout manager 會讓 adapter 用另外的元素內容替換視圖內的內容。回收 View 這個方法能提高性能，因為它避免了創建不必要的view對象，或執行昂貴的 `findViewById()` 查找。

`RecyclerView` 提供以下內建的 layout manager:

- `LinearLayoutManager` 用於顯示橫向或縱向的滾動列表
- `GridLayoutManager` 用於顯示方格元素
- `StaggeredGridLayoutManager` 在 `staggered` 方格中顯示元素

創建一個自定義的 `layout manager`，要繼承於 `RecyclerView.LayoutManager` 類

## 動畫

添加和刪除元素的動畫在 `RecyclerView` 中是默認被啓用的。要自定義動畫，你需要繼承 `RecyclerView.ItemAnimator` 類，使用 `RecyclerView.setItemAnimator()` 方法。

## 例子

以下代碼示例瞭如何添加 `RecyclerView` 到一個 `Layout`：

```
<!-- A RecyclerView with some commonly used attributes -->
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

添加 `RecyclerView` 組件到 `Layout` 之後，獲得一個到 `RecyclerView` 的對象，連接它到 `Layout manager`，再附上 `adapter` 用於數據顯示：

```
public class MyActivity extends Activity {
    private RecyclerView mRecyclerview;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager mLayoutManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity);
        mRecyclerview = (RecyclerView) findViewById(R.id.my_recycler_view);

        // use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        mRecyclerview.setHasFixedSize(true);

        // use a linear layout manager
        mLayoutManager = new LinearLayoutManager(this);
        mRecyclerview.setLayoutManager(mLayoutManager);

        // specify an adapter (see also next example)
        mAdapter = new MyAdapter(myDataset);
        mRecyclerview.setAdapter(mAdapter);
    }
    ...
}
```

`Adapter` 支持獲取數據集元素，創建元素的視圖，並可以將新元素的內容去替代不可見元素視圖中的內容。以下代碼展示了一個簡單的實現，其中的數據集包含了一個字符串數組，數據元素用 `TextView` 顯示：

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {
    private String[] mDataset;
```

```

// Provide a reference to the views for each data item
// Complex data items may need more than one view per item, and
// you provide access to all the views for a data item in a view holder
public static class ViewHolder extends RecyclerView.ViewHolder {
    // each data item is just a string in this case
    public TextView mTextView;
    public ViewHolder(TextView v) {
        super(v);
        mTextView = v;
    }
}

// Provide a suitable constructor (depends on the kind of dataset)
public MyAdapter(String[] myDataset) {
    mDataset = myDataset;
}

// Create new views (invoked by the layout manager)
@Override
public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                int viewType) {
    // create a new view
    View v = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.my_text_view, parent, false);
    // set the view's size, margins, paddings and layout parameters
    ...
    ViewHolder vh = new ViewHolder(v);
    return vh;
}

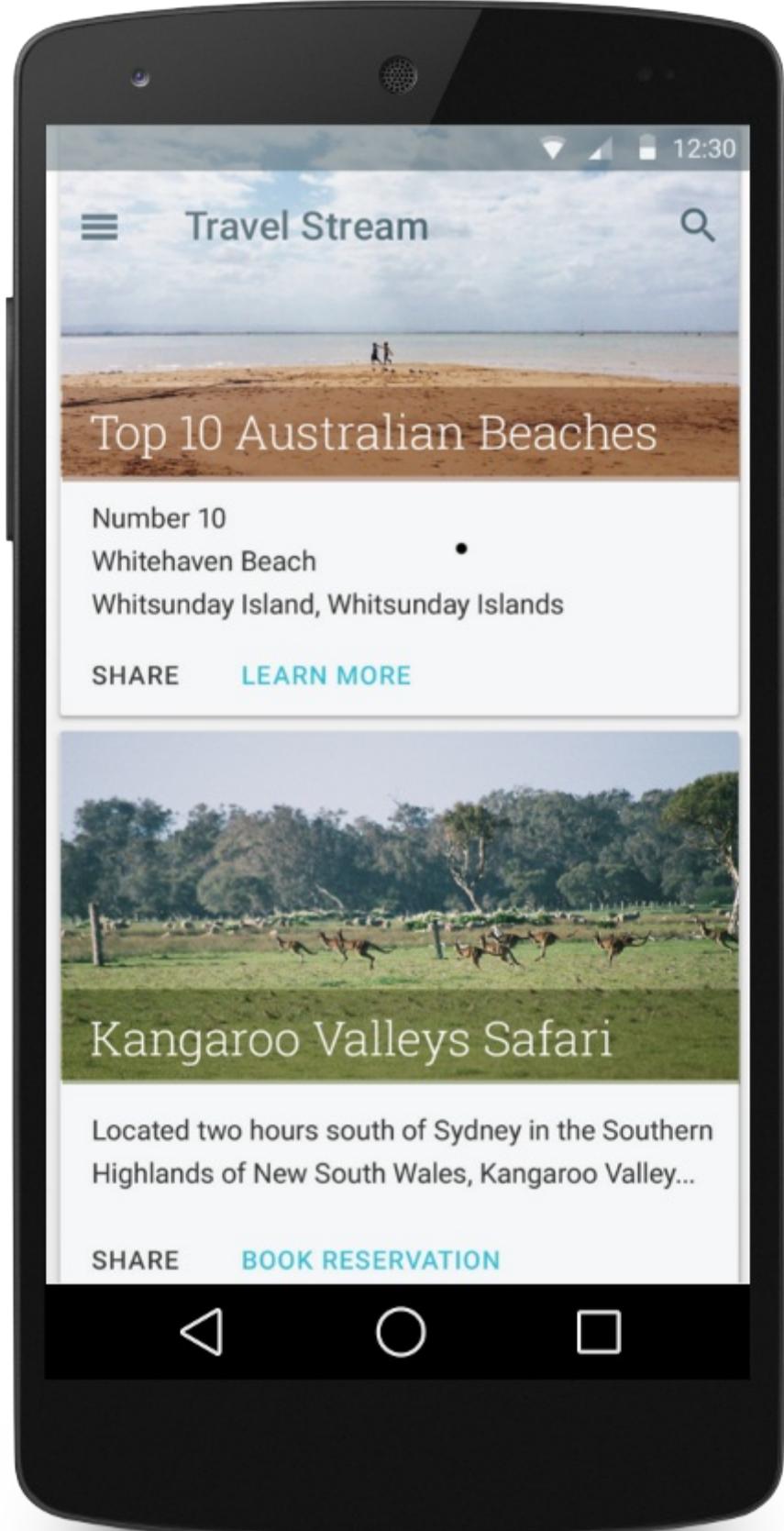
// Replace the contents of a view (invoked by the layout manager)
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    // - get element from your dataset at this position
    // - replace the contents of the view with that element
    holder.mTextView.setText(mDataset[position]);
}

// Return the size of your dataset (invoked by the layout manager)
@Override
public int getItemCount() {
    return mDataset.length;
}
}

```

## 創建卡片

---



CardView 繼承於 FrameLayout 類，它可以在卡片中顯示信息，並保持在不同平臺上擁有統一的風格。CardView 組件可以設定陰影和圓角。

要創建一個帶陰影的卡片，使用 `card_view:cardElevation` 屬性。CardView 使用了Android 5.0 (API level 21)中的真實高度值以及動態陰影效果，在 5.0 以下的版本中有編程實現陰影的備選方案。更多內容，請參見保持兼容性章節。

使用以下屬性來自定義CardView：

- 使用 `card_view:cardCornerRadius` 在layout中設置圓角
- 使用 `CardView.setRadius` 在代碼中設置圓角
- 使用 `card_view:cardBackgroundColor` 來設置背景顏色

以下代碼展示如何在layout中添加CardView：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    ...
    <!-- A CardView that contains a TextView -->
    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view"
        android:layout_gravity="center"
        android:layout_width="200dp"
        android:layout_height="200dp"
        card_view:cardCornerRadius="4dp">

        <TextView
            android:id="@+id/info_text"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </android.support.v7.widget.CardView>
</LinearLayout>
```

更多信息，參見CardView的API文檔。

## 添加依賴

RecyclerView和CardView都是v7 support 庫的一部分。要使用這兩個組件，在你的Gradle依賴中添加兩個模塊：

```
dependencies {
    ...
    compile 'com.android.support:cardview-v7:21.0.+'
    compile 'com.android.support:recyclerview-v7:21.0.+'
}
```

# 定義Shadows與Clipping視圖

編寫: allenlsy - 原文: <https://developer.android.com/training/material/shadows-clipping.html>

Material Design 引入了UI元素深度的概念。深度可以幫助用戶理解每個元素的不同重要性，讓用戶集中注意力做手頭的工作。

視圖的elevation，用 Z 屬性來表示，它決定了陰影的大小：更大的 Z 值可以投射出更大更柔軟的陰影。Z 值較大的視圖會遮蓋住Z值較小的視圖。不過，Z值大小不會影響視圖的大小。

陰影是由被投射視圖的上級視圖來完成繪製，因此他受上級視圖影響，附着在上級視圖上。

Elevation對於創建臨時上升這種動畫同樣很有用。

更多信息，請參見[3D空間中的對象](#)。

## 給視圖賦Elevation值

視圖的 Z 值有兩個組成部分：

- elevation: 靜態組成部分
- translation: 動態部分，用於動畫

$$Z = \text{elevation} + \text{translation}_Z$$

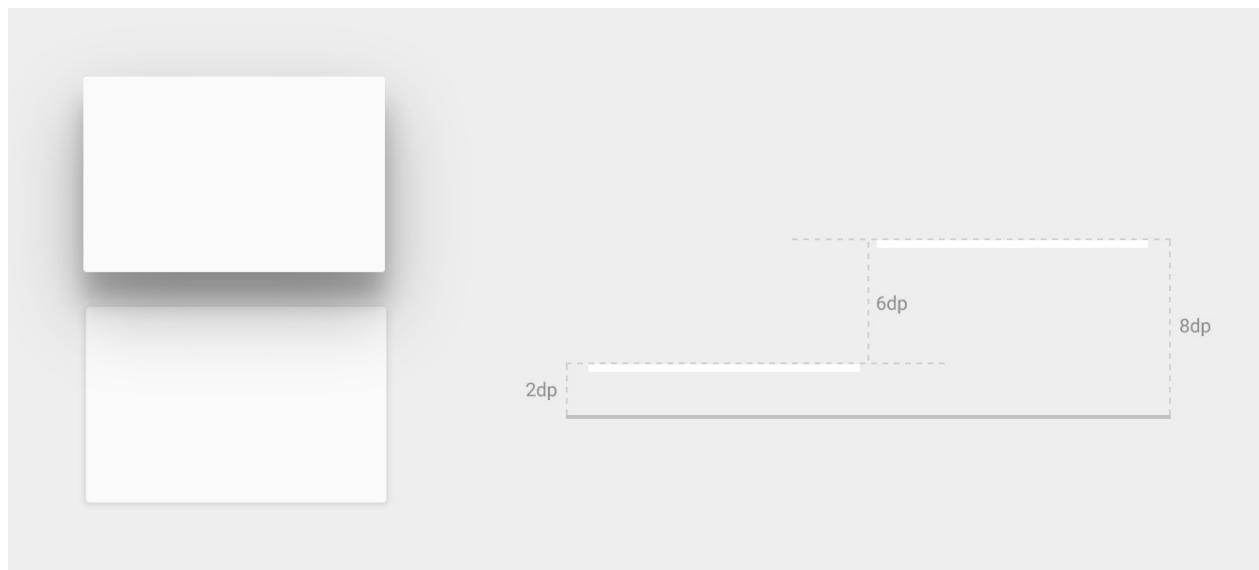


圖1 - 不同深度view的陰影。

在layout中設置視圖的elevation，要使用 `android:elevation` 屬性。要在Activity代碼中設置elevation，使用 `View.setElevation()` 方法。

要設置視圖的translation，使用 `View.setTranslationZ()` 方法。

新的 `ViewPropertyAnimator.z()` 和 `ViewPropertyAnimator.translationZ()` 方法使你可以很容易的實現elevation動畫。更多信息，請查看[ViewPropertyAnimator和屬性動畫開髮指南](#)。

你也可以使用 `StateListAnimator` 來聲明動畫。這非常適用於要通過狀態改變來觸發動畫的情況，比如當用戶按下按鈕。更多信息，請查看 [Animate View State Changes](#) (當視圖狀態變化的動畫，譯者注)。

Z值的計算單位是dp。

## 自定義視圖的陰影和輪廓

視圖背景的邊界決定了陰影的形狀。輪廓是一個圖形對象的外圓形狀，決定了觸摸反饋動畫的ripple區域。

假設以下是個視圖：

```
<TextView  
    android:id="@+id/myview"  
    ...  
    android:elevation="2dp"  
    android:background="@drawable/myrect" />
```

背景drawable定義為一個圓角的矩形：

```
<!-- res/drawable/myrect.xml -->  
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shape="rectangle">  
    <solid android:color="#42000000" />  
    <corners android:radius="5dp" />  
</shape>
```

這個視圖會投影出圓角，因為背景drawable可以決定視圖輪廓。如果提供一個自定義的輪廓，會覆蓋這個默認的陰影形狀。

以下方式可以自定義視圖的輪廓：

1. 繼承 `ViewOutlineProvider` 類
2. 覆寫 `getOutline()` 函數。
3. 用 `View.setOutlineProvider()` 方法來設定視圖的輪廓提供者。

使用 `Outline` 類的函數，你可以創建橢圓和帶圓角的矩形輪廓。視圖的輪廓提供者會從視圖的背景中獲取輪廓。如果不希望讓視圖投射陰影，你可以設置輪廓提供者為 null。

## Clipping 視圖

Clipping 視圖（附着視圖，譯者注）使你輕鬆的改變視圖的形狀。你可以為了一致性而附着視圖，也可以是為了當用戶輸入信息時，改變視圖的形狀。你可以通過 `View.setClipToOutline()` 將視圖附着給一個輪廓，或使用 `android:clipToOutline` 屬性。只有矩形、圓形和圓角矩形輪廓支持附着功能，你可以通過 `outline.canClip()` 方法來檢查是否支持附着。

把視圖附着給drawable的形狀，要將這個drawable設置為視圖的背景，並調用 `View.setClipToOutline()` 方法。

附着視圖是一個昂貴的操作，所以不要對附着過的形狀進行動畫。要實現這個效果，使用 [Reveal Effect](#) 動畫

# 使用Drawables

編寫: allenlsy - 原文: <https://developer.android.com/training/material/drawables.html>

## 使用Drawable

以下這些drawable的功能，能幫助你在應用中實現Material Design：

- Drawable染色
- 提取主色調
- 矢量Drawable

本課教你如何在應用中使用這些特性：

### 給 Drawable 資源染色

使用 Android 5.0 (API level 21)以上版本，你可以使用alpha mask（透明度圖層，譯者注）給位圖和nine patches圖片染色。你可以用顏色Resource或者主題屬性來獲取顏色（比如，`?android:attr/colorPrimary`）。通常，你只需要創建一次這些顏色asset，便可以在主題中自動匹配這些顏色。

你可以用 `setTint()` 方法將一種染色方式應用到 `BitmapDrawable` 或者 `NinePatchDrawable` 對象。你也在layout中使用 `android:tint` 和 `android:initMode` 屬性設置染色的顏色和模式。

### 從圖片中提取主色調

Android Support Library v21及更高版本帶有 `Palatte` 類，可以讓你從圖片中提取主色調。這個類可以提取以下顏色：

- Vibrant: 亮色
- Vibrant dark: 深亮色
- Vibrant light: 淺亮色
- Muted: 暗色
- Muted dark: 深暗色
- Muted light: 淺暗色

提取這些顏色時，在你載入圖片的後臺線程中傳入一個Bitmap對象給 `Palette.generate()` 靜態方法。如果你不能使用那個線程，可以調用 `Palatte.generateAsync()` 方法，並提供一個listener。

你可以用`Palatte`類的一個getter方法從圖片獲取主色調，比如 `Palatte.getVibrantColor()`。

要使用 `Palatte` 類，在你的應用模塊的Gradle依賴中添加以下代碼：

```
dependencies {
    ...
    compile 'com.android.support:palette-v7:21.0.+'
}
```

更多信息，請參見`Palatte`類的API文檔。

# 創建矢量Drawable

在Android 5.0 (API level 21)以上版本中，你可以定義矢量drawable，用於無損的拉伸圖片。相對於一張普通圖片需要為每個不同屏幕密度的設備提供一個圖片來說，一個矢量圖片只需要一個asset文件。要創建矢量圖片，你可以在 `<vector>` XML元素中定義形狀。

以下代碼定義了一個心形：

```
<!-- res/drawable/heart.xml -->
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- intrinsic size of the drawable -->
    android:height="256dp"
    android:width="256dp"
    <!-- size of the virtual canvas -->
    android:viewportWidth="32"
    android:viewportHeight="32">

    <!-- draw a path -->
    <path android:fillColor="#8fff"
        android:pathData="M20.5,9.5
            C-1.955,0,-3.83,1.268,-4.5,3
            C-0.67,-1.732,-2.547,-3,-4.5,-3
            C8.957,9.5,7,11.432,7,14
            C0,3.53,3.793,6.257,9,11.5
            C5.207,-5.242,9,-7.97,9,-11.5
            C25,11.432,23.043,9.5,20.5,9.5z" />
</vector>
```

矢量圖片在Android中用[VectorDrawable](#)對象來表示。更多關於 `pathData` 語法的信息，請看[SVG Path](#)的文檔。更多關於矢量drawable動畫的信息，請參見[矢量drawable動畫](#)。

# 自定義動畫

編寫: allenlsy - 原文: <https://developer.android.com/training/material/animations.html>

Material Design中的動畫對用戶的動作進行反饋，並提供在整個交互過程中的視覺連續性。Material 主題為按鈕和Activity切換提供一些默認的動畫，Android 5.0 (API level 21) 及以上版本支持自定義這些動畫並創建新動畫：

- 觸摸反饋
- 圓形填充
- Activity 切換動畫
- 曲線形動作
- 視圖狀態變換

## 自定義觸摸反饋

Material Design中的觸摸反饋，是在用戶與UI元素交互時，提供視覺上的即時確認。按鈕的默認觸摸反饋動畫使用了新的 RippleDrawable 類，它在按鈕狀態變換時產生波紋效果。

大多數情況下，你需要在你的 XML 文件中設定視圖的背景來實現這個功能：

- ?android:attr/selectableItemBackground 用於有界Ripple動畫
- ?android:attr/selectableItemBackgroundBorderless 用於越出視圖邊界的動畫。它會被繪製在最近的切不是全屏的父視圖上。

Note : selectableItemBackgroundBorderless 是 API level 21 新加入的屬性

另外，你可以使用 ripple 元素在XML資源文件中定義一個 RippleDrawable 。

你可以給 RippleDrawable 賦予一個顏色。要改變默認的觸摸反饋顏色，使用主題的 android:colorControlHighlight 屬性。

更多信息，參見 RippleDrawable 類的API文檔。

## 使用填充效果 (Reveal Effect)

填充效果在UI元素出現或隱藏時，為用戶提供視覺連續性。 ViewAnimationUtils.createCircularReveal() 方法可以使用一個附着在視圖上的圓形，顯示或隱藏這個視圖。

要用此效果顯示一個原本不可見的視圖：

```
// previously invisible view
View myView = findViewById(R.id.my_view);

// get the center for the clipping circle
int cx = (myView.getLeft() + myView.getRight()) / 2;
int cy = (myView.getTop() + myView.getBottom()) / 2;

// get the final radius for the clipping circle
int finalRadius = myView.getWidth();

// create and start the animator for this view
// (the start radius is zero)
Animator anim =
    ViewAnimationUtils.createCircularReveal(myView, cx, cy, 0, finalRadius);
```

```
anim.start();
```

要用此效果隱藏一個原本可見的視圖：

```
// previously visible view
final View myView = findViewById(R.id.my_view);

// get the center for the clipping circle
int cx = (myView.getLeft() + myView.getRight()) / 2;
int cy = (myView.getTop() + myView.getBottom()) / 2;

// get the initial radius for the clipping circle
int initialRadius = myView.getWidth();

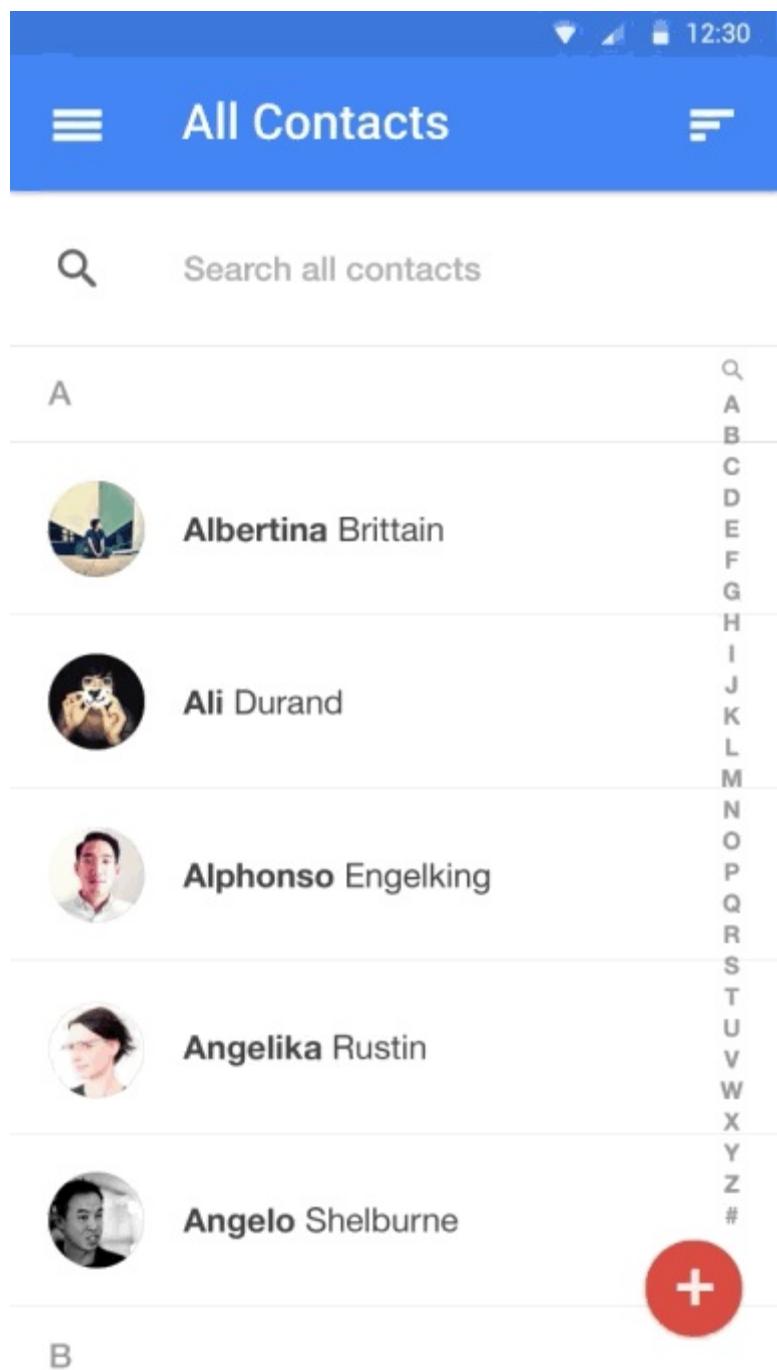
// create the animation (the final radius is zero)
Animator anim =
    ViewAnimationUtils.createCircularReveal(myView, cx, cy, initialRadius, 0);

// make the view invisible when the animation is done
anim.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        super.onAnimationEnd(animation);
        myView.setVisibility(View.INVISIBLE);
    }
});

// start the animation
anim.start();
```

## 自定義Activity切換效果

---



Material Design中的Activity切換，當不同Activity之間擁有共有元素，則可以通過不同狀態之間的動畫和形變提供視覺上的連續性。你可以為共有元素設定進入和退出Activity時的自定義動畫。

- 入場變換決定視圖如何入場。比如，在爆炸式入場變換中，視圖從場外飛到屏幕中央。
- 出場變換決定視圖如何退出。比如，在爆炸式出場變換中，視圖從屏幕中央飛出場外。
- 共有元素的變換決定一個共有視圖在兩個Activity之間如何變換。比如，如果兩個activity有同一張圖片，但是放在不同位置，以及擁有不同大小，變更圖片變換會流暢的把圖片移到相應位置，同時縮放圖片大小。

Android 5.0 (API level 21) 支持這些入場和退出變換：

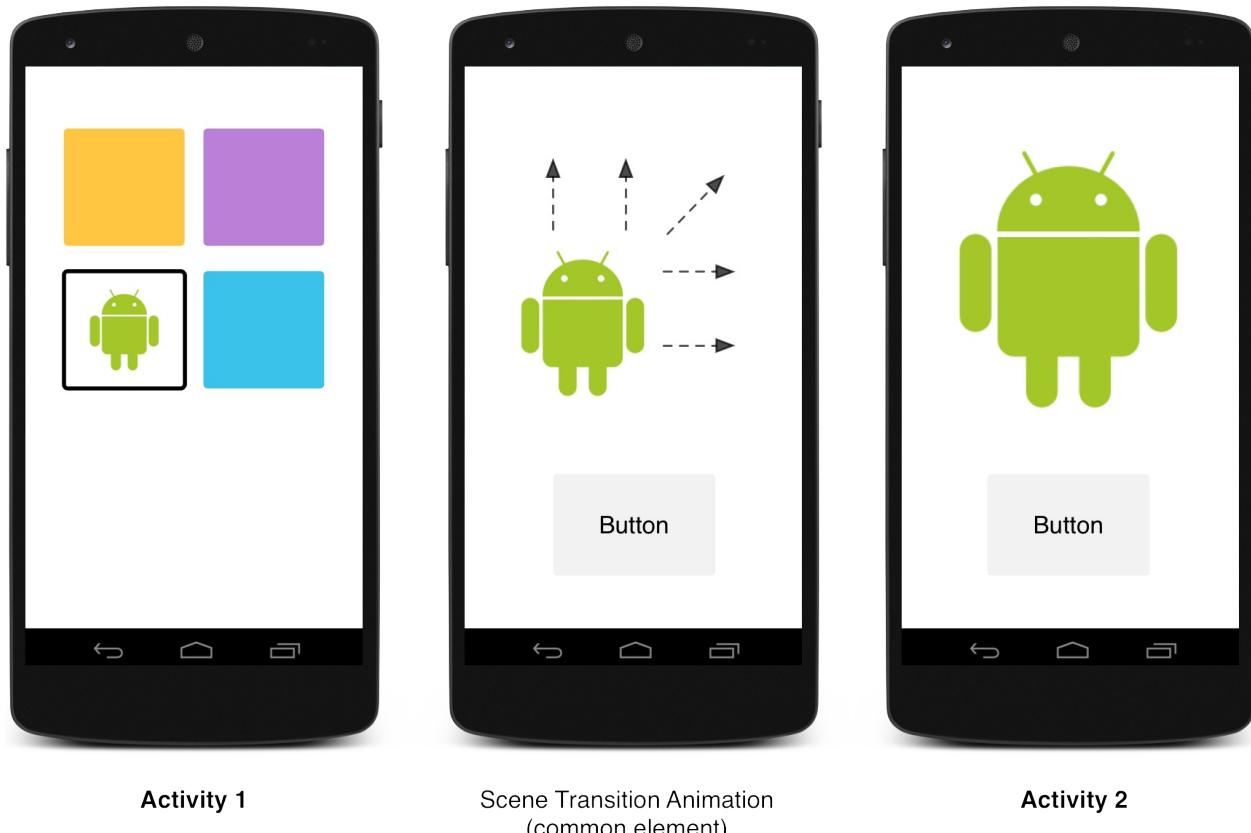
- 爆炸 - 把視圖移入或移出場景的中間
- 滑動 - 把視圖從場景邊緣移入或移出
- 淡入淡出 - 通過改變透明度添加或移除元素

任何繼承於 `visibility` 類的變換，都支持被用於入場或退出變換。更多信息，請參見 `Transition` 類的API文檔。

Android 5.0 (API level 21) 還支持這些共有元素變換效果：

- `changeBounds` - 對目標視圖的外邊界進行動畫
- `chagneClipBounds` - 對目標視圖的附着物的外邊界進行動畫
- `changeTransform` - 對目標視圖進行縮放和旋轉
- `changelImageTransform` - 對目標圖片進行縮放

當你在應用中進行activity 變換時，默認的淡入淡出效果會被用在進入和退出activity的過程中。



Activity 1

Scene Transition Animation  
(common element)

Activity 2

## 自定義切換

首先，當你繼承Material主題的style時，要通過 `android:windowContentTransitions` 屬性來開啓窗口內容變換功能。你也可以在style定義中聲明進入、退出和共有元素切換：

```
<style name="BaseAppTheme" parent="android:Theme.Material">
    <!-- enable window content transitions -->
    <item name="android:windowContentTransitions">true</item>

    <!-- specify enter and exit transitions -->
    <item name="android:windowEnterTransition">@transition/explode</item>
    <item name="android:windowExitTransition">@transition/explode</item>

    <!-- specify shared element transitions -->
    <item name="android:windowSharedElementEnterTransition">
        @transition/change_image_transform</item>
    <item name="android:windowSharedElementExitTransition">
        @transition/change_image_transform</item>
</style>
```

例子中的 `change_image_transform` 切換定義如下：

```
<!-- res/transition/change_image_transform.xml -->
<!-- (see also Shared Transitions below) -->
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android">
    <changeImageTransform/>
</transitionSet>
```

`changeImageTransform` 元素對應 `ChangeImageTransform` 類。更多信息，請參見 `Transition` 類的API文檔。

要在代碼中啓用窗口內容切換，調用 `Window.requestFeature()` 函數：

```
// inside your activity (if you did not enable transitions in your theme)
getWindow().requestFeature(Window.FEATURE_CONTENT_TRANSITIONS);

// set an exit transition
getWindow().setExitTransition(new Explode());
```

要聲明變換類型，就要在 `Transition` 對象上調用以下函數：

- `Window.setEnterTransition()`
- `Window.setExitTransition()`
- `Window.setSharedElementEnterTransition()`
- `Window.setSharedElementExitTransition()`

`setExitTransition()` 和 `setSharedElementExitTransition()` 函數為`activity`定義了退出變換效果。`setEnterTransition()` 和 `setSharedElementEnterTransition()` 函數定義了進入`activity`的變換效果。

要獲得切換的全部效果，你必須在出入的兩個`activity`中都開啓窗口內容切換。否則，調用的`activity`會使用退出效果，但是接着你會看到一個傳統的窗口切換（比如縮放或淡入淡出）。

要儘早開始入場切換，可以在被調用的`Activity`上使用 `Window.setAllowEnterTransitionOverlap()`。它可以使你擁有更戲劇性的入場切換。

## 使用切換啓動一個Activity

如果你開啓`Activity`入場和退出效果，那麼當你在用如下方法開始`Activity`時，切換效果會被應用：

```
startActivity(intent,
    ActivityOptions.makeSceneTransitionAnimation(this).toBundle());
```

如果你為第二個`Activity`設定了入場變換，變換也會在`activity`開始時被啓用。要在開始另一個`activity`時禁用變換，可以給`bundle`的選項提供一個 `null` 對象：

## 啓動一個擁有共用元素的Activity

要在兩個擁有共用元素的`activity`間進行切換動畫：

1. 在主題中開啓窗口內容切換
2. 在`style`中定義共有元素切換
3. 將切換定義為一個XML 資源文件
4. 使用 `android:transitionName` 屬性在兩個`layout`文件中給共有元素賦予同一個名字
5. 使用 `ActivityOptions.makeSceneTransitionAnimation()` 方法

```

// get the element that receives the click event
final View imgContainerView = findViewById(R.id.img_container);

// get the common element for the transition in this activity
final View androidRobotView = findViewById(R.id.image_small);

// define a click listener
imgContainerView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(this, Activity2.class);
        // create the transition animation - the images in the layouts
        // of both activities are defined with android:transitionName="robot"
        ActivityOptions options = ActivityOptions
            .makeSceneTransitionAnimation(this, androidRobotView, "robot");
        // start the new activity
        startActivity(intent, options.toBundle());
    }
});

```

對於用代碼編寫的共有動態視圖，使用 `View.setTransitionName()` 方法來在兩個activity中定義共有元素。

要在第二個activity結束時進行逆向的場景切換動畫，調用 `Activity.finishAfterTransition()` 方法，而不是 `Activity.finish()`。

## 開始一個擁有多個共有元素的Activity

要在擁有多個共有元素的activity之間使用變換動畫，就要用 `android:transitionName` 屬性在兩個layout中定義這個共有元素（或在兩個Activity中使用 `View.setTransitionName()` 方法），再創建 `ActivityOptions` 對象：

```

ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(this,
    Pair.create(view1, "agreedName1"),
    Pair.create(view2, "agreedName2"));

```

## 使用曲線動畫

Material Design中的動畫可以表示為基於時間插值和空間移動模式的曲線。在Android 5.0 (API level 21)以上版本中，你可以為動畫定義時間曲線和曲線動畫模式。

`PathInterpolator` 類是一個基於貝澤爾曲線或 `Path` 對象的新的插值方法。插值方法 是一個定義在  $1 \times 1$  正方形中的曲線函數圖像，其始末兩點分別在 $(0,0)$ 和  $(1,1)$ ，一個用構造函數定義的控制點。你也可以使用XML資源文件定義一個插值方法：

```

<pathInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
    android:controlX1="0.4"
    android:controlY1="0"
    android:controlX2="1"
    android:controlY2="1"/>

```

Material Design標準中，系統提供了三種基本的曲線：

- `@interpolator/fast_out_linear_in.xml`
- `@interpolator/fast_out_slow_in.xml`
- `@interpolator/linear_out_slow_in.xml`

你可以將一個 `PathInterpolator` 對象傳給 `Animator.setInterpolator()` 方法。

`ObjectAnimator` 類有一個新的構造函數，使你可以沿一條路徑使用多個屬性來在座標系中進行變換。比如，以下 `animator`（動畫器，譯者注）使用一個 `Path` 對象來改變一個試圖的X和Y屬性：

```
ObjectAnimator mAnimator;
mAnimator = ObjectAnimator.ofFloat(view, View.X, View.Y, path);
...
mAnimator.start();
```

## 基於視圖狀態改變的動畫

`StateListAnimator` 類是你可以定義在視圖狀態改變啓動的 `Animator`（動畫器，譯者注）。以下例子展示如何在 XML 文件中定義 `StateListAnimator`：

```
<!-- animate the translationZ property of a view when pressed -->
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <set>
            <objectAnimator android:propertyName="translationZ"
                android:duration="@android:integer/config_shortAnimTime"
                android:valueTo="2dp"
                android:valueType="floatType"/>
            <!-- you could have other objectAnimator elements
                here for "x" and "y", or other properties -->
        </set>
    </item>
    <item android:state_enabled="true"
        android:state_pressed="false"
        android:state_focused="true">
        <set>
            <objectAnimator android:propertyName="translationZ"
                android:duration="100"
                android:valueTo="0"
                android:valueType="floatType"/>
        </set>
    </item>
</selector>
```

要把視圖改變 `Animator` 關聯到一個視圖，就要在 XML 資源文件的 `selector` 元素上定義一個 `Animator`，並把此 `Animator` 賦值給視圖的 `android:stateListAnimator` 屬性。要想在 Java 代碼中將狀態列表 `Animator` 賦值給視圖，使用 `AnimationInflater.loadStateListAnimator()` 函數，並用 `View.setStateListAnimator()` 函數把 `Animator` 賦值給你的視圖。

當你的主題繼承於 Material Theme 的時候，`Button` 默認會有一個 Z 值動畫。為了避免 `Button` 的 Z 值動畫，設定它的 `android:stateListAnimator` 屬性為 `@null`。

`AnimatedStateListDrawable` 類使你可以創建一個在視圖狀態變化之間顯示動畫的 `drawable`。有一些 Android 5.0 系統組件默認已經使用了這些動畫。下面的例展示如何在 XML 資源文件中定義 `AnimatedStateListDrawable`：

```
<!-- res/drawable/myanimstatedrawable.xml -->
<animated-selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- provide a different drawable for each state-->
    <item android:id="@+id/pressed" android:drawable="@drawable/drawableP"
        android:state_pressed="true"/>
    <item android:id="@+id/focused" android:drawable="@drawable/drawableF"
        android:state_focused="true"/>
    <item android:id="@+id/default"
```

```
    android:drawable="@drawable/drawableD"/>

    <!-- specify a transition -->
    <transition android:fromId="@+id/default" android:toId="@+id/pressed">
        <animation-list>
            <item android:duration="15" android:drawable="@drawable/dt1"/>
            <item android:duration="15" android:drawable="@drawable/dt2"/>
            ...
        </animation-list>
    </transition>
    ...
</animated-selector>
```

## 動畫矢量 Drawables

矢量Drawable是可以無損縮放的。 AnimatedVectorDrawable 類是你可以操作矢量Drawable。

你通常在3個XML文件中定義動畫矢量Drawable：

- 在 res/drawable/ 中用 `<vector>` 定義一個矢量drawable
- 在 res/drawable/ 中用 `<animated-vector>` 定義一個動畫矢量drawable
- 在`res/anim/'中定義一個或多個Animator

動畫矢量drawable可以用在 `<group>` 和 `<path>` 元素的屬性上。`<group>` 元素定義了一些path或者subgroup，`<path>` 定義了一條被繪畫的路徑。

當你想要定義一個動畫的矢量drawable時，使用 `android:name` 屬性來為group和path賦值一個唯一的名字(name)，這樣你可以通過animator的定義找到他們。比如：

```
<!-- res/drawable/vectordrawable.xml -->
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:height="64dp"
    android:width="64dp"
    android:viewportHeight="600"
    android:viewportWidth="600">
    <group
        android:name="rotationGroup"
        android:pivotX="300.0"
        android:pivotY="300.0"
        android:rotation="45.0" >
        <path
            android:name="v"
            android:fillColor="#000000"
            android:pathData="M300,70 1 0, -70 70,70 0,0 -70,70z" />
    </group>
</vector>
```

動畫矢量drawable的定義是通過name屬性來找到視圖組(group)和路徑(path)的：

```
<!-- res/drawable/animvectordrawable.xml -->
<animated-vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/vectordrawable" >
    <target
        android:name="rotationGroup"
        android:animation="@anim/rotation" />
    <target
        android:name="v"
        android:animation="@anim/path_morph" />
</animated-vector>
```

動畫的定義代表 `ObjectAnimator` 或者 `AnimatorSet` 對象。例子中第一個 animator 將目標組旋轉了 360 度。

```
<!-- res/anim/rotation.xml -->
<objectAnimator
    android:duration="6000"
    android:propertyName="rotation"
    android:valueFrom="0"
    android:valueTo="360" />
```

第二個 animator 將矢量 `drawable` 的路徑從一個形狀(morph)變形到另一個。兩個路徑都必須是可以形變的：他們必須有相同數量的命令，每個命令必須有相同數量的參數

```
<!-- res/anim/path_morph.xml -->
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <objectAnimator
        android:duration="3000"
        android:propertyName="pathData"
        android:valueFrom="M300,70 1 0,-70 70,70 0,0   -70,70z"
        android:valueTo="M300,70 1 0,-70 70,0  0,140 -70,0 z"
        android:valueType="pathType" />
</set>
```

更多信息，請參考 [AnimatedVectorDrawable](#) )的 API 指南。

# 維護兼容性

編寫: allenlsy - 原文: <https://developer.android.com/training/material/compatibility.html>

有些Material Design特性，比如主題和自定義Activits切換效果等，只在Android 5.0 (API level 21) 以上中可用。不過，你仍然可以使用這些特性實現Material Design，並保持對舊版本Android 系統的兼容。

## 定義備選Style

你可以配置你的應用，在支持Material Design的設備上使用Material主題，在舊版本Android上使用舊的主題：

1. 在 `res/values/styles.xml` 中定義一個主題繼承自舊主題（比如Holo）
2. 在 `res/values-v21/styles.xml` 中定義一個同名的主題，繼承自Material 主題
3. 在 `AndroidManifest.xml` 中，將這個主題設置為應用的主題

Note: 如果你的應用設置了一個主題，但是沒有提供備選Style，你可能無法在低於Android 5.0版本的系統中運行應用。

## 提供備選layout

如果你根據Material Design設計的應用的Layout中沒有使用任何Android 5.0 (API level 21)中新的XML屬性，他們在舊版本Android中就能正常工作。否則，你要提供備選Layout。你可以在備選Layout中定義你的應用在舊版本系統中的界面。

在 `res/layout-v21/` 中定義Android 5.0 (API level 21) 以上系統的Layout，在 `res/layout` 中定義早前版本Android的Layout。比如，`res/layout/my_activity.xml` 是對於 `res/layout-v21/my_activity.xml` 的一個備選Layout。

為了避免代碼重複，在 `res/values` 中定義style，然後在 `res/values-v21` 中修改新API需要的style。使用style的繼承，在 `res/values/` 中定義父style，在 `res/values-v21/` 中繼承。

## 使用 Support Library

`v7 support libraries r21` 及更高版本包含了以下Material Design 特性：

- 當你應用一個 `Theme.AppCompat` 主題時，會得到為一些系統控件準備的 Material Design style
- `Theme.AppCompat` 主題包含調色板主體屬性
- `RecyclerView` 組件用於顯示數據集
- `CardView` 組件用於創建卡片
- `Palette` 類用於從圖片提取主色調

## 系統組件

`Theme.AppCompat` 主題中提供了這些組件的 Material Design style：

- `EditText`
- `Spinner`
- `CheckBox`
- `RadioButton`

- SwitchCompat
- CheckedTextView

## 調色板

要獲取Material Design style，並用v7 support library自定義調色板，就要應用以下中的一個Theme.AppCompat主題：

```
<!-- extend one of the Theme.AppCompat themes -->
<style name="Theme.MyTheme" parent="Theme.AppCompat.Light">
    <!-- customize the color palette -->
    <item name="colorPrimary">@color/material_blue_500</item>
    <item name="colorPrimaryDark">@color/material_blue_700</item>
    <item name="colorAccent">@color/material_green_A200</item>
</style>
```

## 列表和卡片

RecyclerView 和 CardView 組件可通過v7 support libraries支持舊版本Android，但有以下限制：

- CardView需要編程實現陰影和其他的padding
- CardView不能將附着與原件有重合部分的子視圖

## 依賴

要在Android 5.0之前的版本使用這些特性，需要在項目的Gradle依賴中加入Android v7 support library:

```
dependencies {
    compile 'com.android.support:appcompat-v7:21.0.+'
    compile 'com.android.support:cardview-v7:21.0.+'
    compile 'com.android.support:recyclerview-v7:21.0.+'
}
```

## 檢查系統版本

以下特性只在Android 5.0 (API level 21) 及以上版本中可用：

- Activity 切換動畫
- 觸摸反饋
- Reveal 動畫（填充動畫效果，譯者注）
- 基於路徑的動畫
- 矢量drawable
- Drawable染色

要保持向下兼容，請在使用這些特性是，使用以下代碼在運行時檢查系統版本：

```
// Check if we're running on Android 5.0 or higher
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    // Call some material design APIs here
} else {
    // Implement this feature without material design
}
```

Note:要聲明應用支持哪些Android 版本，在manifest文件中使用 `android:minSdkVersion` 和 `android:targetSdkVersion` 屬性。要在Android 5.0中使用Material Design特性，設置 `android:targetSdkVersion` 屬性為21。更多信息，參見 [<uses-sdk> API指南](#)。

# 用戶輸入

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/best-user-input.html>

本課程涵蓋的主題包括多種多樣用戶輸入，例如觸摸屏幕手勢、通過屏幕輸入法和硬件鍵盤的文本輸入。

## 使用觸摸手勢

介紹如何編寫允許用戶通過觸摸手勢與觸摸屏幕進行交互的app程序。

## 處理鍵盤輸入事件

介紹在軟輸入方法下（如屏幕鍵盤按鍵情況下）程序的響應表現和執行動作，以及如何優化在硬件鍵盤按鍵下的用戶體驗。

# 使用觸摸手勢

---

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/index.html>

本章節講述，如何編寫一個允許用戶通過觸摸手勢進行交互的app。Android提供了各種各樣的API，來幫你創建和檢測手勢。

儘管對於一些基本的操作來說，你的app不應該依賴於觸摸手勢（因為某些情況下手勢是不用的）。但為你的app添加基於觸摸的交互，將會大大地提高app的可用性和吸引力。

為了給用戶提供一致的、符合直覺的使用體驗，你的app應該遵守Android觸摸手勢的慣常做法。[手勢設計指南](#)告訴你，在Android app中，如何使用常用的手勢。同樣，設計指南也提供了[觸摸反饋](#)的相關內容。

## Lessons

---

- [檢測常用的手勢](#)

學習如何通過使用[GestureDetector](#)來檢測基本的觸摸手勢，如滑動，慣性滑動以及雙擊。

- [追蹤手勢移動](#)

學習如何追蹤手勢移動。

- [Scroll手勢動畫](#)

學習如何使用scrollers（[Scrollers](#)以及[OverScroll](#)）來產生滾動動畫，以響應觸摸事件。

- [處理多觸摸手勢](#)

學習如何檢測多點(手指)觸摸手勢。

- [拖拽與縮放](#)

學習如何實現基於觸摸的拖拽與縮放。

- [管理ViewGroup中的觸摸事件](#)

學習如何在[ViewGroup](#)中管理觸摸事件，以確保事件能被正確地分發到目標views上。

# 檢測常用的手勢

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/detector.html>

當用戶把用一根或多根手指放在觸摸屏上，並且你的應用把這樣的觸摸方式解釋為特定的手勢時，“觸摸手勢”就發生了。相應地，檢測手勢也就有以下兩個階段：

1. 收集觸摸事件的相關數據。
2. 分析這些數據，看它們是否符合，你的app所支持的手勢的標準。

## 支持庫中的類

本節課程的示例程序使用了GestureDetectorCompat和MotionEventCompat類。這些類都是在 Support Library中定義的。如果有可能的情況話，你應該使用支持庫中的類，來為運行着Android1.6及以上版本系統的設備提供兼容性功能。需要注意的一點是，MotionEventCompat並不是MotionEvent的替代品，而是提供了一些靜態工具類函數。你可以把MotionEvent對象作為參數傳遞給這些工具類函數，來獲得與觸摸事件相關的動作(action)。

## 收集數據

當用戶把用一根或多根手指放在觸摸屏上時，接收到這些觸摸事件的View的onTouchEvent()函數就會被回調。對於一系列連續的觸摸事件（位置、壓力、大小、額外的一根手指等等），onTouchEvent()會被調用若干次，並且最終識別為一種手勢。

當用戶第一次觸摸屏幕時，手勢就開始了。其後系統會持續地追蹤用戶手指的位置，在用戶手指全都離開屏幕時，手勢結束。在整個交互期間，被分發給函數的MotionEvent對象，提供了每次交互的詳細信息。你的app可以使用MotionEvent提供的這些數據，來判斷某種特定的手勢是否發生了。

### 為Activity或View捕獲觸摸事件

為了捕獲Activity或View中的觸摸事件，你可以重寫onTouchEvent()回調函數。

接下來的代碼段，使用了getActionMasked()函數，來從event參數中抽取出用戶執行的動作。它提供了一些原始的觸摸數據，你可以使用這些數據，來判斷某個特定手勢是否發生了。

```
public class MainActivity extends Activity {  
    ...  
    // This example shows an Activity, but you would use the same approach if  
    // you were subclassing a View.  
    @Override  
    public boolean onTouchEvent(MotionEvent event){  
  
        int action = MotionEventCompat.getActionMasked(event);  
  
        switch(action) {  
            case (MotionEvent.ACTION_DOWN) :  
                Log.d(DEBUG_TAG, "Action was DOWN");  
                return true;  
            case (MotionEvent.ACTION_MOVE) :  
                Log.d(DEBUG_TAG, "Action was MOVE");  
                return true;  
            case (MotionEvent.ACTION_UP) :  
                Log.d(DEBUG_TAG, "Action was UP");  
                return true;  
            case (MotionEvent.ACTION_CANCEL) :  
                Log.d(DEBUG_TAG, "Action was CANCEL");  
        }  
    }  
}
```

```
        return true;
    case (MotionEvent.ACTION_OUTSIDE) :
        Log.d(DEBUG_TAG, "Movement occurred outside bounds " +
            "of current screen element");
        return true;
    default :
        return super.onTouchEvent(event);
    }
}
```

然後，你可以對這些事件做些自己的處理，以判斷某個手勢是否出現了。這種是針對自定義手勢，你所需要進行地處理。然而，如果你的app僅僅需要一些常見的手勢，如雙擊，長按，快速滑動（fling）等，你可以使用[GestureDetector](#)類來完成。[GestureDetector](#)可以讓你更簡單地檢測常見手勢，並且無需自行處理單個的觸摸事件。相關內容將會在下面的[Detect Gestures](#)中討論。

## 捕獲單個view的觸摸事件

作為[onTouchEvent\(\)](#)的一種替換方式，你也可以使用[setOnTouchListener\(\)](#)函數，來把View.OnTouchListener關聯到任意的View上。這樣可以讓你不繼承已有的View，也能監聽它的觸摸事件。比如：

```
View myView = findViewById(R.id.my_view);
myView.setOnTouchListener(new OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        // ... Respond to touch events
        return true;
    }
});
```

創建listener對象時，對ACTION\_DOWN事件返回false需保持警惕。如果返回false，會讓listener對象接收不到後續的ACTION\_MOVE、ACTION\_UP等系列事件。這是因為ACTION\_DOWN事件是所有觸摸事件的開端。

如果你正在寫一個自定義View，你也可以像上面描述的那樣重寫[onTouchEvent\(\)](#)函數。

## 檢測手勢

Android提供了[GestureDetector](#)類來檢測一般手勢。它所支持的手勢包括[onDown\(\)](#), [onLongPress\(\)](#),[onFling\(\)](#)等。你可以把[GestureDetector](#)和上面描述的[onTouchEvent\(\)](#)函數結合在一起使用。

## 檢測所有支持的手勢

當你實例化一個[GestureDetectorCompat](#)對象時，需要一個實現了[GestureDetector.OnGestureListener](#)接口的的對象作爲參數。當某個特定的觸摸事件發生時，[GestureDetector.OnGestureListener](#)就會通知用戶。爲了讓你的[GestureDetector](#)對象能到接收到觸摸事件，你需要重寫View或Activity的[onTouchEvent\(\)](#)函數，並且把所有捕獲到的事件傳遞給detector對象。

接下來的代碼段中，on型的函數的返回值是true，意味着你已經處理完這個觸摸事件了。如果返回false，則會把事件沿view棧傳遞，直到觸摸事件被成功地處理了。

運行下面的代碼段，來瞭解你與觸摸屏交互時，動作（action）是如何觸發的，以及每個觸摸事件[MotionEvent](#)中的內容。你也會意識到，一個簡單的交互會產生多少的數據。

```
public class MainActivity extends Activity implements
    GestureDetector.OnGestureListener,
    GestureDetector.OnDoubleTapListener{
```

```
private static final String DEBUG_TAG = "Gestures";
private GestureDetectorCompat mDetector;

// Called when the activity is first created.
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Instantiate the gesture detector with the
    // application context and an implementation of
    // GestureDetector.OnGestureListener
    mDetector = new GestureDetectorCompat(this, this);
    // Set the gesture detector as the double tap
    // listener.
    mDetector.setOnDoubleTapListener(this);
}

@Override
public boolean onTouchEvent(MotionEvent event){
    this.mDetector.onTouchEvent(event);
    // Be sure to call the superclass implementation
    return super.onTouchEvent(event);
}

@Override
public boolean onDown(MotionEvent event) {
    Log.d(DEBUG_TAG, "onDown: " + event.toString());
    return true;
}

@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
    float velocityX, float velocityY) {
    Log.d(DEBUG_TAG, "onFling: " + event1.toString() + event2.toString());
    return true;
}

@Override
public void onLongPress(MotionEvent event) {
    Log.d(DEBUG_TAG, "onLongPress: " + event.toString());
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
    float distanceY) {
    Log.d(DEBUG_TAG, "onScroll: " + e1.toString() + e2.toString());
    return true;
}

@Override
public void onShowPress(MotionEvent event) {
    Log.d(DEBUG_TAG, "onShowPress: " + event.toString());
}

@Override
public boolean onSingleTapUp(MotionEvent event) {
    Log.d(DEBUG_TAG, "onSingleTapUp: " + event.toString());
    return true;
}

@Override
public boolean onDoubleTap(MotionEvent event) {
    Log.d(DEBUG_TAG, "onDoubleTap: " + event.toString());
    return true;
}

@Override
public boolean onDoubleTapEvent(MotionEvent event) {
    Log.d(DEBUG_TAG, "onDoubleTapEvent: " + event.toString());
    return true;
}

@Override
public boolean onSingleTapConfirmed(MotionEvent event) {
```

```
        Log.d(DEBUG_TAG, "onSingleTapConfirmed: " + event.toString());
        return true;
    }
}
```

## 檢測支持手勢的部分

如果你只想處理幾種手勢，你可以選擇繼承[GestureDetector.SimpleOnGestureListener](#)類，而不是實現[GestureDetector.OnGestureListener](#)接口。

[GestureDetector.SimpleOnGestureListener](#)類實現了所有的on型函數，其中都返回false。因此，你可以僅僅重寫你所需要的函數。比如，下面的代碼段中，創建了一個繼承自[GestureDetector.SimpleOnGestureListener](#)的類，並重寫了onFling()和onDown()函數。

無論你是否使用[GestureDetector.OnGestureListener](#)類，最好都實現onDown()函數並且返回true。這是因為所有的手勢都是由onDown()消息開始的。如果你讓onDown()函數返回false，就像[GestureDetector.SimpleOnGestureListener](#)類中默認實現地那樣，系統會假定你想忽略手勢的剩餘部分，[GestureDetector.OnGestureListener](#)中的其他函數也就永遠不會被調用。這可能讓你的app出現意想不到的問題。僅僅當你真的想忽略整個手勢時，你才應該讓onDown()函數返回false。

```
public class MainActivity extends Activity {

    private GestureDetectorCompat mDetector;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mDetector = new GestureDetectorCompat(this, new MyGestureListener());
    }

    @Override
    public boolean onTouchEvent(MotionEvent event){
        this.mDetector.onTouchEvent(event);
        return super.onTouchEvent(event);
    }

    class MyGestureListener extends GestureDetector.SimpleOnGestureListener {
        private static final String DEBUG_TAG = "Gestures";

        @Override
        public boolean onDown(MotionEvent event) {
            Log.d(DEBUG_TAG, "onDown: " + event.toString());
            return true;
        }

        @Override
        public boolean onFling(MotionEvent event1, MotionEvent event2,
                float velocityX, float velocityY) {
            Log.d(DEBUG_TAG, "onFling: " + event1.toString() + event2.toString());
            return true;
        }
    }
}
```

# 追蹤手勢移動

編寫:Andrwyw - 原文：<http://developer.android.com/training/gestures/movement.html>

本節課程講述如何追蹤手勢移動。

每當當前的觸摸位置、壓力、大小發生變化時，`ACTION_MOVE`事件都會觸發`onTouchEvent()`函數。正如檢測常用的  
手勢中描述的那樣，觸摸事件全部都記錄在`onTouchEvent()`函數的`MotionEvent`參數中。

因為基於手指的觸摸的交互方式並不總是非常精確，所以檢測觸摸事件更多的是基於手勢移動，而非簡單地基於觸摸。  
為了幫助app區分基於移動的手勢（如滑動）和非移動手勢（如簡單地點擊），Android引入了“touch slop”的概念。  
Touch slop是指，在被識別為基於移動的手勢前，用戶觸摸可移動的那一段像素距離。關於這一主題的更多討論，可以在管理ViewGroup中的觸摸事件中查看。

根據應用的需求，有多種追蹤手勢移動的方式可以選擇。比如：

- 追蹤手指的起始和終止位置（比如，把屏幕上的對象從A點移動到B點）
- 根據x、y軸座標，追蹤手指移動的方向。
- 追蹤歷史狀態。你可以通過調用`MotionEvent`的`getHistorySize()`方法，來獲得一個手勢的歷史尺寸大小。你可以通過移動事件的`getHistorical<Value>`系列函數，來獲得事件之前的位置、尺寸、時間以及按壓力(pressures)。當你需要繪製用戶手指痕跡時，歷史狀態非常有用，比如觸摸繪圖。查看`MotionEvent`來瞭解更多細節。
- 追蹤手指在觸摸屏上滑過的速度。

# 追蹤速度

你可以簡單地用基於距離，或(和)基於手指移動方向的移動手勢。但是速度經常也是追蹤手勢特性的一個決定性因素，甚至是判斷一個手勢是否發生的依據。為了讓計算速度更容易，Android提供了`VelocityTracker`類以及支持庫中的`VelocityTrackerCompat`類。`VelocityTracker`類可以幫助你追蹤觸摸事件中的速度因素。如果速度是你的手勢的一個判斷標準，比如快速滑動(fling)，那麼這些類是很有用的。

下面是一個簡單的例子，說明了`VelocityTracker`中API函數的用處。

```
public class MainActivity extends Activity {
    private static final String DEBUG_TAG = "Velocity";
    ...
    private VelocityTracker mVelocityTracker = null;
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int index = event.getActionIndex();
        int action = event.getActionMasked();
        int pointerId = event.getPointerId(index);

        switch(action) {
            case MotionEvent.ACTION_DOWN:
                if(mVelocityTracker == null) {
                    // Retrieve a new VelocityTracker object to watch the velocity of a motion.
                    mVelocityTracker = VelocityTracker.obtain();
                }
                else {
                    // Reset the velocity tracker back to its initial state.
                    mVelocityTracker.clear();
                }
                // Add a user's movement to the tracker.
                mVelocityTracker.addMovement(event);
                break;
            case MotionEvent.ACTION_MOVE:
```

```
mVelocityTracker.addMovement(event);
// When you want to determine the velocity, call
// computeCurrentVelocity(). Then call getXVelocity()
// and getYVelocity() to retrieve the velocity for each pointer ID.
mVelocityTracker.computeCurrentVelocity(1000);
// Log velocity of pixels per second
// Best practice to use VelocityTrackerCompat where possible.
Log.d("", "X velocity: " +
    VelocityTrackerCompat.getXVelocity(mVelocityTracker,
    pointerId));
Log.d("", "Y velocity: " +
    VelocityTrackerCompat.getYVelocity(mVelocityTracker,
    pointerId));
    break;
case MotionEvent.ACTION_UP:
case MotionEvent.ACTION_CANCEL:
    // Return a VelocityTracker object back to be re-used by others.
    mVelocityTracker.recycle();
    break;
}
return true;
}
}
```

注意：需要注意的是，你應該在ACTION\_MOVE事件後計算速度，而不是在ACTION\_UP事件後。在ACTION\_UP事件之後計算，x、y方向上的速度都會是0。

# Scroll手勢動畫

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/scroll.html>

在Android中，通常使用`ScrollView`類來實現滾動（scroll）。任何可能超過父類邊界的佈局，都應該嵌套在`ScrollView`中，來提供一個由系統框架管理的可滾動的view。僅在某些特殊情形下，我們纔要實現一個自定義scroller。本節課程就描述了這樣一個情形：使用scrollers顯示滾動效果，以響應觸摸手勢。

為了收集數據來產生滾動動畫，以響應一個觸摸事件，你可以使用scrollers(`Scroller`或者`OverScroller`)。這兩個類很相似，但`OverScroller`有一些函數，能在平移或快速滑動手勢後，向用戶指出已經達到內容盡頭了。`InteractiveChart`例子使用了`EdgeEffect`類（實際上是`EdgeEffectCompat`類），在用戶到達內容盡頭時顯示發光效果。

注意：比起`Scroller`類，我們更推薦使用`OverScroller`類來產生滾動動畫。`OverScroller`類為老設備提供了很好的向後兼容性。另外需要注意的是，僅當你要自己實現滾動時，才需要使用scrollers。如果你把佈局嵌套在`ScrollView`和`HorizontalScrollView`中，它們會幫你把這些做好。

通過使用平臺標準的滾動物理因素（摩擦、速度等），scroller被用來根據時間，產生滾動動畫。實際上，scroller本身不會繪製任何東西。Scrollers只是隨着時間的推移，幫你追蹤滾動的偏移量，但它們不會自動地把這些位置應用到你的view上。你應該按一定頻率，獲取並應用這些新的座標值，來讓滾動動畫更加順滑。

## 理解術語Scrolling

在Android中，“Scrolling”這個詞根據不同情景有着不同的含義。

Scrolling是指移動視窗（viewport）（指你正在看的內容所在的‘窗口’）的一般過程。當在x軸和y軸方向同時滾動時，就叫做平移（panning）。示例程序提供的`InteractiveChart`類，展示了兩種不同類型的scrolling，拖拽與快速滑動。

- 拖拽(dragging)是scrolling的一種類型，當用戶在觸摸屏上滑動手指時發生。重寫`GestureDetector.OnGestureListener`的`onScroll()`函數，經常用來實現簡單的拖拽。關於拖拽的更多討論，可以查看[拖拽與縮放](#)章節。
- 快速滑動(fling)這種類型的scrolling，在用戶快速拖拽後，擡起手指時發生。當用戶擡高手指後，你通常想繼續保持scrolling(移動視窗)，但會一直減速直到視窗停止移動。通過重寫`GestureDetector.OnGestureListener`的`onFling()`函數，使用scroller對象，可實現快速滑動。這種用法也就是本節課程的主題。

scroller對象通常會與快速滑動手勢結合起來使用。但在任何你想讓UI展示scrolling動畫，以響應觸摸事件的場景，都可以用它們來實現。比如，你可以重寫`onTouchEvent()`函數，直接處理觸摸事件，並且產生一個scrolling效果或“對齊到頁”動畫(snapping to page)，來響應這些觸摸事件。

## 實現基於觸摸的Scrolling

本節講述如何使用scroller。下面的代碼段來自`InteractiveChart`示例。它使用`[GestureDetector]` [`GestureDetector_url`]，並且重寫了`GestureDetector.SimpleOnGestureListener`的`onFling()`函數。它使用`OverScroller`追蹤快速滑動（fling）手勢。快速滑動手勢後，如果用戶到達內容盡頭了，應用會顯示一種發光效果。

注意：`InteractiveChart`樣例程序展示了一個可縮放、平移、滑動的表格。在接下來的代碼段中，`mContentRect`表示view中的一塊矩形座標區域，該區域將被用來繪製表格。在任意給定的時間點，表格中某一部分會被繪製在這個區域內。`mCurrentViewport`表示當前在屏幕上可見的那一部分表格。因為像素偏移量通

常當作整型處理，所以mContentRect是Rect類型的。因為圖表的區域範圍是數值型/浮點型值，所以mCurrentViewport是RectF類型。

代碼段的第一部分展示了`onFling()`函數的實現：

```
// The current viewport. This rectangle represents the currently visible
// chart domain and range. The viewport is the part of the app that the
// user manipulates via touch gestures.
private RectF mCurrentViewport =
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);

// The current destination rectangle (in pixel coordinates) into which the
// chart data should be drawn.
private Rect mContentRect;

private OverScroller mScroller;
private RectF mScrollerStartViewport;
...

private final GestureDetector.SimpleOnGestureListener mGestureListener
    = new GestureDetector.SimpleOnGestureListener() {
    @Override
    public boolean onDown(MotionEvent e) {
        // Initiates the decay phase of any active edge effects.
        releaseEdgeEffects();
        mScrollerStartViewport.set(mCurrentViewport);
        // Aborts any active scroll animations and invalidates.
        mScroller.forceFinished(true);
        ViewCompat.postInvalidateOnAnimation(InteractiveLineGraphView.this);
        return true;
    }
    ...
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2,
        float velocityX, float velocityY) {
        fling((int) -velocityX, (int) -velocityY);
        return true;
    }
};

private void fling(int velocityX, int velocityY) {
    // Initiates the decay phase of any active edge effects.
    releaseEdgeEffects();
    // Flings use math in pixels (as opposed to math based on the viewport).
    Point surfaceSize = computeScrollSurfaceSize();
    mScrollerStartViewport.set(mCurrentViewport);
    int startX = (int) (surfaceSize.x * (mScrollerStartViewport.left -
        AXIS_X_MIN) / (
        AXIS_X_MAX - AXIS_X_MIN));
    int startY = (int) (surfaceSize.y * (AXIS_Y_MAX -
        mScrollerStartViewport.bottom) / (
        AXIS_Y_MAX - AXIS_Y_MIN));
    // Before flinging, aborts the current animation.
    mScroller.forceFinished(true);
    // Begins the animation
    mScroller.fling(
        // Current scroll position
        startX,
        startY,
        velocityX,
        velocityY,
        /*
         * Minimum and maximum scroll positions. The minimum scroll
         * position is generally zero and the maximum scroll position
         * is generally the content size less the screen size. So if the
         * content width is 1000 pixels and the screen width is 200
         * pixels, the maximum scroll offset should be 800 pixels.
         */
        0, surfaceSize.x - mContentRect.width(),
        0, surfaceSize.y - mContentRect.height(),
        // The edges of the content. This comes into play when using
        // the EdgeEffect class to draw "glow" overlays.
```

```

    mContentRect.width() / 2,
    mContentRect.height() / 2);
// Invalidates to trigger computeScroll()
ViewCompat.postInvalidateOnAnimation(this);
}

```

當 `onFling()` 函數調用 `postInvalidateOnAnimation()` 時，它會觸發 `>computeScroll()` 來更新 x、y 的值。通常一個子 view 用 scroller 對象來產生滾動動畫時會這樣做，就像本例一樣。

大多數 views 直接通過 `scrollTo()` 函數傳遞 scroller 對象的 x、y 座標值。接下來的 `computeScroll()` 函數的實現中採用了一種不同的方式。它調用 `computeScrollOffset()` 函數來獲得當前位置的 x、y 值。當滿足邊緣顯示發光效果的條件時（圖表已被放大顯示，x 或 y 值超過邊界，並且 app 當前沒有顯示 overscroll），這段代碼會設置 overscroll 發光效果，並調用 `postInvalidateOnAnimation()` 函數來讓 view 失效重繪：

```

// Edge effect / overscroll tracking objects.
private EdgeEffectCompat mEdgeEffectTop;
private EdgeEffectCompat mEdgeEffectBottom;
private EdgeEffectCompat mEdgeEffectLeft;
private EdgeEffectCompat mEdgeEffectRight;

private boolean mEdgeEffectTopActive;
private boolean mEdgeEffectBottomActive;
private boolean mEdgeEffectLeftActive;
private boolean mEdgeEffectRightActive;

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

    boolean needsInvalidate = false;

    // The scroller isn't finished, meaning a fling or programmatic pan
    // operation is currently active.
    if (mScroller.computeScrollOffset()) {
        Point surfaceSize = computeScrollSurfaceSize();
        int currX = mScroller.getCurrX();
        int currY = mScroller.getCurrY();

        boolean canScrollX = (mCurrentViewport.left > AXIS_X_MIN
            || mCurrentViewport.right < AXIS_X_MAX);
        boolean canScrollY = (mCurrentViewport.top > AXIS_Y_MIN
            || mCurrentViewport.bottom < AXIS_Y_MAX);

        /*
         * If you are zoomed in and currX or currY is
         * outside of bounds and you're not already
         * showing overscroll, then render the overscroll
         * glow edge effect.
         */
        if (canScrollX
            && currX < 0
            && mEdgeEffectLeft.isFinished()
            && !mEdgeEffectLeftActive) {
            mEdgeEffectLeft.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller));
            mEdgeEffectLeftActive = true;
            needsInvalidate = true;
        } else if (canScrollX
            && currX > (surfaceSize.x - mContentRect.width())
            && mEdgeEffectRight.isFinished()
            && !mEdgeEffectRightActive) {
            mEdgeEffectRight.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller));
            mEdgeEffectRightActive = true;
            needsInvalidate = true;
        }

        if (canScrollY
            && currY < 0
            && mEdgeEffectTop.isFinished()
            && !mEdgeEffectTopActive) {
            mEdgeEffectTop.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller));
            mEdgeEffectTopActive = true;
            needsInvalidate = true;
        } else if (canScrollY
            && currY > (surfaceSize.y - mContentRect.height())
            && mEdgeEffectBottom.isFinished()
            && !mEdgeEffectBottomActive) {
            mEdgeEffectBottom.onAbsorb((int)
                OverScrollerCompat.getCurVelocity(mScroller));
            mEdgeEffectBottomActive = true;
            needsInvalidate = true;
        }
    }
}

```

```

        && mEdgeEffectTop.isFinished()
        && !mEdgeEffectTopActive) {
    mEdgeEffectTop.onAbsorb((int)
        OverScrollerCompat.getCurvVelocity(mScroller));
    mEdgeEffectTopActive = true;
    needsInvalidate = true;
} else if (canScrollY
    && currY > (surfaceSize.y - mContentRect.height())
    && mEdgeEffectBottom.isFinished()
    && !mEdgeEffectBottomActive) {
    mEdgeEffectBottom.onAbsorb((int)
        OverScrollerCompat.getCurvVelocity(mScroller));
    mEdgeEffectBottomActive = true;
    needsInvalidate = true;
}
...
}

```

這是縮放部分的代碼：

```

// Custom object that is functionally similar to Scroller
Zoomer mZoomer;
private PointF mZoomFocalPoint = new PointF();
...

// If a zoom is in progress (either programmatically or via double
// touch), performs the zoom.
if (mZoomer.computeZoom()) {
    float newWidth = (1f - mZoomer.getCurvZoom()) *
        mScrollerStartViewport.width();
    float newHeight = (1f - mZoomer.getCurvZoom()) *
        mScrollerStartViewport.height();
    float pointWithinViewportX = (mZoomFocalPoint.x -
        mScrollerStartViewport.left)
        / mScrollerStartViewport.width();
    float pointWithinViewportY = (mZoomFocalPoint.y -
        mScrollerStartViewport.top)
        / mScrollerStartViewport.height();
    mCurrentViewport.set(
        mZoomFocalPoint.x - newWidth * pointWithinViewportX,
        mZoomFocalPoint.y - newHeight * pointWithinViewportY,
        mZoomFocalPoint.x + newWidth * (1 - pointWithinViewportX),
        mZoomFocalPoint.y + newHeight * (1 - pointWithinViewportY));
    constrainViewport();
    needsInvalidate = true;
}
if (needsInvalidate) {
    ViewCompat.postInvalidateOnAnimation(this);
}

```

這是上面代碼段中調用過的computeScrollSurfaceSize()函數。他會計算當前可滾動的尺寸，以像素為單位。舉例來說，如果整個圖表區域都是可見的，它的值就簡單地等於mContentRect的大小。如果圖表在兩個方向上都放大到200%，此函數返回的尺寸在水平、垂直方向上都會大兩倍。

```

private Point computeScrollSurfaceSize() {
    return new Point(
        (int) (mContentRect.width() * (AXIS_X_MAX - AXIS_X_MIN)
            / mCurrentViewport.width()),
        (int) (mContentRect.height() * (AXIS_Y_MAX - AXIS_Y_MIN)
            / mCurrentViewport.height()));
}

```

關於scroller用法的另一個示例，可查看[ViewPager類的源代碼](#)。它用滾動來響應flings，並且使用滾動來實現“對齊到頁”(snapping to page)動畫。



# 處理多觸摸手勢

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/multi.html>

多點觸摸觸手勢是指在同一時間有多點（手指）觸碰屏幕。本節課程講述，如何檢測涉及多點的觸摸手勢。

## 追蹤多點

當多個手指同時觸摸屏幕時，系統會產生如下的觸摸事件：

- **ACTION\_DOWN**-針對觸摸屏幕的第一個點。此事件是手勢的開端。第一觸摸點的數據在MotionEvent中的索引總是0。
- **ACTION\_POINTER\_DOWN**-針對第一點後，出現在屏幕上的額外的點。這個點的數據在MotionEvent中的索引，可以通過getActionIndex()獲得。
- **ACTION\_MOVE**-按下手勢時發生變化。
- **ACTION\_POINTER\_UP**-當非主要點（non-primary pointer）離開屏幕時，發送此事件。
- **ACTION\_UP**-當最後一點離開屏幕時發送此事件。

你可以通過各個點的索引以及id，單獨地追蹤MotionEvent中的每個點。

- **Index**：MotionEvent把各個點的信息都存儲在一個數組中。點的索引值就是它在數組中的位置。大多數用來與點交互的MotionEvent函數都是以索引值作為參數的，而不是點的ID。
- **ID**：每個點也都有一個ID，該ID在整個手勢期間一直存在，以便你單獨地追蹤每個點。

每個獨立的點在移動事件中出現的次序是不固定的。因此，從一個事件到另一個事件，點的索引值是可以改變的，但點的ID在它的生命週期內是保證不會改變的。使用getPointerId()可以獲得一個點的ID，在手勢隨後的移動事件中，就可以用該ID來追蹤這個點。對於隨後一系列的事件，可以使用findPointerIndex()函數，來獲得對應給定ID的點在移動事件中的索引值。如下：

```
private int mActivePointerId;

public boolean onTouchEvent(MotionEvent event) {
    ...
    // Get the pointer ID
    mActivePointerId = event.getPointerId(0);

    // ... Many touch events later...

    // Use the pointer ID to find the index of the active pointer
    // and fetch its position
    int pointerIndex = event.findPointerIndex(mActivePointerId);
    // Get the pointer's current position
    float x = event.getX(pointerIndex);
    float y = event.getY(pointerIndex);
}
```

## 獲取MotionEvent的動作

你應該總是使用getActionMasked()函數（或者更好用MotionEventCompat.getActionMasked()這個兼容版本）來獲取MotionEvent的動作(action)。與舊的getAction()函數不同的是，getActionMasked()是設計用來處理多點觸摸的。它會返回執行過的動作的掩碼值，不包括點的索引位。你可以使用getActionIndex()來獲得與該動作關聯的點的索引值。這在接下來的代碼段中可以看到。

注意：這個樣例使用的是MotionEventCompat類。該類在Support Library中。你應該使用MotionEventCompat類，來提供對更多平臺的支持。需要注意的一點是，MotionEventCompat並不是MotionEvent類的替代品。準確來說，它提供了一些靜態工具類函數，你可以把MotionEvent對象作為參數傳遞給這些函數，來得到與事件相關的動作。

```
int action = MotionEventCompat.getActionMasked(event);
// Get the index of the pointer associated with the action.
int index = MotionEventCompat.getActionIndex(event);
int xPos = -1;
int yPos = -1;

Log.d(DEBUG_TAG, "The action is " + actionToString(action));

if (event.getPointerCount() > 1) {
    Log.d(DEBUG_TAG, "Multitouch event");
    // The coordinates of the current screen contact, relative to
    // the responding View or Activity.
    xPos = (int)MotionEventCompat.getX(event, index);
    yPos = (int)MotionEventCompat.getY(event, index);

} else {
    // Single touch event
    Log.d(DEBUG_TAG, "Single touch event");
    xPos = (int)MotionEventCompat.getX(event, index);
    yPos = (int)MotionEventCompat.getY(event, index);
}

...
...

// Given an action int, returns a string description
public static String actionToString(int action) {
    switch (action) {

        case MotionEvent.ACTION_DOWN: return "Down";
        case MotionEvent.ACTION_MOVE: return "Move";
        case MotionEvent.ACTION_POINTER_DOWN: return "Pointer Down";
        case MotionEvent.ACTION_UP: return "Up";
        case MotionEvent.ACTION_POINTER_UP: return "Pointer Up";
        case MotionEvent.ACTION_OUTSIDE: return "Outside";
        case MotionEvent.ACTION_CANCEL: return "Cancel";
    }
    return "";
}
```

關於多點觸摸的更多內容以及示例，可以查看[拖拽與縮放](#)章節。

# 拖拽與縮放

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/scale.html>

本節課程講述，使用`onTouchEvent()`截獲觸摸事件後，如何使用觸摸手勢拖拽、縮放屏幕上的對象。

## 拖拽一個對象

如果你的目標版本為3.0或以上，你可以使用`View.OnDragListener`監聽內置的drag-and-drop事件，[拖拽與釋放](#)中有更多相關描述。

對於觸摸手勢來說，一個很常見的操作是在屏幕上拖拽一個對象。接下來的代碼段讓用戶可以拖拽屏幕上的圖片。需要注意以下幾點：

- 拖拽操作時，即使有額外的手指放置到屏幕上，app也必須保持對最初的點（手指）的追蹤。比如，想像在拖拽圖片時，用戶放置了第二根手指在屏幕上，並且擡起了第一根手指。如果你的app只是單獨地追蹤每個點，它會把第二個點當做默認的點，並且把圖片移到該點的位置。
- 為了防止這種情況發生，你的app需要區分初始點以及隨後的任意的觸摸點。要做到這一點，它需要追蹤[處理多觸摸手勢](#)中提到過的`ACTION_POINTER_DOWN`、`ACTION_POINTER_UP`事件。每當第二根手指按下或拿起時，`ACTION_POINTER_DOWN`、`ACTION_POINTER_UP`事件就會傳遞給`onTouchEvent()`回調函數。
- 當`ACTION_POINTER_UP`事件發生時，示例程序會移除對該點的索引值的引用，確保操作中的點的ID(the active pointer ID)不會引用已經不在觸摸屏上的觸摸點。這種情況下，app會選擇另一個觸摸點來作為操作中(active)的點，並保存它當前的x、y值。由於在`ACTION_MOVE`事件時，這個保存的位置會被用來計算屏幕上的對象將要移動的距離，所以app會始終根據正確的觸摸點來計算移動的距離。

下面的代碼段允許用戶拖拽屏幕上的對象。它會記錄操作中的點（active pointer）的初始位置，計算觸摸點移動過的距離，再把對象移動到新的位置。如上所述，它也正確地處理了額外觸摸點的可能。

需要注意的是，代碼段中使用了`getActionMasked()`函數。你應該始終使用這個函數（或者更好用`MotionEventCompat.getActionMasked()`這個兼容版本）來獲得`MotionEvent`對應的動作(action)。不像舊的`getAction()`函數，`getActionMasked()`就是設計用來處理多點觸摸的。它會返回執行過的動作的掩碼值，不包括該點的索引位。

```
// The 'active pointer' is the one currently moving our object.  
private int mActivePointerId = INVALID_POINTER_ID;  
  
@Override  
public boolean onTouchEvent(MotionEvent ev) {  
    // Let the ScaleGestureDetector inspect all events.  
    mScaleDetector.onTouchEvent(ev);  
  
    final int action = MotionEventCompat.getActionMasked(ev);  
  
    switch (action) {  
        case MotionEvent.ACTION_DOWN: {  
            final int pointerIndex = MotionEventCompat.getActionIndex(ev);  
            final float x = MotionEventCompat.getX(ev, pointerIndex);  
            final float y = MotionEventCompat.getY(ev, pointerIndex);  
  
            // Remember where we started (for dragging)  
            mLasteTouchX = x;  
            mLasteTouchY = y;  
            // Save the ID of this pointer (for dragging)  
            mActivePointerId = MotionEventCompat.getPointerId(ev, 0);  
            break;  
        }  
    }  
}
```

```

}

case MotionEvent.ACTION_MOVE: {
    // Find the index of the active pointer and fetch its position
    final int pointerIndex =
        MotionEventCompat.findPointerIndex(ev, mActivePointerId);

    final float x = MotionEventCompat.getX(ev, pointerIndex);
    final float y = MotionEventCompat.getY(ev, pointerIndex);

    // Calculate the distance moved
    final float dx = x - mLstTouchX;
    final float dy = y - mLstTouchY;

    mPosX += dx;
    mPosY += dy;

    invalidate();

    // Remember this touch position for the next move event
    mLstTouchX = x;
    mLstTouchY = y;

    break;
}

case MotionEvent.ACTION_UP: {
    mActivePointerId = INVALID_POINTER_ID;
    break;
}

case MotionEvent.ACTION_CANCEL: {
    mActivePointerId = INVALID_POINTER_ID;
    break;
}

case MotionEvent.ACTION_POINTER_UP: {

    final int pointerIndex = MotionEventCompat.getActionIndex(ev);
    final int pointerId = MotionEventCompat.getPointerId(ev, pointerIndex);

    if (pointerId == mActivePointerId) {
        // This was our active pointer going up. Choose a new
        // active pointer and adjust accordingly.
        final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
        mLstTouchX = MotionEventCompat.getX(ev, newPointerIndex);
        mLstTouchY = MotionEventCompat.getY(ev, newPointerIndex);
        mActivePointerId = MotionEventCompat.getPointerId(ev, newPointerIndex);
    }
    break;
}
}

return true;
}

```

## 通過拖拽平移

前一節展示了一個，在屏幕上拖拽對象的例子。另一個常見的場景是平移（panning），是指用戶通過拖拽移動引起x、y軸方向發生滾動(scrolling)。上面的代碼段直接截獲了MotionEvent動作來實現拖拽。這一部分的代碼段，利用了平臺對常用手勢的內置支持。它重寫了GestureDetector.SimpleOnGestureListener的onScroll()函數。

更詳細地說，當用戶拖拽手指來平移內容時，onScroll()函數就會被調用。onScroll()函數只會在手指按下的情況下被調用，一旦手指離開屏幕了，要麼手勢終止，要麼快速滑動(fling)手勢開始（如果手指在離開屏幕前快速移動了一段距離）。關於滾動與快速滑動的更多討論，可以查看[Scroll手勢動畫](#)章節。

這裏是onScroll()的相關代碼段：

```

// The current viewport. This rectangle represents the currently visible
// chart domain and range.
private RectF mCurrentViewport =
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);

// The current destination rectangle (in pixel coordinates) into which the
// chart data should be drawn.
private Rect mContentRect;

private final GestureDetector.SimpleOnGestureListener mGestureListener
    = new GestureDetector.SimpleOnGestureListener() {
    ...

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2,
    float distanceX, float distanceY) {
    // Scrolling uses math based on the viewport (as opposed to math using pixels).

    // Pixel offset is the offset in screen pixels, while viewport offset is the
    // offset within the current viewport.
    float viewportOffsetX = distanceX * mCurrentViewport.width()
        / mContentRect.width();
    float viewportOffsetY = -distanceY * mCurrentViewport.height()
        / mContentRect.height();

    ...
    // Updates the viewport, refreshes the display.
    setViewportBottomLeft(
        mCurrentViewport.left + viewportOffsetX,
        mCurrentViewport.bottom + viewportOffsetY);
    ...
    return true;
}

```

onScroll() 函數中滑動視窗(viewport)來響應觸摸手勢的實現：

```

/**
 * Sets the current viewport (defined by mCurrentViewport) to the given
 * X and Y positions. Note that the Y value represents the topmost pixel position,
 * and thus the bottom of the mCurrentViewport rectangle.
 */
private void setViewportBottomLeft(float x, float y) {
    /*
     * Constrains within the scroll range. The scroll range is simply the viewport
     * extremes (AXIS_X_MAX, etc.) minus the viewport size. For example, if the
     * extremes were 0 and 10, and the viewport size was 2, the scroll range would
     * be 0 to 8.
     */

    float curWidth = mCurrentViewport.width();
    float curHeight = mCurrentViewport.height();
    x = Math.max(AXIS_X_MIN, Math.min(x, AXIS_X_MAX - curWidth));
    y = Math.max(AXIS_Y_MIN + curHeight, Math.min(y, AXIS_Y_MAX));

    mCurrentViewport.set(x, y - curHeight, x + curWidth, y);

    // Invalidates the View to update the display.
    ViewCompat.postInvalidateOnAnimation(this);
}

```

## 使用觸摸手勢進行縮放

如同[檢測常用手勢](#)章節中提到的，[GestureDetector](#)可以幫助你檢測Android中的常見手勢，例如滾動，快速滾動以及長按。對於縮放，Android也提供了[ScaleGestureDetector](#)類。當你想讓view能識別額外的手勢時，你可以同時使用[GestureDetector](#)和[ScaleGestureDetector](#)類。

為了報告檢測到的手勢事件，手勢檢測需要一個作為構造函數參數的`listener`對象。`ScaleGestureDetector`使用`ScaleGestureDetector.OnScaleGestureListener`。Android提供`ScaleGestureDetector.SimpleOnScaleGestureListener`類作為幫助類，如果你不是關注所有的手勢事件，你可以繼承(extend)它。

## 基本的縮放示例

下面的代碼段展示了縮放功能中的基本部分。

```
private ScaleGestureDetector mScaleDetector;
private float mScaleFactor = 1.0f;

public MyCustomView(Context mContext){
    ...
    // View code goes here
    ...
    mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
}

@Override
public boolean onTouchEvent(MotionEvent ev) {
    // Let the ScaleGestureDetector inspect all events.
    mScaleDetector.onTouchEvent(ev);
    return true;
}

@Override
public void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    canvas.save();
    canvas.scale(mScaleFactor, mScaleFactor);
    ...
    // onDraw() code goes here
    ...
    canvas.restore();
}

private class ScaleListener
    extends ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        mScaleFactor *= detector.getScaleFactor();

        // Don't let the object get too small or too large.
        mScaleFactor = Math.max(0.1f, Math.min(mScaleFactor, 5.0f));

        invalidate();
        return true;
    }
}
```

## 更加複雜的縮放示例

這是本章節提供的`InteractiveChart`樣例中一個更複雜的示範。通過使用`ScaleGestureDetector`中的"span"(`getCurrentSpanX/Y`)和"focus"(`getFocusX/Y`)功能，`InteractiveChart`樣例同時支持滾動(平移)以及多指縮放。

```
@Override
private RectF mCurrentViewport =
    new RectF(AXIS_X_MIN, AXIS_Y_MIN, AXIS_X_MAX, AXIS_Y_MAX);
private Rect mContentRect;
private ScaleGestureDetector mScaleGestureDetector;
...
public boolean onTouchEvent(MotionEvent event) {
```

```

        boolean retVal = mScaleGestureDetector.onTouchEvent(event);
        retVal = mGestureDetector.onTouchEvent(event) || retVal;
        return retVal || super.onTouchEvent(event);
    }

    /**
     * The scale listener, used for handling multi-finger scale gestures.
     */
    private final ScaleGestureDetector.OnScaleGestureListener mScaleGestureListener
        = new ScaleGestureDetector.SimpleOnScaleGestureListener() {
    /**
     * This is the active focal point in terms of the viewport. Could be a local
     * variable but kept here to minimize per-frame allocations.
     */
    private PointF viewportFocus = new PointF();
    private float lastSpanX;
    private float lastSpanY;

    // Detects that new pointers are going down.
    @Override
    public boolean onScaleBegin(ScaleGestureDetector scaleGestureDetector) {
        lastSpanX = ScaleGestureDetectorCompat.
            getCurrentSpanX(scaleGestureDetector);
        lastSpanY = ScaleGestureDetectorCompat.
            getCurrentSpanY(scaleGestureDetector);
        return true;
    }

    @Override
    public boolean onScale(ScaleGestureDetector scaleGestureDetector) {

        float spanX = ScaleGestureDetectorCompat.
            getCurrentSpanX(scaleGestureDetector);
        float spanY = ScaleGestureDetectorCompat.
            getCurrentSpanY(scaleGestureDetector);

        float newWidth = lastSpanX / spanX * mCurrentViewport.width();
        float newHeight = lastSpanY / spanY * mCurrentViewport.height();

        float focusX = scaleGestureDetector.getFocusX();
        float focusY = scaleGestureDetector.getFocusY();
        // Makes sure that the chart point is within the chart region.
        // See the sample for the implementation of hitTest().
        hitTest(scaleGestureDetector.getFocusX(),
            scaleGestureDetector.getFocusY(),
            viewportFocus);

        mCurrentViewport.set(
            viewportFocus.x
                - newWidth * (focusX - mContentRect.left)
                / mContentRect.width(),
            viewportFocus.y
                - newHeight * (mContentRect.bottom - focusY)
                / mContentRect.height(),
            0,
            0);
        mCurrentViewport.right = mCurrentViewport.left + newWidth;
        mCurrentViewport.bottom = mCurrentViewport.top + newHeight;
        ...
        // Invalidates the View to update the display.
        ViewCompat.postInvalidateOnAnimation(InteractiveLineGraphView.this);

        lastSpanX = spanX;
        lastSpanY = spanY;
        return true;
    }
};


```

# 管理ViewGroup中的觸摸事件

編寫:Andrwyw - 原文:<http://developer.android.com/training/gestures/viewgroup.html>

因為很多時候event的目標是ViewGroup的孩子，並不是ViewGroup本身，所以處理ViewGroup中的觸摸事件需要特別注意。為了確保每個view能正確地接受到它們想要的觸摸事件，可以重載onInterceptTouchEvent()函數。

## 在ViewGroup中截獲觸摸事件

每當在ViewGroup的表面上檢測到一個觸摸事件，包括它子view的表面，onInterceptTouchEvent() 都會被調用。如果 onInterceptTouchEvent() 返回 true，MotionEvent 就被截獲了，這表示它不會被傳遞給孩子了，而是傳遞給該父view自身的onTouchEvent()方法。

onInterceptTouchEvent() 方法讓父view能夠在它的子view之前處理觸摸事件。如果你讓 onInterceptTouchEvent() 返回 true，則之前處理觸摸事件的子view會收到ACTION\_CANCEL事件，並且該點之後的事件會被發送給該父view自身的 onTouchEvent() 函數，進行常規處理。onInterceptTouchEvent() 也可以返回 false，這樣事件沿view層級分發到目標前，父view可以簡單地觀察該事件。這裏的目標是指，通過 onTouchEvent() 處理消息事件的view。

接下來的代碼段中，My ViewGroup 繼承自ViewGroup。My ViewGroup 有多個子view。如果你水平地拖動手指經過某個子view，該子view不會接收到觸摸事件，而是 My ViewGroup 處理這些觸摸事件來滾動它的內容。然而，如果你點擊子view 中的button，或垂直地滾動子view，則父view不會截獲這些觸摸事件，因為子view本就是預定目標。在這些情況下，onInterceptTouchEvent() 應該返回 false，My ViewGroup 的 onTouchEvent() 也不會被調用。

```
public class My ViewGroup extends ViewGroup {  
    private int mTouchSlop;  
  
    ...  
  
    ViewConfiguration vc = ViewConfiguration.get(view.getContext());  
    mTouchSlop = vc.getScaledTouchSlop();  
  
    ...  
  
    @Override  
    public boolean onInterceptTouchEvent(MotionEvent ev) {  
        /*  
         * This method JUST determines whether we want to intercept the motion.  
         * If we return true, onTouchEvent will be called and we do the actual  
         * scrolling there.  
         */  
  
        final int action = MotionEventCompat.getActionMasked(ev);  
  
        // Always handle the case of the touch gesture being complete.  
        if (action == MotionEvent.ACTION_CANCEL || action == MotionEvent.ACTION_UP) {  
            // Release the scroll.  
            mIsScrolling = false;  
            return false; // Do not intercept touch event, let the child handle it  
        }  
  
        switch (action) {  
            case MotionEvent.ACTION_MOVE: {  
                if (mIsScrolling) {  
                    // We're currently scrolling, so yes, intercept the  
                    // touch event!  
                    return true;  
                }  
            }  
        }  
    }  
}
```

```

        // If the user has dragged her finger horizontally more than
        // the touch slop, start the scroll

        // left as an exercise for the reader
        final int xDiff = calculateDistanceX(ev);

        // Touch slop should be calculated using ViewConfiguration
        // constants.
        if (xDiff > mTouchSlop) {
            // Start scrolling!
            mIsScrolling = true;
            return true;
        }
        break;
    }
    ...
}

// In general, we don't want to intercept touch events. They should be
// handled by the child view.
return false;
}

@Override
public boolean onTouchEvent(MotionEvent ev) {
    // Here we actually handle the touch event (e.g. if the action is ACTION_MOVE,
    // scroll this container).
    // This method will only be called if the touch event was intercepted in
    // onInterceptTouchEvent
    ...
}
}

```

注意`ViewGroup`也提供了`requestDisallowInterceptTouchEvent()`方法。當子view不想該父view和祖先view通過`onInterceptTouchEvent()`截獲它的觸摸事件時，可調用`ViewGroup`的該方法。

## 使用`ViewConfiguration`的常量

上面的代碼段中使用了當前的`ViewConfiguration`來初始化`mTouchSlop`變量。你可以使用`ViewConfiguration`類來獲取Android系統常用的一些距離、速度、時間值。

“Touch slop” 是指在被識別為移動的手勢前，用戶觸摸可移動的那一段像素距離。Touch slop通常用來預防用戶在做一些其他觸摸操作時，出現意外地滑動，例如觸摸屏幕上的元素。

另外兩個常用的`ViewConfiguration`函數是`getScaledMinimumFlingVelocity()`和`getScaledMaximumFlingVelocity()`。這兩個函數會返回初始化一個快速滑動(fling)的最小、最大速度（分別地），以像素每秒為測量單位。如：

```

ViewConfiguration vc = ViewConfiguration.get(view.getContext());
private int mSlop = vc.getScaledTouchSlop();
private int mMinFlingVelocity = vc.getScaledMinimumFlingVelocity();
private int mMaxFlingVelocity = vc.getScaledMaximumFlingVelocity();

...
case MotionEvent.ACTION_MOVE: {
    ...
    float deltaX = motionEvent.getRawX() - mDownX;
    if (Math.abs(deltaX) > mSlop) {
        // A swipe occurred, do something
    }
}

...
case MotionEvent.ACTION_UP: {
    ...
}

```

```
} if (mMinFlingVelocity
```

## 擴展view的可觸摸區域

Android提供了[TouchDelegate](#)類，讓父view擴展子view的可觸摸區域，擴展後的區域可超過子view本身的邊界。這在子view很小，但需要一個更大的觸摸區域時非常有用。如果需要，你也可以使用這種方式來實現對子view的觸摸區域的收縮。

在下面的例子中，`ImageButton`對象是所謂的"delegate view"（是指觸摸區域將被父view擴展的那個子view）。這是佈局文件：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/parent_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity" >  
  
    <ImageButton android:id="@+id/button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:background="@null"  
        android:src="@drawable/icon" />  
  
</RelativeLayout>
```

下面的代碼段做了這樣幾件事：

- 獲得父view對象並發送一段Runnable到UI線程。這會確保父view在調用getHitRect()函數前會佈局它的子view。getHitRect()函數會獲得子view在父view座標系中的點擊矩形（觸摸區域）。
  - 找到ImageButton子view，然後調用getHitRect()來獲得它的觸摸區域的邊界。
  - 擴展ImageButton的點擊矩形的邊界。
  - 實例化一個TouchDelegate對象，並把擴展過的點擊矩形和ImageButton子view作為參數傳遞給它。
  - 設置父view的TouchDelegate，這樣在touch delegate邊界內的點擊就會傳遞到該子view上。

在ImageButton子view的touch delegate範圍內，父view會接收到所有的觸摸事件。如果觸摸事件發生在子view自身的點擊矩形中，父view會把觸摸事件交給子view處理。

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        // Get the parent view  
        View parentView = findViewById(R.id.parent_layout);  
  
        parentView.post(new Runnable() {  
            // Post in the parent's message queue to make sure the parent  
            // lays out its children before you call getHitRect()  
            @Override  
            public void run() {  
                // The bounds for the delegate view (an ImageButton  
                // in this example)  
                Rect delegateArea = new Rect();  
                ImageButton myButton = (ImageButton) findViewById(R.id.button);  
                myButton.setEnabled(true);  
                myButton.setOnClickListener(new View.OnClickListener() {  
                    @Override  
                    public void onClick(View view) {  
                        Toast.makeText(MainActivity.this,  
                            "Clicked button!",  
                            Toast.LENGTH_SHORT).show();  
                    }  
                });  
            }  
        });  
    }  
}
```

```
        "Touch occurred within ImageButton touch region.",
        Toast.LENGTH_SHORT).show();
    });
}

// The hit rectangle for the ImageButton
myButton.getHitRect(delegateArea);

// Extend the touch area of the ImageButton beyond its bounds
// on the right and bottom.
delegateArea.right += 100;
delegateArea.bottom += 100;

// Instantiate a TouchDelegate.
// "delegateArea" is the bounds in local coordinates of
// the containing view to be mapped to the delegate view.
// "myButton" is the child view that should receive motion
// events.
TouchDelegate touchDelegate = new TouchDelegate(delegateArea,
    myButton);

// Sets the TouchDelegate on the parent view, such that touches
// within the touch delegate bounds are routed to the child.
if (View.class.isInstance(myButton.getParent())) {
    ((View) myButton.getParent()).setTouchDelegate(touchDelegate);
}
}
});
}
}
```

# 處理鍵盤輸入

---

編寫:zhaochunqi - 原文:<http://developer.android.com/training/keyboard-input/index.html>

Android 系統展示了一個屏幕上的鍵盤－被稱為軟輸入法－當一個文本域在UI中接收到聚焦時。為了提供最好的用戶體驗，你可以指定你期望的輸入類型(電話號碼或Email地址)和輸入法的表現形式(是否需要自動糾正拼寫錯誤)。

除了屏幕上的輸入法，Android也支持實體鍵盤，所以充分利用可能的外接鍵盤來優化用戶的交互體驗是很重要的。

## Lessons

---

- [指定輸入法](#)

學習如何表現特定的虛擬輸入法，如為電話號碼、網址和其他一些格式所設計的。同樣應該學習如何指定建議的操作如確定(Done)或者下一步(Next)。

- [處理輸入法的顯示](#)

學習如何指定合適展示軟鍵盤輸入法，如何讓你的佈局適合因為輸入法而減少的屏幕空間。

- [支持鍵盤導航](#)

學習如何驗證用戶能夠使用鍵盤導航到你的應用以及如何對導航順序做出相應的改變。

- [處理鍵盤行為](#)

學習如何對用戶的鍵盤輸入進行迴應。

# 指定輸入法類型

編寫:zhaochunqi - 原文:<http://developer.android.com/training/keyboard-input/style.html>

每個文本域期待特定的文本類型，如Email，電話號碼，或者純文本。為應用中的每一個文本域指定特定的輸入類型以便系統展示更為合適的軟鍵盤輸入法(比如屏幕上鍵盤)是很重要的。

## 指定鍵盤類型

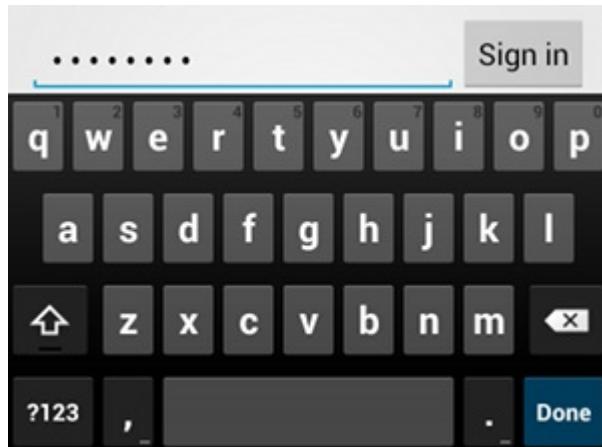
你總是可以為你的文本域定義輸入法通過添加`android:inputType` 屬性到 `<EditText>` 元素中。

舉例來說，如果你想要一個為輸入電話號碼的輸入法，使用"phone"值:



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

或着如果文本域是密碼，使用"textPassword"值來隱藏用戶的輸入:



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

有幾種可供選擇的值在`android:inputType`屬性中記錄，一些值可以組合起來實現特定的輸入法表現和附加的行爲。

## 開啓拼寫建議和其他的行爲

`android:inputType`屬性允許你為輸入法指定不同的行爲。最為重要的是，如果你的文本域是為基本的文本輸入(如短消息)，你應該使用"textAutoCorrect"來開啓拼寫檢查。

你可以組合不同的行爲和輸入法形式通過`textAutoCorrect`這個屬性。如：如何創建一個文本域句子單詞的首字母答謝



並開啓拼寫檢查：

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

## 指定輸入法的行爲

多數的軟鍵盤會在底部角落裏為用戶提供一個合適的動作按鈕來觸發當前文本域的操作。默認情況下，系統使用下一步(Next)或者確認(DONE)除非你的文本域允許多行(如`android:inputType="textMultiLine"`)，這種情況下，動作按鈕就是回車換行。然而，你可以制定額外的動作一邊更適合你的文本域，比如SEND和GO。

指定特定的動作按鈕，將`android:imeOptions`屬性的值設為"actionSend"或"actionSearch"。如：



```
<EditText  
    android:id="@+id/search"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/search_hint"  
    android:inputType="text"  
    android:imeOptions="actionSend" />
```

然後你可以通過為`EditText`定義`TextView.OnEditorActionListener`來監聽動作按鈕的啓動。在監聽器中，對輸入法編輯器對合適的迴應的動作ID對應在`EditorInfo`類中，如`IME_ACTION_SEND`。例如：

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

# 處理輸入法可見性

編寫:zhaochunqi - 原文:<http://developer.android.com/training/keyboard-input/visibility.html>

當輸入焦點移入或移出可編輯當文本域時，Android會相應的顯示或隱藏輸入法(如屏幕輸入法)。系統也會決定你的輸入法上方UI和文本域的顯示。舉例來說，當屏幕上豎直空間被壓縮時，文本域可能填充所有的輸入法上方的空間。對於多數的應用來說，這些默認的行爲基本就足夠了。

然而，在一些事例中，你可能會想要更加直接的控制輸入法的顯示，指定你的佈局在在輸入法顯示時候的表現。這節課會向你解釋如何控制和迴應輸入法的可見性。

## 在Activity啓動時顯示輸入法

儘管Android會在Activity啓動時給予第一個文本域焦點，但是並不會顯示輸入法。因為進入文本可能並不是activity中的首要任務，所以這是很合理的。可是，如果進入文本確實需要是首要的任務(如登錄界面)，可能需要用到輸入法默認顯示。

為了在activity啓動時展示輸入法，添加`android:windowSoftInputMode` 屬性到`<activity>`元素中，使用 "stateVisible"，如下：

```
<application ... >
    <activity
        android:windowSoftInputMode="stateVisible" ... >
        ...
    </activity>
    ...
</application>
```

注意：如果用戶設備有一個實體鍵盤，軟鍵盤輸入法可能不顯示。

## 需要時顯示輸入法

如果在activity生命週期中有一個方法在想要確保輸入法是可見的，可以使用 `InputMethodManager` 來實現。

舉例來說，下面的方法調用了一個需要用戶填寫文本的View，調用了`requestFocus()` 來獲取焦點，然後`showSoftInput()`來打開輸入法。

```
public void showSoftKeyboard(View view) {
    if (view.requestFocus()) {
        InputMethodManager imm = (InputMethodManager)
            getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.showSoftInput(view, InputMethodManager.SHOW_IMPLICIT);
    }
}
```

注意：一旦輸入法設定可見了，你不應該用程序來隱藏。系統會在用戶結束文本域的任務的時候隱藏，或者可以使用系統控制(如返回鍵)來隱藏。

## 指定你的UI迴應方式

當你的輸入法顯示在屏幕上，減少了UI中的可用空間。系統會為你的UI的可見區的UI做調整但是可能並非很正確。為了確保你應用的最佳表現，你應該在UI的剩餘空間中展示你想要展示的系統界面。

為了聲明你在[activity](#)中的合適的對待，使用 `android:windowSoftInputMode` 屬性在你的清單文件中的[`<activity>`](#)元素使用某個"adjust"值。

舉例來說，為了確保系統會在可用空間中重新調整佈局的大小。為了確保你所有的佈局內容都是可用的(儘管可能需要滑動)使用"adjustResize":

```
<application ... >
    <activity
        android:windowSoftInputMode="adjustResize" ... >
        ...
    </activity>
    ...
</application>
```

你可以結合調整和使用上面的[初始輸入法可見性](#)來指定：

```
<activity
    android:windowSoftInputMode="stateVisible|adjustResize" ... >
    ...
</activity>
```

如果你的UI中包含用戶可能需要在文本輸入時立即執行的事情，那麼使用"adjustResize"時很重要的。例如，如果你使用相對佈局在屏幕底部放置一個按鈕，使用"adjustResize"來重新調整大小，使得按鈕欄出現在輸入法上方。

# 兼容鍵盤導航

編寫:zhaochunqi - 原文:<http://developer.android.com/training/keyboard-input/navigation.html>

除了軟鍵盤輸入法(如屏幕鍵盤)以外，Android支持物理鍵盤連接到設備上。一個鍵盤不僅提供為文本輸入提供方便地模式，而提供一個合適的方法來導航和與應用交互。儘管多數的手持設備像手機使用觸摸作為主要的交互方式，平板和一些其他的設備正在逐步流行起來，許多用戶喜歡外接鍵盤。

隨着更多的Android設備提供這種體驗，為你的應用添加通過鍵盤進行交互的支持優化是很重要的。這節課介紹了怎樣為鍵盤導航提供更好的支持。

注意：對那些沒有使用可見導航提示的應用來說，在應用中支持方向性的導航對於應用的可用性也是很重要的。

完全支持方向性導航在你的應用中還可以幫助你使用諸如uiautomator進行自動化用戶界面測試化。

## 測試你的應用

可能用戶已經可以在你的應用中使用鍵盤導航了，因為Android系統默認開啓了大多是必要的行為。

所有由Android framework(如Button和EditText) 提供的交互 widgets是可獲得焦點的。這意味着用戶可以使手控設備如D-pad或鍵盤或widgets發亮或者其他一些獲得輸入焦點的行為改變外觀。

為了測試你的應用：

1. 在實體鍵盤的設備上安裝你的應用

如果你沒有帶實體鍵盤的設備，連接一個藍牙鍵盤或者USB鍵盤(儘管並不是所有的設備都支持USB連接)

你還可以使用Android虛擬機

- i. 在AVD管理器中，或者點擊New Device或者選擇一個已存在的文檔點擊Clone.
  - ii. 在出現的窗口中，確保鍵盤和D-pad開啓。
2. 為了驗證你的應用，只是用Tab鍵來進行UI導航，確保每一個UI控制的焦點如預期的一致。查看每個不在預期焦點的實例。
  3. 從頭開始，使用方向鍵(鍵盤上的箭頭鍵)來控制你應用的導航。從每一個在你UI中的焦點元素，按上、下、左、右。查看每個不在預期焦點的實例

如果你遇到任何使用Tab鍵或方向鍵不如預期，在佈局文件中指定應該的焦點，如下面幾部分所討論的。

## 處理Tab導航

當一個用戶使用鍵盤上到Tab鍵導航到你的應用時，系統會在元素之間傳遞焦點，取決於他們在佈局文件中的顯示順序。如果你使用相對佈局，在屏幕上的元素順序與文件中元素的順序不一致，那樣你可能需要手動的指定焦點順序。

舉例來說，在下面的佈局文件中，兩個對其右邊的按鈕和一個對齊第二個按鈕導航。為了把焦點從第一個按鈕傳遞到文本域，然後再傳遞到第二個按鈕，佈局文件需要清楚的為每一個可聚焦的元素定義焦點順序，使用屬性android:nextFocusForward：

```
<RelativeLayout ...>
```

```
<Button  
    android:id="@+id/button1"  
    android:layout_alignParentTop="true"  
    android:layout_alignParentRight="true"  
    android:nextFocusForward="@+id/editText1"  
    ... />  
<Button  
    android:id="@+id/button2"  
    android:layout_below="@+id/button1"  
    android:nextFocusForward="@+id/button1"  
    ... />  
<EditText  
    android:id="@+id/editText1"  
    android:layout_alignBottom="@+id/button2"  
    android:layout_toLeftOf="@+id/button2"  
    android:nextFocusForward="@+id/button2"  
    ... />  
...  
</RelativeLayout>
```

現在焦點從button1到button2再到editText1改成了合適的按照出現在屏幕上順序到從button1到editText1再到button2.

## 處理直接的導航

用戶也能夠使用鍵盤上的方向鍵在你的app中導航(這種行爲與在D-pad和軌跡球中的導航一致)。系統提供了一個最佳猜測對於哪個視圖應該給予焦點在一個基於方向的基於佈局文件的在屏幕上展現的佈局。然而有時，系統會猜測錯誤。

如果你的系統沒有傳遞焦點到合適的視圖中在導航到一個給定的視圖中的時候，指定一個視圖使用如下的屬性：

- android:nextFocusUp
- android:nextFocusDown
- android:nextFocusLeft
- android:nextFocusRight

每一個屬性設計了下一個接受焦點的視圖當用戶導航到那個方向時，如指定當view的ID一樣。舉例來說：

```
<Button  
    android:id="@+id/button1"  
    android:nextFocusRight="@+id/button2"  
    android:nextFocusDown="@+id/editText1"  
    ... />  
<Button  
    android:id="@+id/button2"  
    android:nextFocusLeft="@+id/button1"  
    android:nextFocusDown="@+id/editText1"  
    ... />  
<EditText  
    android:id="@+id/editText1"  
    android:nextFocusUp="@+id/button1"  
    ... />
```

# 處理按鍵動作

編寫:[zhaochunqi](#) - 原文:<http://developer.android.com/training/keyboard-input/commands.html>

當預估給予可編輯當文本域焦點時，如一個EditText元素，而且用戶擁有一個實體鍵盤連接，所有當輸入由系統處理。然而如果你想接管或直接處理鍵盤輸入鍵盤操作，通過實現接口KeyEvent.Callback的回調方法，如onKeyDown()和onKeyMultiple()。

Activity和View類都實現了KeyEvent.Callback的接口，所以通常你只需要在這些重寫回調方法來適當的擴展這些類。

注意：當使用KeyEvent類和相關的API處理鍵盤事件時，你期望的應該是只從實體鍵盤中接收。你永遠不應該指望從一個軟鍵盤(如屏幕鍵盤)來接受點擊事件。

## 處理單個按鍵點擊事件

處理單個的按鍵點擊，實現合適的onKeyDown()或onKeyUp()。通常，你使用onKeyUp()來確保你只接收一個事件。如果用戶點擊並按住按鈕不放，onKeyDown()會被調用多次。

舉例，這是一個對一些按鍵控制遊戲的實現：

```
@Override  
public boolean onKeyDown(int keyCode, KeyEvent event) {  
    switch (keyCode) {  
        case KeyEvent.KEYCODE_D:  
            moveShip(MOVE_LEFT);  
            return true;  
        case KeyEvent.KEYCODE_F:  
            moveShip(MOVE_RIGHT);  
            return true;  
        case KeyEvent.KEYCODE_J:  
            fireMachineGun();  
            return true;  
        case KeyEvent.KEYCODE_K:  
            fireMissile();  
            return true;  
        default:  
            return super.onKeyDown(keyCode, event);  
    }  
}
```

## 處理修飾鍵

為了對修飾鍵進行迴應如一個組合Shift和Control修飾鍵，你可以查詢KeyEvent傳遞到回調方法。一些方法提供一些信息關於修飾鍵如getModifiers()和getMetaState()。然而，最簡單的解決方案時檢查你關心的按鍵是否被按下了的方法，如isShiftPressed()和isCtrlPressed()。

例如，有一個onKeyDown()的實現，當Shift鍵和一個其他當鍵按下當時候做一些額外的處理：

```
@Override  
public boolean onKeyDown(int keyCode, KeyEvent event) {  
    switch (keyCode) {  
        ...  
        case KeyEvent.KEYCODE_J:  
            if (event.isShiftPressed()) {
```

```
        fireLaser();
    } else {
        fireMachineGun();
    }
    return true;
case KeyEvent.KEYCODE_K:
    if (event.isShiftPressed()) {
        fireSeekingMissle();
    } else {
        fireMissile();
    }
    return true;
default:
    return super.onKeyUp(keyCode, event);
}
}
```

# 兼容遊戲控制器

---

編寫:heray1990 - 原文:<http://developer.android.com/training/game-controllers/index.html>

你可以通過支持用戶的遊戲控制器來增強用戶體驗。Android的框架提供了APIs來檢測和處理遊戲控制器的用戶輸入。

這節課介紹瞭如何使你的遊戲在不同的API levels(API level 9或者更高)穩定地工作。還介紹瞭如何通過在你的App支持多個遊戲控制器來增強用戶體驗。

## Lessons

---

### 處理控制器輸入動作

學習如何處理遊戲控制器常用的輸入元素，包括十字鍵按鈕、遊戲鍵盤和搖桿。

### 支持不同的Android系統版本

學習如何使遊戲控制器在運行不同Android系統版本的設備上保持行為一致。

### 支持多個遊戲控制器

學習如何檢測和使用多個同時連接的遊戲控制器。

# 處理控制器輸入動作

編寫:heray1990 - 原文:<http://developer.android.com/training/game-controllers/controller-input.html>

在系統層面上，Android會以Android鍵值和座標值的形式來報告來自遊戲控制器的輸入事件。在我們的遊戲應用裏，我們可以接收這些鍵值和座標值，並將它們轉化成特定的遊戲行爲。

當玩家將一個遊戲控制器通過有線連接或者無線配對到基於Android的設備時，系統會自動檢測控制器，將它設置成輸入設備並且開始報告它的輸入事件。我們的遊戲應用可以通過在活動的Activity或者聚焦的View裏調用下面這些回調方法來接收上述輸入事件(要麼在Activity，要麼在View中實現實現這些回調方法，不要兩個地方都實現回調)。

- From Activity
  - `dispatchGenericMotionEvent(android.view.MotionEvent)`
    - 處理一般的運動事件，如搖動搖桿
  - `dispatchKeyEvent(android.view.KeyEvent)`
    - 處理按鍵事件，如按下或者釋放遊戲鍵盤的按鍵或者十字方向鍵。
- From View
  - `onGenericMotionEvent(android.view.MotionEvent)`
    - 處理一般的運動事件，如搖動搖桿
  - `onKeyDown(int, android.view.KeyEvent)`
    - 處理按下一個按鍵的事件，如按下遊戲鍵盤的按鍵或者十字方向鍵。
  - `onKeyUp(int, android.view.KeyEvent)`
    - 處理釋放一個按鍵的事件，如釋放遊戲鍵盤的按鍵或者十字方向鍵。

建議的方法是從與用戶交互的View對象捕獲事件。請查看下面回調提供的對象，從而獲取關於接收到的輸入事件的類型：

**KeyEvent**：描述方向按鍵和遊戲按鍵事件的對象。按鍵事件伴隨着一個表示特定按鍵觸發的按鍵碼值(key code)，如DPAD\_DOWN或者BUTTON\_A。我們可以通過調用`getKeyCode()`或者從按鍵事件回調方法(如`onKeyDown()`)來獲得按鍵碼值。

**MotionEvent**：描述搖桿和肩鍵運動的輸入。動作事件伴隨着一個動作碼(action code)和一系列座標值(axis values)。動作碼表示出現變化的狀態，例如搖動一個搖桿。座標值描述了位置和其它運動屬性，例如AXIS\_X或者AXIS\_RTRIGGER。我們可以通過調用`getAction()`來獲得動作碼，通過調用`getAxisValue()`來獲得座標值。

這節課主要介紹如何通過上述的View回調方法和處理KeyEvent和MotionEvent對象來處理常用控制器(遊戲鍵盤按鍵、方向按鍵和搖桿)的輸入。

## 驗證遊戲控制器是否已連接

在報告輸入事件的時候，Android不會區分遊戲控制器事件與非遊戲控制器事件。例如，一個觸屏動作會產生一個表示觸摸表面上X座標的AXIS\_X，但是一個搖桿動作產生的AXIS\_X則表示搖桿水平移動的位置。如果我們的遊戲關注遊戲控制器的輸入，那麼我們應該首先檢測事件相應的來源類型。

通過調用`getSources()`來獲得設備上支持的輸入類型的位字段，來驗證一個已連接的輸入設備是一個遊戲控制器。我們可以測試以查看下面的字段是否被設置：

- `SOURCE_GAMEPAD`源類型表示輸入設備有遊戲手柄按鍵(如，BUTTON\_A)。注意雖然一般的遊戲手柄都會有方向控制鍵，但是這個源類型並不代表遊戲控制器具有十字方向鍵。
- `SOURCE_DPAD`源類型表示輸入設備有十字方向鍵(如，DPAD\_UP)。

- SOURCE\_JOYSTICK 源類型表示輸入設備有遙控杆(如，會用 `AXIS_X` 和 `AXIS_Y` 記錄動作的搖桿)。

下面的一小段代碼介紹了一個helper方法，它的作用是讓你檢驗已接入的輸入設備是否是遊戲控制器。如果檢測到是遊戲控制器，那麼這個方法會獲取到遊戲控制器的設備ID。然後，我們應該將每個設備ID與遊戲中的玩家關聯起來，並且單獨處理每個已接入的玩家的遊戲操作。想更詳細地瞭解關於在一臺Android設備中同時支持多個遊戲控制器的方法，請見[支持多個遊戲控制器](#)。

```
public ArrayList<String> getGameControllerIds() {
    ArrayList<String> gameControllerDeviceIds = new ArrayList<String>();
    int[] deviceIds = InputDevice.getDeviceIds();
    for (int deviceId : deviceIds) {
        InputDevice dev = InputDevice.getDevice(deviceId);
        int sources = dev.getSources();

        // Verify that the device has gamepad buttons, control sticks, or both.
        if (((sources & InputDevice.SOURCE_GAMEPAD) == InputDevice.SOURCE_GAMEPAD)
            || ((sources & InputDevice.SOURCE_JOYSTICK)
                == InputDevice.SOURCE_JOYSTICK)) {
            // This device is a game controller. Store its device ID.
            if (!gameControllerDeviceIds.contains(deviceId)) {
                gameControllerDeviceIds.add(deviceId);
            }
        }
    }
    return gameControllerDeviceIds;
}
```

另外，我們會想去檢查已接入的遊戲控制器個體的輸入性能。這種檢查在某些場合會很有用，例如，我們想遊戲只用到遊戲“知道”的物理操控。

用下面這些方法檢測一個遊戲控制器是否支持一個特定的按鍵碼或者座標碼：

- 在Android 4.4(API level 19)或者更高的系統中，調用 `hasKeys(int)` 來確定遊戲控制器是否支持一個按鍵碼。
- 在Android 3.1(API level 12)或者更高的系統中，首先調用 `getMotionRanges()`，然後在每個返回的 `InputDevice.MotionRange` 對象中調用 `getAxis()` 來獲得座標ID。這樣就可以得到遊戲控制器支持的所有可用座標軸。

## 處理遊戲手柄按鍵

Figure 1 介紹了Android如何將按鍵碼和座標值映射到實際的遊戲手柄上。

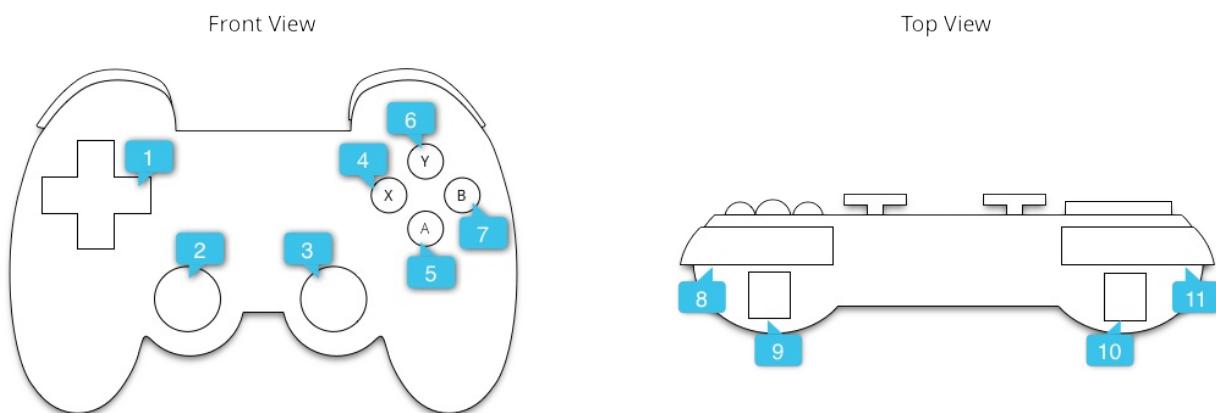


Figure 1. 一個常用的遊戲手柄的外形

上圖的標註對應下面的內容：

1. `AXIS_HAT_X`, `AXIS_HAT_Y`, `DPAD_UP`, `DPAD_DOWN`, `DPAD_LEFT`, `DPAD_RIGHT`
2. `AXIS_X`, `AXIS_Y`, `BUTTON_THUMBL`
3. `AXIS_Z`, `AXIS_RZ`, `BUTTON_THUMBR`
4. `BUTTON_X`
5. `BUTTON_A`
6. `BUTTON_Y`
7. `BUTTON_B`
8. `BUTTON_R1`
9. `AXIS_RTRIGGER`, `AXIS_THROTTLE`
10. `AXIS_LTRIGGER`, `AXIS_BRAKE`
11. `BUTTON_L1`

遊戲手柄產生的通用的按鍵碼包括`BUTTON_A`、`BUTTON_B`、`BUTTON_SELECT`和`BUTTON_START`。當按下十字方向鍵的中間交叉按鍵時，一些遊戲控制器會觸發`DPAD_CENTER`按鍵碼。我們的遊戲可以通過調用`getKeyCode()`或者從按鍵事件回調(如`onKeyDown()`)得到按鍵碼。如果一個事件與我們的遊戲相關，那麼將其處理成一個遊戲動作。Table 1列出供大多數通用遊戲手柄的按鈕使用的推薦的遊戲動作。

Table 1. 供遊戲手柄使用的推薦的遊戲動作

遊戲動作	按鍵碼
在主菜單中啓動遊戲，或者在遊戲過程中暫停/取消暫停	<code>BUTTON_START</code> *
顯示菜單	<code>BUTTON_SELECT</code> * 和 <code>KEYCODE_MENU</code> *
跟Android導航設計指導中的Back導航行爲一樣	<code>KEYCODE_BACK</code>
返回到菜單中上一項	<code>BUTTON_B</code>
確認選擇，或者執行主要的遊戲動作	<code>BUTTON_A</code> 和 <code>DPAD_CENTER</code>

\* 我們的遊戲不應該依賴於Start、Select或者Menu按鍵的存在。

Tip:

# 支持不同的Android系統版本

---

| 編寫: - 原文:

待認領進行編寫，有意向的小夥伴，可以直接修改對應的markdown文件，進行提交！

# 支持多個控制器

---

| 編寫: - 原文:

待認領進行編寫，有意向的小夥伴，可以直修改對應的markdown文件，進行提交！

# Android後臺任務

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/best-background.html>

These classes show you how to run jobs in the background to boost your application's performance and minimize its drain on the battery.

## Running in a Background Service

How to improve UI performance and responsiveness by sending work to a Service running in the background

## Loading Data in the Background

How to use CursorLoader to query data without affecting UI responsiveness.

## Managing Device Awake State

How to use repeating alarms and wake locks to run background jobs.

# 在IntentService中執行後臺任務

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/run-background-service/index.html>

除非你特別指定，否則大部分在前臺UI界面上的操作都執行在一個叫做UI Thread的特殊線程中。這可能會導致某些問題，因為耗時操作可能會干擾界面的響應性能。這會惹惱用戶，甚至可以導致系統錯誤。為了避免這樣的問題，Android Framework提供了幾個類，用來幫助你把那些耗時操作移動到後臺線程中執行。那些類中最常用的就是IntentService。

這一章節會講到如何實現一個IntentService，向它發送任務並反饋它的結果給其他模塊。

## Lessons

---

- [Creating a Background Service:創建IntentService](#)

學習如何創建一個IntentService。

- [Sending Work Requests to the Background Service:發送任務請求到IntentService](#)

學習如何發送工作任務到IntentService。

- [Reporting Work Status:報告後臺任務的執行狀態](#)

學習如何使用Intent與LocalBroadcastManager在Activit與IntentService之間進行交互。

# 創建後臺服務

編寫:kesenhoo - 原文:<http://developer.android.com/training/run-background-service/create-service.html>

IntentService為在單個後臺線程執行一個操作提供了一種直接的實現方式。它可以處理一個長時間操作的任務並確保不影響到UI的響應性。而且IntentService的執行並不受UI的生命週期的影響。所以繼續運行的情況下將關閉一個AsyncTask。

IntentService有下面幾個侷限性：

- 不可以直接和UI做交互。為了把他執行的結果體現在UI上，需要發送給Activity。
- 工作任務隊列是順序執行的，如果一個任務正在IntentService中執行，此時你再發送一個任務請求，這個任務會一直等待直到前面一個任務執行完畢。
- 正在執行的任務無法打斷。

然而，在大多數情況下，IntentService都是簡單後臺任務操作的理想選擇。

這節課會演示如何創建繼承的IntentService。同樣也會演示如何創建必須實現的回調onHandleIntent()。最後，還會解釋如何在manifest文件中定義這個IntentService。

## 1)創建IntentService

為你的app創建一個IntentService組件，需要定義一個類，並繼承IntentService類，在裏面重寫onHandleIntent()方法，如下所示：

```
public class RSSPullService extends IntentService {  
    @Override  
    protected void onHandleIntent(Intent workIntent) {  
        // 從傳入的intent獲取數據  
        String dataString = workIntent.getDataString();  
        ...  
        // 根據dataString的內容在這裏進行操作  
        ...  
    }  
}
```

注意一個普通Service組件的其他回調，例如 onStartCommand() 會被IntentService自動調用。在IntentService中，要避免重寫那些回調。

## 2)在Manifest文件中定義IntentService

IntentService需要在manifest文件添加相應的條目，將此條目 <service> 作為 <application> 元素的子元素下進行定義，如下所示：

```
<application  
    android:icon="@drawable/icon"  
    android:label="@string/app_name">  
    ...  
    <!--  
        因為android:exported 被設置為false，該服務只能在本應用中使用  
    -->  
    <service
```

```
    android:name=".RSSPullService"
    android:exported="false"/>
...
<application/>
```

android:name 屬性指明瞭IntentService的名字。

注意 <service> 標籤並沒有包含任何intent filter。因為發送任務給IntentService的Activity需要使用顯式Intent，所以不需要filter。這也意味着只有在同一個app或者其他使用同一個UserID的組件才能夠訪問到這個Service。

至此，你已經有了一個基本的IntentService類，你可以通過Intent對象向它發送操作請求。構造這些對象以及發送它們到你的IntentService的方式，將在接下來的課程中描述。

# 向後臺服務發送任務請求

編寫:kesenhoo - 原文:<http://developer.android.com/training/run-background-service/send-request.html>

前一篇文章演示瞭如何創建一個IntentService類。這次會演示如何通過發送一個Intent來觸發IntentService執行任務。這個Intent可以傳遞一些數據給IntentService。可以在Activity或者Fragment的任何時間點發送這個Intent。

## 創建任務請求（Work Request）並發送到IntentService

為了創建一個任務請求並發送到IntentService。需要先創建一個explicit Intent，並將請求數據添加到intent，然後通過調用 `startService()` 函數把工作請求數據發送到IntentService。

下面是代碼示例：

- 創建一個新的顯式的Intent用來啓動IntentService。

```
/*
 * 創建一個新的Intent來啓動RSSPullService，通過intent的"data"屬性傳入一個URI
 */
mServiceIntent = new Intent(getActivity(), RSSPullService.class);
mServiceIntent.setData(Uri.parse(dataUrl));
```

- 執行`startService()`

```
// 啓動IntentService
getActivity().startService(mServiceIntent);
```

注意可以在Activity或者Fragment的任何位置發送任務請求。例如，如果你先獲取用戶輸入，您可以從一個回調發送請求，響應按鈕單擊或類似的手勢。

一旦執行了`startService()`，IntentService在自己本身的`onHandleIntent()`方法裏面開始執行這個任務，然後停止。

下一步是如何把工作任務的執行結果返回給發送任務的Activity或者Fragment。下節課會演示如何使用BroadcastReceiver來完成這個任務。

# 報告任務執行狀態

編寫:kesenhoo - 原文:<http://developer.android.com/training/run-background-service/report-status.html>

這章節會演示如何回傳IntentService中執行的任務狀態與結果給發送方。例如，回傳任務的狀態給Activity並進行更新UI。推薦的方式是使用LocalBroadcastManager，這個組件可以限制broadcast Intent只在自己的App中進行傳遞。

## 利用IntentService 發送任務狀態

為了在IntentService中向其他組件發送任務狀態，首先創建一個Intent並在data字段中包含需要傳遞的信息。作為一個可選項，還可以給這個Intent添加一個action與data URI。

下一步，通過執行LocalBroadcastManager.sendBroadcast() )來發送Intent。Intent被發送到任何有註冊接受它的組件中。為了獲取到LocalBroadcastManager的實例，可以執行getInstance().代碼示例如下：

```
public final class Constants {  
    ...  
    // 定義一個自定義的Intent動作名  
    public static final String BROADCAST_ACTION =  
        "com.example.android.threadsample.BROADCAST";  
    ...  
  
    // 定義一個附加鍵名來表示狀態應用於包裝在一個Intent中  
    public static final String EXTENDED_DATA_STATUS =  
        "com.example.android.threadsample.STATUS";  
    ...  
}  
public class RSSPullService extends IntentService {  
    ...  
    /*  
     * 創建一個新的包含名Uri的對象  
     * BROADCAST_ACTION是一個自定義的Intent動作  
     *  
     */  
    Intent localIntent =  
        new Intent(Constants.BROADCAST_ACTION)  
            // 把status放到Intent中  
            .putExtra(Constants.EXTENDED_DATA_STATUS, status);  
    // 廣播Intent給應用中的接收器  
    LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent);  
    ...  
}
```

下一步是在發送任務的組件中接收發送出來的broadcast數據。

## 接收來自IntentService的狀態廣播

為了接受廣播的數據對象，需要使用BroadcastReceiver的子類並實現BroadcastReceiver.onReceive() )的方法，這裏可以接收LocalBroadcastManager發出的廣播數據。

```
// 廣播接收器，用於接受IntentService更新的狀態  
private class ResponseReceiver extends BroadcastReceiver  
{  
    // 防止實例化  
    private DownloadStateReceiver() {
```

```

}

// 它會被註冊來接收Intent，當廣播接收器獲得這個Intent時調用。
@Override
public void onReceive(Context context, Intent intent) {
    ...
    /*
     * 在這裏處理Intent.
     */
    ...
}
}

```

一旦定義了BroadcastReceiver，也應該定義actions，categories與data用來做廣播過濾。為了實現這些，需要使用IntentFilter。如下所示：

```

// 用於現實照片的類
public class DisplayActivity extends FragmentActivity {
    ...
    public void onCreate(Bundle stateBundle) {
        ...
        super.onCreate(stateBundle);
        ...
        // 過濾器的動作條件是自定義的BROADCAST_ACTION
        IntentFilter mStatusIntentFilter = new IntentFilter(
            Constants.BROADCAST_ACTION);

        // 為http協議添加一個數據過濾器
        mStatusIntentFilter.addDataScheme("http");
    }
}

```

為了給系統註冊這個BroadcastReceiver和IntentFilter，需要通過LocalBroadcastManager執行registerReceiver()的方法。如下所示：

```

// 實例化一個新的 DownloadStateReceiver
DownloadStateReceiver mDownloadStateReceiver =
    new DownloadStateReceiver();

// 註冊DownloadStateReceiver和Intent過濾器
LocalBroadcastManager.getInstance(this).registerReceiver(
    mDownloadStateReceiver,
    mStatusIntentFilter);
...

```

一個BroadcastReceiver可以處理多種類型的廣播數據。每個廣播數據都有自己的ACTION。這個功能使得不用定義多個不同的BroadcastReceiver來分別處理不同的ACTION數據。為BroadcastReceiver定義另外一個IntentFilter，只需要創建一個新的IntentFilter並重複執行registerReceiver()即可。例如：

```

/*
 * 實例化一個新的動作過濾器.
 * 不需要數據類型過濾器.
 */
statusIntentFilter = new IntentFilter(Constants.ACTION_ZOOM_IMAGE);
...
// Registers the receiver with the new filter
// 用新的意圖過濾器註冊廣播接收器
LocalBroadcastManager.getInstance(getActivity()).registerReceiver(
    mDownloadStateReceiver,
    mIntentFilter);

```

發送一個廣播Intent並不會啓動或重啓一個Activity。即使是你的app在後臺運行，Activity的BroadcastReceiver也可以

接收、處理Intent對象。但是這不會迫使你的app進入前臺。當你的app不可見時，如果想通知用戶一個發生在後臺的事件，建議使用Notification。永遠不要為了響應一個廣播Intent而去啓動Activity。

---

筆者評論:使用LocalBroadcastManager結合IntentService其實是一種很典型高效的做法，同時也更符合OO的思想，通過廣播註冊與反註冊的方式，對兩個組件進行解耦。如果使用Handler傳遞到後臺線程作為回調，容易帶來的內存泄漏。原因是：匿名內部類對外面的Activity持有引用，如果在Activity被銷燬的時候，沒有對Handler進行顯式的解綁，會導致Activity無法正常銷燬，這樣自然就有了內存泄漏。當然，如果用文章中的方案，通常也要記得在Activity的onPause的時候進行unRegisterReceiver，除非你有充足的理由為解釋這裏為何要繼續保留。

# 使用CursorLoader在後臺加載數據

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/load-data-background/index.html>

從ContentProvider查詢你需要顯示的數據是比較耗時的。如果你在Activity中直接執行查詢的操作，那麼有可能導致Activity出現ANR的錯誤。即使沒有發生ANR，用戶也會看到一個令人煩惱的UI延遲。為了避免那些問題，你應該在另外一個線程中執行查詢的操作，等待查詢操作完成，然後再顯示查詢結果。

通過CursorLoader對象，你可以以一種簡單的方式實現異步查詢，查詢結束時它會和Activity進行重新連接。CursorLoader不僅僅能夠實現在後臺查詢數據，還能夠在查詢數據發生變化時自動執行重新查詢的操作。

這節課會介紹如何使用CursorLoader來執行一個後臺查詢數據的操作。在這節課中的演示代碼使用的是v4 Support Library中的類。

## 下載演示代碼

---

[ThreadSample](#)

## Lessons

---

- [使用CursorLoader執行查詢任務：Running a Query with a CursorLoader](#)

學習如何使用CursorLoader在後臺執行查詢操作。

- [處理查詢的結果：Handling the Results](#)

學習如何處理從CursorLoader查詢到的數據，以及在loader框架重置CursorLoader時如何解除當前Cursor的引用。

# 使用CursorLoader執行查詢任務

編寫:kesenhoo - 原文:<http://developer.android.com/training/load-data-background/setup-loader.html>

CursorLoader通過ContentProvider在後臺執行一個異步的查詢操作，並且返回數據給調用它的Activity或者FragmentActivity。這使得Activity或者FragmentActivity能夠在查詢任務正在執行可以繼續與用戶進行其他的交互。

## 定義使用CursorLoader的Activity

為了在Activity或者FragmentActivity中使用CursorLoader，它們需要實現LoaderCallbacks接口。CursorLoader會調用LoaderCallbacks定義的這些回調方法與這些類進行交互；這節課與下節課會詳細介紹每一個回調方法。

例如，下面演示了FragmentActivity如何使用CursorLoader。

```
public class PhotoThumbnailFragment extends FragmentActivity implements  
    LoaderManager.LoaderCallbacks<Cursor> {  
    ...  
}
```

## 初始化查詢

為了初始化查詢，需要執行LoaderManager.initLoader()。這個方法可以初始化後臺查詢框架。你可以在用戶輸入查詢條件之後觸發初始化的操作，如果你不需要用戶輸入數據作為查詢條件，你可以觸發這個方法在onCreate()或者onCreateView()。例如：

```
// 標識一個特定的Loader加載器來使用這個組件  
private static final int URL_LOADER = 0;  
...  
/**  
 * 當系統已經準備好顯示Fragment時，  
 * 這裏顯示Fragment的佈局  
 */  
public View onCreateView(  
    LayoutInflater inflater,  
    ViewGroup viewGroup,  
    Bundle bundle) {  
    ...  
    /*  
     * 初始化一個CursorLoader。URL_LOADER值最終會被傳遞到  
     * onCreateLoader().  
     */  
    getLoaderManager().initLoader(URL_LOADER, null, this);  
    ...  
}
```

Note: getLoaderManager() 僅僅是在Fragment類中可以直接訪問。為了在FragmentActivity中獲取到LoaderManager，需要執行getSupportLoaderManager()。

## 開始查詢

一旦後臺任務被初始化好，它會執行你實現的回調方法[onCreateLoader\(\)](#)。為了啓動查詢任務，會在這個方法裏面返回 CursorLoader。你可以初始化一個空的CursorLoader然後使用它的方法來定義你的查詢條件，或者你可以在初始化 CursorLoader對象的時候就同時定義好查詢條件：

```
/**  
 * 系統在完成Loader的初始化並且準備好查詢的時候會回調這個方法。  
 * 這個通常在initLoader()方法被調用發生。包含loaderID值的參數  
 * 通過initLoader()方法的傳遞得到。  
 */  
@Override  
public Loader<Cursor> onCreateLoader(int loaderID, Bundle bundle)  
{  
    /*  
     * 通過加載器ID執行創建加載器動作  
     */  
    switch (loaderID) {  
        case URL_LOADER:  
            // Returns a new CursorLoader  
            return new CursorLoader(  
                getActivity(), // 父Activity上下文  
                mDataUrl, // 要查詢的表  
                mProjection, // 要返回的Projection  
                null, // 沒有條件從句  
                null, // 沒有條件參數  
                null // 默認排序  
            );  
        default:  
            // 一個非法的id傳入  
            return null;  
    }  
}
```

一旦後臺查詢任務獲取到了這個Loader對象，就開始在後臺執行查詢的任務。當查詢完成之後，會執行[onLoadFinished\(\)](#)這個回調函數，關於這些內容會在下一節講到。

# 處理查詢的結果

編寫:kesenhoo - 原文:<http://developer.android.com/training/load-data-background/handle-results.html>

正如前面一節課講到的，你應該在 `onCreateLoader()` 的回調裏面使用 `CursorLoader` 執行加載數據的操作。`Loader` 查詢完後會調用 `Activity` 或者 `FragmentActivity` 的 `LoaderCallbacks.onLoadFinished()` 將結果回調回來。這個回調方法的參數之一是 `Cursor`，它包含了查詢的數據。你可以使用 `Cursor` 對象來更新需要顯示的數據或者進行下一步的處理。

除了 `onCreateLoader()` 與 `onLoadFinished()`，你也需要實現 `onLoaderReset()`。這個方法在 `CursorLoader` 檢測到 `Cursor` 上的數據發生變化的時候會被觸發。當數據發生變化時，系統會觸發重新查詢的操作。

## Handle Query Results

為了顯示 `CursorLoader` 返回的 `Cursor` 數據，需要使用實現 `AdapterView` 的類，併為這個類綁定一個實現了 `CursorAdapter` 的 `Adapter`。系統會自動把 `Cursor` 中的數據顯示到 `View` 上。

你可以在顯示數據之前建立 `View` 與 `Adapter` 的關聯。然後在 `onLoadFinished()` 的時候把 `Cursor` 與 `Adapter` 進行綁定。一旦你把 `Cursor` 與 `Adapter` 進行綁定之後，系統會自動更新 `View`。當 `Cursor` 上的內容發生改變的時候，也會觸發這些操作。

例如：

```
public String[] mFromColumns = {
    DataProviderContract.IMAGE_PICTURENAME_COLUMN
};
public int[] mToFields = {
    R.id.PictureName
};
// 取得ListView的引用[原詞是句柄handle]
ListView mListview = (ListView) findViewById(R.id.dataList);
/*
 * 為ListView定義一個SimpleCursorAdapter
 */
SimpleCursorAdapter mAdapter =
    new SimpleCursorAdapter(
        this, // 當前上下文
        R.layout.list_item, // 一個只有單行的文本的佈局
        null, // 暫時還沒有Cursor遊標
        mFromColumns, // 要使用遊標的列
        mToFields, // Layout fields to use
        0 // No flags
    );
// 為View設置適配器
mListview.setAdapter(mAdapter);
...
/*
 * 定義CursorLoader完成查詢時候的回調
 */
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    ...
    /*
     * 轉移查詢結果給適配器，並激發適配器更新前端的ListView數據
     */
    mAdapter.changeCursor(cursor);
}
```

## Delete Old Cursor References

---

當Cursor失效的時候，CursorLoader會被重置。這通常發生在Cursor相關的數據改變的時候。在重新執行查詢操作之前，系統會執行你的[onLoaderReset\(\)](#)回調方法。在這個回調方法中，你應該刪除當前Cursor上的所有數據，避免發生內存泄露。一旦onLoaderReset()執行結束，CursorLoader就會重新執行查詢操作。

例如：

```
/*
 * CursorLoader被重置時調用。舉個栗子，當提供者中的數據發生變動Cursor變得陳舊
 * 時會被調用。
 */
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    /*
     * 清除適配器裏對Cursor的引用，可以防止內存泄漏。
     */
    mAdapter.changeCursor(null);
}
```

# 管理設備的喚醒狀態

---

編寫:jdneo,litowq - 原文:<http://developer.android.com/training/scheduling/index.html>

當一個Android設備閒置時，首先它的屏幕將會變暗，然後關閉屏幕，最後關閉CPU。這樣可以防止設備的電量被迅速消耗殆盡。但是，有時候也會存在一些特例：

- 例如遊戲或視頻應用需要保持屏幕常亮；
- 其它應用也許不需要屏幕常亮，但或許會需要CPU保持運行，直到某個關鍵操作結束。

這節課描述如何在必要的時候保持設備喚醒，同時又不會過多消耗它的電量。

## 樣例代碼

---

## 課程

---

### 保持設備喚醒

學習如何在必要的時候保持屏幕和CPU喚醒，同時減少對電池壽命的影響。

### 調度重複鬧鐘

對於那些發生在應用生命週期之外的操作，學習如何使用重複鬧鐘對它們進行調度，即使該應用沒有運行或者設備處於睡眠狀態。

# 保持設備喚醒

編寫:jdneo - 原文:<http://developer.android.com/training/scheduling/wakelock.html>

為了避免電量過度消耗，Android設備會在被閒置之後迅速進入睡眠狀態。然而有時候應用會需要喚醒屏幕或者CPU並且保持它們的喚醒狀態，直至一些任務被完成。

想要做到這一點，所採取的方法依賴於應用的具體需求。但是，通常來說，你應該使用最輕量級的方法，減小其對系統資源的影響。在接下來的部分中，我們將會描述在設備默認的睡眠行為與應用的需求不相符合的情況下，你應該如何進行對應的處理。

## 保持屏幕常亮

某些應用需要保持屏幕常亮，比如遊戲與視頻應用。最好的方式是在你的Activity中（且僅在Activity中，而不是Service或其他應用組件裏）使用FLAG\_KEEP\_SCREEN\_ON，例如：

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);  
    }  
}
```

該方法的優點與喚醒鎖（Wake Locks）不同（喚醒鎖的內容在本章節後半部分），它不需要任何特殊的權限，系統會正確地管理應用之間的切換，且不必關心釋放資源的問題。

另外一種方法是在應用的XML佈局文件裏，使用android:keepScreenOn屬性：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:keepScreenOn="true">  
    ...  
</RelativeLayout>
```

使用 android:keepScreenOn="true" 與使用FLAG\_KEEP\_SCREEN\_ON等效。你可以選擇最適合你的應用的方法。在Activity中通過代碼設置常亮標識的優點在於：你可以通過代碼動態清除這個標示，從而使屏幕可以關閉。

Notes：除非你不再希望正在運行的應用長時間點亮屏幕（例如：在一定時間無操作發生後，你想要將屏幕關閉），否則你是不需要清除FLAG\_KEEP\_SCREEN\_ON 標識的。WindowManager會在應用進入後臺或者返回前臺時，正確管理屏幕的點亮或者關閉。但是如果你想要顯式地清除這一標識，從而使得屏幕能夠關閉，可以使

```
getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

## 保持CPU運行

如果你需要在設備睡眠之前，保持CPU運行來完成一些工作，你可以使用[PowerManager](#)系統服務中的喚醒鎖功能。喚醒鎖允許應用控制設備的電源狀態。

創建和保持喚醒鎖會對設備的電源壽命產生巨大影響。因此你應該僅在你確實需要時使用喚醒鎖，且使用的時間應該越短越好。如果想要在[Activity](#)中使用喚醒鎖就顯得沒有必要了。如上所述，可以在[Activity](#)中使用[FLAG\\_KEEP\\_SCREEN\\_ON](#)讓屏幕保持常亮。

使用喚醒鎖的一種合理情況可能是：一個後臺服務需要在屏幕關閉時利用喚醒鎖保持CPU運行。再次強調，應該儘可能規避使用該方法，因為它會影響到電池壽命。

不必使用喚醒鎖的情況：

1. 如果你的應用正在執行一個HTTP長連接的下載任務，可以考慮使用[DownloadManager](#)。
2. 如果你的應用正在從一個外部服務器同步數據，可以考慮創建一個[SyncAdapter](#)
3. 如果你的應用需要依賴於某些後臺服務，可以考慮使用[RepeatingAlarm](#)或者[Google Cloud Messaging](#)，以此每隔特定的時間，將這些服務激活。

為了使用喚醒鎖，首先需要在應用的Manifest清單文件中增加[WAKE\\_LOCK](#)權限：

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

如果你的應用包含一個BroadcastReceiver並使用Service來完成一些工作，你可以通過[WakefulBroadcastReceiver](#)管理你喚醒鎖。後續章節中將會提到，這是一種推薦的方法。如果你的應用不滿足上述情況，可以使用下面的方法直接設置喚醒鎖：

```
PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);
WakeLock wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
    "MyWakeLockTag");
wakeLock.acquire();
```

可以調用[wakelock.release\(\)](#)來釋放喚醒鎖。當應用使用完畢時，應該釋放該喚醒鎖，以避免電量過度消耗。

## 使用WakefulBroadcastReceiver

你可以將BroadcastReceiver和Service結合使用，以此來管理後臺任務的生命週期。[WakefulBroadcastReceiver](#)是一種特殊的BroadcastReceiver，它關注於創建和管理應用的[PARTIAL\\_WAKE\\_LOCK](#)。[WakefulBroadcastReceiver](#)會將任務交付給Service（一般會是一個[IntentService](#)），同時確保設備在此過程中不會進入睡眠狀態。如果在該過程當中沒有保持住喚醒鎖，那麼還沒等任務完成，設備就有可能進入睡眠狀態了。其結果就是：應用可能會在未來的某一個時間節點才把任務完成，這顯然不是你所期望的。

要使用[WakefulBroadcastReceiver](#)，首先在Manifest文件添加一個標籤：

```
<receiver android:name=".MyWakefulReceiver"></receiver>
```

下面的代碼通過[startWakefulService\(\)](#)啟動 [MyIntentService](#)。該方法和[startService\(\)](#)類似，除了[WakefulBroadcastReceiver](#)會在Service啟動後將喚醒鎖保持住。傳遞給[startWakefulService\(\)](#)的Intent會攜帶有一個Extra數據，用來標識喚醒鎖。

```
public class MyWakefulReceiver extends WakefulBroadcastReceiver {
```

```
@Override  
public void onReceive(Context context, Intent intent) {  
  
    // Start the service, keeping the device awake while the service is  
    // launching. This is the Intent to deliver to the service.  
    Intent service = new Intent(context, MyIntentService.class);  
    startWakefulService(context, service);  
}  
}
```

當Service結束之後，它會調用[MyWakefulReceiver.completeWakefulIntent\(\)](#)來釋放喚醒鎖。[completeWakefulIntent\(\)](#)方法中的Intent參數是和[WakefulBroadcastReceiver](#)傳遞進來的Intent參數一致的：

```
public class MyIntentService extends IntentService {  
    public static final int NOTIFICATION_ID = 1;  
    private NotificationManager mNotificationManager;  
    NotificationCompat.Builder builder;  
    public MyIntentService() {  
        super("MyIntentService");  
    }  
    @Override  
    protected void onHandleIntent(Intent intent) {  
        Bundle extras = intent.getExtras();  
        // Do the work that requires your app to keep the CPU running.  
        // ...  
        // Release the wake lock provided by the WakefulBroadcastReceiver.  
        MyWakefulReceiver.completeWakefulIntent(intent);  
    }  
}
```

# 調度重複的鬧鐘

編寫:jdneo - 原文:<http://developer.android.com/training/scheduling/alarms.html>

## 教學視頻（需翻牆）：

- [The App Clinic:Cricket](#)
- [DevBytes:Efficient Data Transfers](#)

鬧鐘（基於`AlarmManager`類）給予你一種在應用使用期之外執行與時間相關的操作的方法。你可以使用鬧鐘初始化一個長時間的操作，例如每天開啓一次後臺服務，下載當日的天氣預報。

鬧鐘具有如下特性：

- 允許你通過預設時間或者設定某個時間間隔，來觸發Intent；
- 你可以將它與`BroadcastReceiver`相結合，來啓動服務並執行其他操作；
- 可在應用範圍之外執行，所以你可以在你的應用沒有運行或設備處於睡眠狀態的情況下，使用它來觸發事件或行為；
- 幫助你的應用最小化資源需求，你可以使用鬧鐘調度你的任務，來替代計時器或者長時間連續運行的後臺服務。

Note：對於那些需要確保在應用使用期之內發生的定時操作，可以使用鬧鐘替代使用`Handler`結合`Timer`與`Thread`的方法。因為它可以讓Android系統更好地統籌系統資源。

## 權衡利弊

重複鬧鐘的機制比較簡單，沒有太多的靈活性。它對於你的應用來說或許不是一種最好的選擇，特別是當你想要觸髮網絡操作的時候。設計不佳的鬧鐘會導致電量快速耗盡，而且會對服務端產生巨大的負荷。

當我們從服務端同步數據時，往往會在應用不被使用的時候時觸發某些操作。此時你可能希望使用重複鬧鐘。但是如果存儲數據的服務端是由你控制的，使用`Google Cloud Messaging`（GCM）結合`sync adapter`是一種更好解決方案。`SyncAdapter`提供的任務調度選項和`AlarmManager`基本相同，但是它能提供更多的靈活性。比如：同步的觸發可能基於一條“新數據”提示消息，而消息的產生可以基於服務器或設備（詳見[執行Sync Adapter](#)），用戶的操作（或者沒有操作），每天的某一時刻等等。你可以觀看本節課最開始提供的兩端視頻瞭解一下有關如何以及何時使用GCM和`SyncAdapter`的知識。

## 最佳實踐方法

在設計重複鬧鐘過程中，你所做出的每一個決定都有可能影響到你的應用將會如何使用系統資源。例如，我們假想一個會從服務器同步數據的應用。同步操作基於的是時鐘時間，具體來說，每一個應用的實例會在下午11十一點整進行同步，巨大的服務器負荷會導致服務器響應時間變長，甚至拒絕服務。因此在我們使用鬧鐘時，請牢記下面的最佳實踐建議：

- 對任何由重複鬧鐘觸發的網絡請求添加一定的隨機性（抖動）：
  - 在鬧鐘觸發時做一些本地任務。“本地任務”指的是任何不需要訪問服務器或者從服務器獲取數據的任務；
  - 同時對於那些包含有網絡請求的鬧鐘，在調度時機上增加一些隨機性。
- 儘量讓你的鬧鐘頻率最小；
- 如果不是必要的情況，不要喚醒設備（這一點與鬧鐘的類型有關，本節課後續部分會提到）；
- 觸發鬧鐘的時間不必過度精確； 儘量使用`setInexactRepeating()`方法替代`setRepeating()`方法。當你使用`setInexactRepeating()`方法時，Android系統會集中多個應用的重複鬧鐘同步請求，並一起觸發它們。這可以減

少系統將設備喚醒的總次數，以此減少電量消耗。從Android 4.4（API Level 19）開始，所有的重複鬧鐘都將是非精確型的。注意雖然`setInexactRepeating()`是`setRepeating()`的改進版本，它依然可能會導致每一個應用的實例在某一時間段內同時訪問服務器，造成服務器負荷過重。因此如之前所述，對於網絡請求，我們需要為鬧鐘的觸發時機增加隨機性。

- 儘量避免讓鬧鐘基於時鐘時間。想要在某一個精確時刻觸發重複鬧鐘是比較困難的。我們應該儘可能使用`ELAPSED_REALTIME`。不同的鬧鐘類型會在本節課後半部分展開。

## 設置重複鬧鐘

如上所述，對於定期執行的任務或者數據查詢而言，使用重複鬧鐘是一個不錯的選擇。它具有下列屬性：

- 鬧鐘類型（後續章節中會展開討論）；
- 觸發時間。如果觸發時間是過去的某個時間點，鬧鐘會立即被觸發；
- 鬧鐘間隔時間。例如，一天一次，每小時一次，每五秒一次，等等；
- 在鬧鐘被觸發時才被髮出的Pending Intent。如果你為同一個Pending Intent設置了另一個鬧鐘，那麼它會將第一個鬧鐘覆蓋。

### 選擇鬧鐘類型

使用重複鬧鐘要考慮的第一件事情是鬧鐘的類型。

鬧鐘類型有兩大類：`ELAPSED_REALTIME` 和 `REAL_TIME_CLOCK`（RTC）。`ELAPSED_REALTIME` 從系統啓動之後開始計算，`REAL_TIME_CLOCK` 使用的是世界統一時間（UTC）。也就是說由於 `ELAPSED_REALTIME` 不受地區和時區的影響，所以它適合於基於時間差的鬧鐘（例如一個每過30秒觸發一次的鬧鐘）。`REAL_TIME_CLOCK` 適合於那些依賴於地區位置的鬧鐘。

兩種類型的鬧鐘都還有一個喚醒（WAKEUP）版本，也就是可以在設備屏幕關閉的時候喚醒CPU。這可以確保鬧鐘會在既定的時間被激活，這對於那些實時性要求比較高的應用（比如含有一些對執行時間有要求的操作）來說非常有效。如果你沒有使用喚醒版本的鬧鐘，那麼所有的重複鬧鐘會在下一次設備被喚醒時被激活。

如果你只是簡單的希望鬧鐘在一個特定的時間間隔被激活（例如每半小時一次），那麼你可以使用任意一種 `ELAPSED_REALTIME` 類型的鬧鐘，通常這會是一個更好的選擇。

如果你的鬧鐘是在每一天的特定時間被激活，那麼你可以選擇 `REAL_TIME_CLOCK` 類型的鬧鐘。不過需要注意的是，這個方法會有一些缺陷——如果地區發生了變化，應用可能無法做出正確的改變；另外，如果用戶改變了設備的時間設置，這可能會造成應用產生預期之外的行為。使用 `REAL_TIME_CLOCK` 類型的鬧鐘還會有精度的問題，因此我們建議你儘可能使用 `ELAPSED_REALTIME` 類型。

下面列出鬧鐘的具體類型：

- `ELAPSED_REALTIME`：從設備啓動之後開始算起，度過了某一段特定時間後，激活Pending Intent，但不會喚醒設備。其中設備睡眠的時間也會包含在內。
- `ELAPSED_REALTIME_WAKEUP`：從設備啓動之後開始算起，度過了某一段特定時間後喚醒設備。
- `RTC`：在某一個特定時刻激活Pending Intent，但不會喚醒設備。
- `RTC_WAKEUP`：在某一個特定時刻喚醒設備並激活Pending Intent。

### ELAPSED\_REALTIME\_WAKEUP案例

下面是使用`ELAPSED_REALTIME_WAKEUP`的例子。

每隔在30分鐘後喚醒設備以激活鬧鐘：

```
// Hopefully your alarm will have a lower frequency than this!
alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    AlarmManager.INTERVAL_HALF_HOUR,
    AlarmManager.INTERVAL_HALF_HOUR, alarmIntent);
```

在一分鐘後喚醒設備並激活一個一次性（無重複）鬧鐘：

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);

alarmMgr.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    SystemClock.elapsedRealtime() +
    60 * 1000, alarmIntent);
```

## RTC案例

下面是使用[RTC\\_WAKEUP](#)的例子。

在大約下午2點喚醒設備並激活鬧鐘，並不斷重複：

```
// Set the alarm to start at approximately 2:00 p.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 14);

// With setInexactRepeating(), you have to use one of the AlarmManager interval
// constants--in this case, AlarmManager.INTERVAL_DAY.
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
    AlarmManager.INTERVAL_DAY, alarmIntent);
```

讓設備精確地在上午8點半被喚醒並激活鬧鐘，自此之後每20分鐘喚醒一次：

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);

// Set the alarm to start at 8:30 a.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 8);
calendar.set(Calendar.MINUTE, 30);

// setRepeating() lets you specify a precise custom interval--in this case,
// 20 minutes.
alarmMgr.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
    1000 * 60 * 20, alarmIntent);
```

## 決定鬧鐘的精確度

如上所述，創建鬧鐘的第一步是要選擇鬧鐘的類型，然後你需要決定鬧鐘的精確度。對於大多數應用而言，[setInexactRepeating\(\)](#)會是一個正確的選擇。當你使用該方法時，Android系統會集中多個應用的重複鬧鐘同步請

求，並一起觸發它們。這樣可以減少電量的損耗。

對於另一些實時性要求較高的應用——例如，鬧鐘需要精確地在上午8點半被激活，並且自此之後每隔1小時激活一次——那麼可以使用[setRepeating\(\)](#)。不過你應該儘量避免使用精確的鬧鐘。

使用[setRepeating\(\)](#)時，你可以制定一個自定義的時間間隔，但在使用[setInexactRepeating\(\)](#)時不支持這麼做。此時你只能選擇一些時間間隔常量，例如：[INTERVAL\\_FIFTEEN\\_MINUTES](#)，[INTERVAL\\_DAY](#)等。完整的常量列表，可以查看[AlarmManager](#)。

## 取消鬧鐘

你可能希望在應用中添加取消鬧鐘的功能。要取消鬧鐘，可以調用AlarmManager的[cancel\(\)](#)方法，並把你不想激活的PendingIntent傳遞進去，例如：

```
// If the alarm has been set, cancel it.  
if (alarmMgr!= null) {  
    alarmMgr.cancel(alarmIntent);  
}
```

## 在設備啓動後啓用鬧鐘

默認情況下，所有的鬧鐘會在設備關閉時被取消。要防止鬧鐘被取消，你可以讓你的應用在用戶重啓設備後自動重啓一個重複鬧鐘。這樣可以讓[AlarmManager](#)繼續執行它的工作，且不需要用戶手動重啓鬧鐘。

具體步驟如下：

1.在應用的Manifest文件中設置[RECEIVE\\_BOOT\\_COMPLETED](#)權限，這將允許你的應用接收系統啓動完成後發出的[ACTION\\_BOOT\\_COMPLETED](#)廣播（只有在用戶至少將你的應用啓動了一次後，這樣做纔有效）：

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

2.實現BroadcastReceiver用於接收廣播：

```
public class SampleBootReceiver extends BroadcastReceiver {  
  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {  
            // Set the alarm here.  
        }  
    }  
}
```

3.在你的Manifest文件中添加一個接收器，其Intent-Filter接收[ACTION\\_BOOT\\_COMPLETED](#)這一Action：

```
<receiver android:name=".SampleBootReceiver"  
         android:enabled="false">  
    <intent-filter>  
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>  
    </intent-filter>  
</receiver>
```

注意Manifest文件中，對接收器設置了 `android:enabled="false"` 屬性。這意味着除非應用顯式地啓用它，不然該接收器將不被調用。這可以防止接收器被不必要的調用。你可以像下面這樣啓動接收器（比如用戶設置了一個鬧鐘）：

```
ComponentName receiver = new ComponentName(context, SampleBootReceiver.class);
PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP);
```

一旦你像上面那樣啓動了接收器，它將一直保持啓動狀態，即使用戶重啓了設備也不例外。換句話說，通過代碼設置的啓用配置將會覆蓋掉Manifest文件中的現有配置，即使重啓也不例外。接收器將保持啓動狀態，直到你的應用將其禁用。你可以像下面這樣禁用接收器（比如用戶取消了一個鬧鐘）：

```
ComponentName receiver = new ComponentName(context, SampleBootReceiver.class);
PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
    PackageManager.DONT_KILL_APP);
```

# 性能優化

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/best-performance.html>

下面的這些課程會介紹如何提升應用的性能，如何儘量減少電量的消耗。

## 管理應用的內存

如何減少內存的佔用。

## 代碼性能優化建議

如何提高應用的響應性與電池的使用效率。

## 提升Layout的性能

如何提升UI的性能。

## 優化電池壽命

如何優化電量的消耗。

## 多線程操作

如何通過多線程分拆任務來提高程序性能。

## 避免出現程序無響應ANR

如何避免ANR。

## JNI技巧

如何高效的使用JNI。

## SMP Primer for Android

優化多核處理器架構下的Android程序。

# 管理應用的內存

編寫:kesenhoo - 原文:<http://developer.android.com/training/articles/memory.html>

Random Access Memory(RAM)在任何軟件開發環境中都是一個很寶貴的資源。這一點在物理內存通常很有限的移動操作系統上，顯得尤爲突出。儘管Android的Dalvik虛擬機扮演了常規的垃圾回收的角色，但這並不意味着你可以忽視app的內存分配與釋放的時機與地點。

爲了GC能夠從你的app中及時回收內存，你需要避免內存泄露(通常由於在全局成員變量中持有對象引用而導致)並且在適當的時機(下面會講到的lifecycle callbacks)來釋放引用對象。對於大多數apps來說，Dalvik的GC會自動把離開活動線程的對象進行回收。

這篇文章會解釋Android是如何管理app的進程與內存分配，以及在開發Android應用的時候如何主動的減少內存的使用。關於Java的資源管理機制，請參考其它書籍或者線上材料。如果你正在尋找如何分析你的內存使用情況的文章，請參考這裏[Investigating Your RAM Usage](#)。

## 第1部分: Android是如何管理內存的

Android並沒有提供內存的交換區(Swap space)，但是它有使用paging與memory-mapping(mmapping)的機制來管理內存。這意味着任何你修改的內存(無論是通過分配新的對象還是訪問到mmaped pages的內容)都會貯存在RAM中，而且不能被paged out。因此唯一完整釋放內存的方法是釋放那些你可能hold住的對象的引用，這樣使得它能夠被GC回收。只有一種例外是：如果系統想要在其他地方reuse這個對象。

### 1) 共享內存

Android通過下面幾個方式在不同的Process中來共享RAM:

- 每一個app的process都是從同一個被叫做Zygote的進程中fork出來的。Zygote進程在系統啓動並且載入通用的framework的代碼與資源之後開始啓動。爲了啓動一個新的程序進程，系統會fork Zygote進程生成一個新的進程，然後在新的進程中加載並運行app的代碼。這使得大多數的RAM pages被用來分配給framework的代碼與資源，並在應用的所有進程中進行共享。
- 大多數static的數據被mmapped到一個進程中。這不僅僅使得同樣的數據能夠在進程間進行共享，而且使得它能夠在需要的時候被paged out。例如下面幾種static的數據：
  - Dalvik 代碼 (放在一個已經鏈接好的 .odex file 用於直接內存映射查找)
  - App resources (一個資源表結構，可以被內存映射，以及在APK文件中做zip aligning)
  - 傳統項目元素，比如 .so 文件中的本地代碼。
- 在許多地方，Android通過顯式的分配共享內存區域(例如ashmem或者gralloc)來實現一些動態RAM區域的能夠在不同進程間的共享。例如，window surfaces在app與screen compositor之間使用共享的內存，cursor buffers在content provider與client之間使用共享的內存。

關於如何查看app所使用的共享內存，請查看[Investigating Your RAM Usage](#)

### 2) 分配與回收內存

這裏有下面幾點關於Android如何分配與回收內存的事實：

- 每一個進程的Dalvik heap都有一個限制的虛擬內存範圍。這就是邏輯上講的heap size，它可以隨着需要進行增長，但是會有一個系統爲它所定義的上限。
- 邏輯上講的heap size和實際物理上使用的內存數量是不等的，Android會計算一個叫做Proportional Set

- Size(PSS)的值，它記錄了那些和其他進程進行共享的內存大小。（假設共享內存大小是10M，一共有20個Process在共享使用，根據權重，可能認為其中有0.3M才能真正算是你的進程所使用的）
- Dalvik heap與邏輯上的heap size不吻合，這意味着Android並不會去做heap中的碎片整理用來關閉空閒區域。Android僅僅會在heap的尾端出現不使用的空間時纔會做收縮邏輯heap size大小的動作。但是這並不是意味着被heap所使用的物理內存大小不能被收縮。在垃圾回收之後，Dalvik會遍歷heap並找出不使用的pages，然後使用madvise(系統調用)把那些pages返回給kernel。因此，成對的allocations與deallocations大塊的數據可以使得物理內存能夠被正常的回收。然而，回收碎片化的內存則會使得效率低下很多，因為那些碎片化的分配頁面也許會被其他地方所共享到。

### 3) 限制應用的內存

為了維持多任務的功能環境，Android為每一個app都設置了一個硬性的heap size限制。準確的heap size限制隨着不同設備的不同RAM大小而各有差異。如果你的app已經到了heap的限制大小並且再嘗試分配內存的話，會引起 `OutOfMemoryError` 的錯誤。

在一些情況下，你也許想要查詢當前設備的heap size限制大小是多少，然後決定cache的大小。可以通過 `getMemoryClass()` 來查詢。這個方法會返回一個整數，表明你的app heap size限制是多少Mb(megabates)。

### 4) 切換應用

Android並不會在用戶切換不同應用時候做交換內存的操作。Android會把那些不包含foreground組件的進程放到LRU cache中。例如，當用戶剛開始啓動了一個應用，這個時候為它創建了一個進程，但是當用戶離開這個應用，這個進程並沒有離開。系統會把這個進程放到cache中，如果用戶後來回到這個應用，這個進程能夠被resued，從而實現app的快速切換。

如果你的應用有一個當前並不需要使用到的被緩存的進程，它被保留在內存中，這會對系統的整個性能有影響。因此當系統開始進入低內存狀態時，它會由系統根據LRU的規則與其他因素選擇殺掉某些進程，為了保持你的進程能夠儘可能長久的被cached，請參考下面的章節學習何時釋放你的引用。

對於不在foreground的進程是Android是如何決定kill掉哪一類進程的問題，請參考[Processes and Threads](#).

## 第2部分: 你的應用該如何管理內存

---

你應該在開發過程的每一個階段都考慮到RAM的有限性，甚至包括在開發開始之前的設計階段就應該開始考慮RAM的限制。我們可以有許多種設計與實現方式，他們有着不同的效率，即使這些方式是對同樣一種技術的不斷組合與演變。

為了使得你的應用效率更高，你應該在設計與實現代碼時，遵循下面的技術要點。

### 1) 珍惜Services資源

如果你的app需要在後臺使用service，除非它被觸發執行一個任務，否則其他時候都應該是非運行狀態。同樣需要注意當這個service已經完成任務後停止service失敗引起的泄漏。

當你啓動一個service，系統會傾向為了這個Service而一直保留它的進程。這使得進程的運行代價很高，因為系統沒有辦法把Service所佔用的RAM讓給其他組件或者被paged out。這減少了系統能夠存放到LRU緩存當中的進程數量，它會影響app之間的切換效率。它甚至會導致系統內存使用不穩定，從而無法繼續hold住所有目前正在運行的Service。

限制你的Service的最好辦法是使用[IntentService](#), 它會在處理完扔給它的intent任務之後儘快結束自己。更多信息，請閱讀[Running in a Background Service](#).

當一個Service已經不需要的時候還繼續保留它，這對Android應用的內存管理來說是最糟糕的錯誤之一。因此千萬不要

貪婪的使得一個Service持續保留。不僅僅是因為它會使得你的app因RAM的限制而性能糟糕，而且用戶會發現那些有着常駐後臺行爲的app並且卸載它。

## 2) 你的UI隱藏時釋放內存

當用戶切換到其它app並且你的app UI不再可見時，你應該釋放你的UI上佔用的任何資源。在這個時候釋放UI資源可以顯著的增加系統cached process的能力，它會對用戶體驗有着直接的影響。

為了能夠接收到用戶離開你的UI時的通知，你需要實現Activty類裏面的[onTrimMemory\(\)](#)回調方法。你應該使用這個方法來監聽到TRIM\_MEMORY\_UI\_HIDDEN級別，它意味着你的UI已經隱藏，你應該釋放那些僅僅被你的UI使用的資源。

請注意：你的app僅僅會在所有UI組件的被隱藏的時候接收到onTrimMemory()的回調並帶有參數 TRIM\_MEMORY\_UI\_HIDDEN。這與onStop()的回調是不同的，onStop會在activity的實例隱藏時會執行，例如當用戶從你的app的某個activity跳轉到另外一個activity時onStop會被執行。因此你應該實現onStop回調，並且在此回調裏面釋放activity的資源，例如網絡連接，unregister廣播接收者。除非接收到onTrimMemory(TRIM\_MEMORY\_UI\_HIDDEN)的回調，否者你不應該釋放你的UI資源。這確保了用戶從其他activity切回來時，你的UI資源仍然可用，並且可以迅速恢復activity。

## 3) 當內存緊張時釋放部分內存

在你的app生命週期的任何階段，onTrimMemory回調方法同樣可以告訴你整個設備的內存資源已經開始緊張。你應該根據onTrimMemory方法中的內存級別來進一步決定釋放哪些資源。

- [TRIM\\_MEMORY\\_RUNNING\\_MODERATE](#):你的app正在運行並且不會被列為可殺死的。但是設備此時正運行於低內存狀態下，系統開始觸發殺死LRU Cache中的Process的機制。
- [TRIM\\_MEMORY\\_RUNNING\\_LOW](#):你的app正在運行且沒有被列為可殺死的。但是設備正運行於更低內存的狀態下，你應該釋放不用的資源用來提升系統性能（但是這也會直接影響到你的app的性能）。
- [TRIM\\_MEMORY\\_RUNNING\\_CRITICAL](#):你的app仍在運行，但是系統已經把LRU Cache中的大多數進程都已經殺死，因此你應該立即釋放所有非必須的資源。如果系統不能回收到足夠的RAM數量，系統將會清除所有的LRU緩存中的進程，並且開始殺死那些之前被認為不應該殺死的進程，例如那個包含了一個運行態Service的進程。

同樣，當你的app進程正在被cached時，你可能會接收到從onTrimMemory()中返回的下面的值之一：

- [TRIM\\_MEMORY\\_BACKGROUND](#): 系統正運行於低內存狀態並且你的進程正處於LRU緩存名單中最不容易殺掉的位置。儘管你的app進程並不是處於被殺掉的高危險狀態，系統可能已經開始殺掉LRU緩存中的其他進程了。你應該釋放那些容易恢復的資源，以便於你的進程可以保留下來，這樣當用戶回退到你的app的時候才能夠迅速恢復。
- [TRIM\\_MEMORY\\_MODERATE](#): 系統正運行於低內存狀態並且你的進程已經已經接近LRU名單的中部位置。如果系統開始變得更加內存緊張，你的進程是有可能被殺死的。
- [TRIM\\_MEMORY\\_COMPLETE](#): 系統正運行與低內存的狀態並且你的進程正處於LRU名單中最容易被殺掉的位置。你應該釋放任何不影響你的app恢復狀態的資源。

因為onTrimMemory()的回調是在API 14才被加進來的，對於老的版本，你可以使用[onLowMemory\(\)](#)回調來進行兼容。onLowMemory相當與TRIM\_MEMORY\_COMPLETE。

Note: 當系統開始清除LRU緩存中的進程時，儘管它首先按照LRU的順序來操作，但是它同樣會考慮進程的內存使用量。因此消耗越少的進程則越容易被保留下來。

## 4) 檢查你應該使用多少的內存

正如前面提到的，每一個Android設備都會有不同的RAM總大小與可用空間，因此不同設備為app提供了不同大小的heap限制。你可以通過調用[getMemoryClass\(\)](#)來獲取你的app的可用heap大小。如果你的app嘗試申請更多的內存，會

出現OutOfMemory的錯誤。

在一些特殊的情景下，你可以通過在manifest的application標簽下添加 `largeHeap=true` 的屬性來聲明一個更大的heap空間。如果你這樣做，你可以通過`getLargeMemoryClass()`來獲取到一個更大的heap size。

然而，能夠獲取更大heap的設計本意是爲了一小部分會消耗大量RAM的應用(例如一個大圖片的編輯應用)。不要輕易的因爲你需要使用大量的內存而去請求一個大的heap size。只有當你清楚的知道哪裏會使用大量的內存並且爲什麼這些內存必須被保留時纔去使用large heap. 因此請儘量少使用large heap。使用額外的內存會影響系統整體的用戶體驗，並且會使得GC的每次運行時間更長。在任務切換時，系統的性能會變得大打折扣。

另外，large heap並不一定能夠獲取到更大的heap。在某些有嚴格限制的機器上，large heap的大小和通常的heap size是一樣的。因此即使你申請了large heap，你還是應該通過執行`getMemoryClass()`來檢查實際獲取到的heap大小。

## 5) 避免bitmaps的浪費

當你加載一個bitmap時，僅僅需要保留適配當前屏幕設備分辨率的數據即可，如果原圖高於你的設備分辨率，需要做縮小的動作。請記住，增加bitmap的尺寸會對內存呈現出2次方的增加，因爲X與Y都在增加。

Note:在Android 2.3.x (API level 10)及其以下，bitmap對象的pixel data是存放在native內存中的，它不便於調試。然而，從Android 3.0(API level 11)開始，bitmap pixel data是分配在你的app的Dalvik heap中，這提升了GC的工作效率並且更加容易Debug。因此如果你的app使用bitmap並在舊的機器上引發了一些內存問題，切換到3.0以上的機器上進行Debug。

## 6) 使用優化的數據容器

利用Android Framework裏面優化過的容器類，例如[SparseArray](#), [SparseBooleanArray](#), 與 [LongSparseArray](#)。通常的HashMap的實現方式更加消耗內存，因爲它需要一個額外的實例對象來記錄Mapping操作。另外，SparseArray更加高效在於他們避免了對key與value的autobox自動裝箱，並且避免了裝箱後的解箱。

## 7) 請注意內存開銷

對你所使用的語言與庫的成本與開銷有所瞭解，從開始到結束，在設計你的app時謹記這些信息。通常，表面上看起來無關痛癢(innocuous)的事情也許實際上會導致大量的開銷。例如：

- Enums的內存消耗通常是static constants的2倍。你應該儘量避免在Android上使用enums。
- 在Java中的每一個類(包括匿名內部類)都會使用大概500 bytes。
- 每一個類的實例花銷是12-16 bytes。
- 往HashMap添加一個entry需要額一個額外佔用的32 bytes的entry對象。

## 8) 請注意代碼 “抽象”

通常，開發者使用抽象作爲“好的編程實踐”，因爲抽象能夠提升代碼的靈活性與可維護性。然而，抽象會導致一個顯著的開銷：通常他們需要同等量的代碼用於可執行。那些代碼會被map到內存中。因此如果你的抽象沒有顯著的提升效率，應該儘量避免他們。

## 9) 為序列化的數據使用nano protobufs

[Protocol buffers](#)是由Google爲序列化結構數據而設計的，一種語言無關，平臺無關，具有良好擴展性的協議。類似XML，卻比XML更加輕量，快速，簡單。如果你需要爲你的數據實現協議化，你應該在客戶端的代碼中總是使用nano protobufs。通常的協議化操作會生成大量繁瑣的代碼，這容易給你的app帶來許多問題：增加RAM的使用量，顯著增加APK的大小，更慢的執行速度，更容易達到DEX的字符限制。

關於更多細節，請參考[protobuf readme](#)的"Nano version"章節。

## 10) 避免使用依賴注入框架

使用類似[Guice](#)或者[RoboGuice](#)等framework injection包是很有效的，因為他們能夠簡化你的代碼。

RoboGuice 2 通過依賴注入改變代碼風格，讓Android開發時的體驗更好。你在調用 `getIntent().getExtras()` 時經常忘記檢查 `null` 嗎？RoboGuice 2 可以幫你做。你認為將 `findViewById()` 的返回值強制轉換成 `TextView` 是本不必要的工作嗎？RoboGuice 2 會幫你。RoboGuice 把這些需要猜測性的工作移到Android 開發以外去了。注入你的 View, Resource, System Service，或者其他對象，RoboGuice 2 會負責這些細節。

然而，那些框架會通過掃描你的代碼執行許多初始化的操作，這會導致你的代碼需要大量的RAM來map代碼。但是 mapped pages 會長時間的被保留在RAM中。

## 11) 謹慎使用external libraries

很多External library的代碼都不是為移動網絡環境而編寫的，在移動客戶端則顯示的效率不高。至少，當你決定使用一個external library的時候，你應該針對移動網絡做繁瑣的porting與maintenance的工作。

即使是針對Android而設計的library，也可能是很危險的，因為每一個library所做的事情都是不一樣的。例如，其中一個lib使用的是nano protobufs，而另外一個使用的是micro protobufs。那麼這樣，在你的app裏面就有2種protobuf的實現方式。這樣的衝突同樣可能發生在輸出日誌，加載圖片，緩存等等模塊裏面。

同樣不要陷入為了1個或者2個功能而導入整個library的陷阱。如果沒有一個合適的庫與你的需求相吻合，你應該考慮自己去實現，而不是導入一個大而全的解決方案。

## 12) 優化整體性能

官方有列出許多優化整個app性能的文章：[Best Practices for Performance](#). 這篇文章就是其中之一。有些文章是講解如何優化app的CPU使用效率，有些是如何優化app的內存使用效率。

你還應該閱讀[optimizing your UI](#)來為layout進行優化。同樣還應該關注lint工具所提出的建議，進行優化。

## 13) 使用ProGuard來剔除不需要的代碼

ProGuard能夠通過移除不需要的代碼，重命名類，域與方法等方對代碼進行壓縮，優化與混淆。使用ProGuard可以是的你的代碼更加緊湊，這樣能夠使用更少mapped代碼所需要的RAM。

## 14) 對最終的APK使用zipalign

在編寫完所有代碼，並通過編譯系統生成APK之後，你需要使用[zipalign](#)對APK進行重新校準。如果你不做這個步驟，會導致你的APK需要更多的RAM，因為一些類似圖片資源的東西不能被mapped。

注意：Google Play不接受沒有經過zipalign的APK。

## 15) 分析你的RAM使用情況

一旦你獲取到一個相對穩定的版本後，需要分析你的app整個生命週期內使用的內存情況，並進行優化，更多細節請參考[Investigating Your RAM Usage](#).

## 16) 使用多進程

如果合適的話，有一個更高級的技術可以幫助你的app管理內存使用：通過把你的app組件切分成多個組件，運行在不同的進程中。這個技術必須謹慎使用，大多數app都不應該運行在多個進程中。因為如果使用不當，它會顯著增加內存的使用，而不是減少。當你的app需要在後臺運行與前臺一樣的大量的任務的時候，可以考慮使用這個技術。

一個典型的例子是創建一個可以長時間後臺播放的Music Player。如果整個app運行在一個進程中，當後臺播放的時候，前臺的那些UI資源也沒有辦法得到釋放。類似這樣的app可以切分成2個進程：一個用來操作UI，另外一個用來後臺的Service.

你可以通過在manifest文件中聲明'android:process'屬性來實現某個組件運行在另外一個進程的操作。

```
<service android:name=".PlaybackService"
    android:process=":background" />
```

更多關於使用這個技術的細節，請參考原文，鏈接如下。

<http://developer.android.com/training/articles/memory.html>

# 代碼性能優化建議

編寫:kesenhoo - 原文:<http://developer.android.com/training/articles/perf-tips.html>

這篇文章主要介紹一些小細節的優化技巧，當這些小技巧綜合使用起來的時候，對於整個App的性能提升還是有作用的，只是不能較大幅度的提升性能而已。選擇合適的算法與數據結構才應該是你首要考慮的因素，在這篇文章中不會涉及這方面。你應該使用這篇文章中的小技巧作為平時寫代碼的習慣，這樣能夠提升代碼的效率。

通常來說，高效的代碼需要滿足下面兩個規則：

- 不要做冗餘的工作
- 如果能避免，儘量不要分配內存

在優化App時最難解決的問題之一就是讓App能在各種類型的設備上運行。不同版本的虛擬機在不同的處理器上會有不同的運行速度。你甚至不能簡單的認為“設備X的速度是設備Y的F倍”，然後還用這種倍數關係去推測其他設備。特別的是，在模擬器上的運行速度和在實際設備上的速度沒有半點關係。同樣，設備有沒有JIT（即時編譯，譯者注）也對運行速度有重大影響：在有JIT情況下的最優化代碼不一定在沒有JIT的情況下也最優化。

為了確保App在各設備上都能良好運行，就要確保你的代碼在不同檔次的設備上都儘可能的優化。

## 避免創建不必要的對象

創建對象從來不是無代價的。在線程分配池裏的逐代垃圾回收器可以使臨時對象的分配變得廉價一些，但是分配內存總是比不分配更昂貴。

隨着你在App中分配更多的對象，你可能需要強制GC（垃圾回收，譯者注），為用戶體驗做一個小小的減壓。Android 2.3 中引入的併發GC會幫助你做這件事情，但畢竟不必要的工作應該儘量避免

因此請儘量避免創建不必要的對象，有下面一些例子來說明這個問題：

- 如果你需要返回一個String對象，並且你知道它最終會需要連接到一個StringBuffer，請修改你的函數簽名和實現方式，避免直接進行連接操作，應該採用創建一個臨時對象來做這個操作。
- 當從輸入的數據集中抽取出String的時候，嘗試返回原數據的substring對象，而不是創建一個重複的對象。你將會new一個 String 對象，但是它應該和原數據共享內部的 char[] (代價是如果你只是用原數據中的一小部分，你只需要保存這一小部分的對象在內存中)

一個稍微激進點的做法是把所有多維的數據分解成一維的數組：

- 一組int數據要比一組Integer對象要好很多。可以得知，兩組一維數組要比一個二維數組更加的有效率。同樣的，這個道理可以推廣至其他原始數據類型。
- 如果你需要實現一個數組用來存放(Foo,Bar)的對象，記住使用Foo[]與Bar[]要比(Foo,Bar)好很多。(例外的是，為了某些好的API的設計，可以適當做一些妥協。但是在自己的代碼內部，你應該多多使用分解後的容易)。

通常來說，需要避免創建更多的臨時對象。更少的對象意味著更少的GC動作，GC會對用戶體驗有比較直接的影響。

## 選擇Static而不是Virtual

如果你不需要訪問一個對象的值域，請保證這個方法是static類型的，這樣方法調用將快15%-20%。這是一個好的習慣，因為你可以從方法聲明中得知調用無法改變這個對象的狀態。

# 常量聲明為Static Final

考慮下面這種聲明的方式

```
static int intValue = 42;
static String strValue = "Hello, world!";
```

編譯器會使用一個初始化類的函數，然後當類第一次被使用的時候執行。這個函數將42存入 `intValue`，還從class文件的常量表中提取了 `strValue` 的引用。當之後使用 `intValue` 或 `strValue` 的時候，他們會直接被查詢到。

我們可以用 `final` 關鍵字來優化：

```
static final int intValue = 42;
static final String strValue = "Hello, world!";
```

這時再也不需要上面的方法了，因為`final`聲明的常量進入了靜態dex文件的域初始化部分。調用 `intValue` 的代碼會直接使用42，調用 `strValue` 的代碼也會使用一個相對廉價的“字符串常量”指令，而不是查表。

注意：這個優化方法只對原始類型和String類型有效，而不是任意引用類型。不過，在必要時使用 `static final` 是個很好的習慣

## 避免內部的Getters/Setters

像C++等native language,通常使用getters(`i = getCount()`)而不是直接訪問變量(`i = mCount`)。這是編寫C++的一種優秀習慣，而且通常也被其他面向對象的語言所採用，例如C#與Java，因為編譯器通常會做inline訪問，而且你需要限制或者調試變量，你可以在任何時候在getter/setter裏面添加代碼。

然而，在Android上，這是一個糟糕的寫法。虛函數的調用比起直接訪問變量要耗費更多。在面向對象編程中，將getter和setting暴露給公用接口是合理的，但在類內部應該僅僅使用域直接訪問。

在沒有JIT(Just In Time Compiler)時，直接訪問變量的速度是調用getter的3倍。有JIT時，直接訪問變量的速度是通過getter訪問的7倍。

請注意，如果你使用[ProGuard](#), 你可以獲得同樣的效果，因為ProGuard可以為你inline accessors.

## 使用增強的For循環

增強的For循環（也被稱為 for-each 循環）可以被用在實現了 `Iterable` 接口的 `collections` 以及數組上。使用collection的時候，`Iterator`（迭代器，譯者注）會被分配，用於for-each調用 `hasNext()` 和 `next()` 方法。使用ArrayList時，手寫的計數式for循環會快3倍（不管有沒有JIT），但是對於其他collection，增強的for-each循環寫法會和迭代器寫法的效率一樣。

請比較下面三種循環的方法：

```
static class Foo {
    int mSplat;
}

Foo[] mArray = ...
```

```

public void zero() {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one() {
    int sum = 0;
    Foo[] localArray = mArray;
    int len = localArray.length;

    for (int i = 0; i < len; ++i) {
        sum += localArray[i].mSplat;
    }
}

public void two() {
    int sum = 0;
    for (Foo a : mArray) {
        sum += a.mSplat;
    }
}

```

- `zero()`是最慢的，因為JIT沒有辦法對它進行優化。
- `one()`稍微快些。
- `two()`在沒有做JIT時是最快的，可是如果經過JIT之後，與方法`one()`是差不多一樣快的。它使用了增強的循環方法`for-each`。

所以請儘量使用`for-each`的方法，但是對於`ArrayList`，請使用方法`one()`。

你還可以參考 Josh Bloch 的《Effective Java》這本書的第46條

## 使用包級訪問而不是內部類的私有訪問

參考下面一段代碼

```

public class Foo {
    private class Inner {
        void stuff() {
            Foo.this.doStuff(Foo.this.mValue);
        }
    }

    private int mValue;

    public void run() {
        Inner in = new Inner();
        mValue = 27;
        in.stuff();
    }

    private void doStuff(int value) {
        System.out.println("Value is " + value);
    }
}

```

這裏重要的是，我們定義了一個私有的內部類（`Foo$Inner`），它直接訪問了外部類中的私有方法以及私有成員對象。這是合法的，這段代碼也會如同預期一樣打印出"Value is 27"。

問題是，VM因為`Foo`和`Foo$Inner`是不同的類，會認為在`Foo$Inner`中直接訪問`Foo`類的私有成員是不合法的。即使

Java語言允許內部類訪問外部類的私有成員。為了去除這種差異，編譯器會產生一些仿造函數：

```
/*package*/ static int Foo.access$100(Foo foo) {
    return foo.mValue;
}
/*package*/ static void Foo.access$200(Foo foo, int value) {
    foo.doStuff(value);
}
```

每當內部類需要訪問外部類中的mValue成員或需要調用doStuff()函數時，它都會調用這些靜態方法。這意味着，上面的代碼可以歸結為，通過accessor函數來訪問成員變量。早些時候我們說過，通過accessor會比直接訪問域要慢。所以，這是一個特定語言用法造成性能降低的例子。

如果你正在性能熱區（hotspot:高頻率、重複執行的代碼段）使用像這樣的代碼，你可以把內部類需要訪問的域和方法聲明為包級訪問，而不是私有訪問權限。不幸的是，這意味着在相同包中的其他類也可以直接訪問這些域，所以在公開的API中你不能這樣做。

## 避免使用float類型

Android系統中float類型的數據存取速度是int類型的一半，儘量優先採用int類型。

就速度而言，現代硬件上，float 和 double 的速度是一樣的。空間而言，double 是兩倍float的大小。在桌面機上，空間不是問題的情況下，你應該使用 double 。

同樣，對於整型，有些處理器實現了硬件幾倍的乘法，但是沒有除法。這時，整型的除法和取餘是在軟件內部實現的，這在你使用哈希表或大量輸血操作時要考慮到。

## 使用庫函數

除了那些常見的讓你多使用自帶庫函數的理由以外，記得系統函數有時可以替代第三方庫，並且還有彙編級別的優化，他們通常比帶有JIT的Java編譯出來的代碼更高效。典型的例子是：Android API 中的 `String.indexOf()`，Dalvik出於內聯性能考慮將其替換。同樣 `System.arraycopy()` 函數也被替換，這樣的性能在Nexus One測試，比手寫的for循環並使用JIT還快9倍。

參見 Josh Bloch 的《Effective Java》這本書的第47條

## 謹慎使用native函數

結合Android NDK使用native代碼開發，並不總是比Java直接開發的效率更好的。Java轉native代碼是有代價的，而且JIT不能在這種情況下做優化。如果你在native代碼中分配資源（比如native堆上的內存，文件描述符等等），這會對收集這些資源造成巨大的困難。你同時也需要為各種架構重新編譯代碼（而不是依賴JIT）。你甚至對已同樣架構的設備都需要編譯多個版本：為G1的ARM架構編譯的版本不能完全使用Nexus One上ARM架構的優勢，反之亦然。

Native 代碼是在你已經有本地代碼，想把它移植到Android平臺時有優勢，而不是為了優化已有的Android Java代碼使用。

如果你要使用JNI,請學習[JNI Tips](#)

參見 Josh Bloch 的《Effective Java》這本書的第54條

## 關於性能的誤區

---

在沒有JIT的設備上，使用一種確切的數據類型確實要比抽象的數據類型速度要更有效率（例如，調用 `HashMap map` 要比調用 `Map map` 效率更高）。有誤傳效率要高一倍，實際上只是6%左右。而且，在JIT之後，他們直接並沒有大多差異。

在沒有JIT的設備上，讀取緩存域比直接讀取實際數據大概快20%。有JIT時，域讀取和本地讀取基本無差。所以優化並不值得除非你覺得能讓你的代碼更易讀（這對 `final`, `static`, `static final` 域同樣適用）。

## 關於測量

---

在優化之前，你應該決定你遇到了性能問題。你應該確保你能夠準確測量出現在的性能，否則你也不會知道優化是否真的有效。

本章節中所有的技巧都需要Benchmark（基準測試）的支持。Benchmark可以在 [code.google.com "dalvik" project](http://code.google.com/dalvik/project) 中找到

Benchmark是基於Java版本的 [Caliper](#) microbenchmarking（基準微測，譯者注）框架開發的。Microbenchmarking很難做準確，所以Caliper幫你完成這部分工作，甚至還幫你測了你沒想到需要測量的部分（因為，VM幫你管理了代碼優化，你很難知道這部分優化有多大效果）。我們強烈推薦使用Caliper來做你的基準微測工作。

我們也可以用[Traceview](#) 來測量，但是測量的數據是沒有經過JIT優化的，所以實際的效果應該是要比測量的數據稍微好些。

關於如何測量與調試，還可以參考下面兩篇文章：

- [Profiling with Traceview and dmtracedump](#)
- [Analysing Display and Performance with Systrace](#)

# 提升Layout的性能

編寫: allenlsy - 原文: <http://developer.android.com/training/improving-layouts/index.html>

Layout 是 Android 應用中直接影響用戶體驗的關鍵部分。如果實現的不好，你的 Layout 會導致程序非常佔用內存並且 UI 緩慢。Android SDK 帶有幫助你找到 Layout 性能問題的工具。結合本課內容使用它，你會用最小的內存空間實現流暢的 UI。

## 課程

### 優化Layout的層級

就像一個複雜的網頁會減慢載入速度，你的Layout結構如果太複雜，也會造成性能問題。本節教你如何使用SDK自帶工具來檢查Layout並找到瓶頸。

### 使用 `<include/>` 標籤重用Layout

如果你的程序的 UI 在不同地方重複使用某個 Layout，那本節教你如何創建高效的，可重用的 Layout 部件，並把它們“包含”到 UI Layout 中。

### 按需載入視圖

除了簡單的把一個 Layout 包含到另一箇中，你可能還想在程序開始後，僅當你的 Layout 對用戶可見時纔開始載入。本節告訴你如何分步載入 Layout 來提高初始性能。

### 優化ListView的滑動性能

如果你有一個每個列表項 (item) 都包含很多數據或者複雜數據的 ListView，那麼列表滾動的性能可能會降低。本節給你一些關於如何把滾動變得更流暢的提示。

# 優化layout的層級

編寫:allenlsy - 原文:<http://developer.android.com/training/improving-layouts/optimizing-layout.html>

一個常見的誤區是，用最基礎的 Layout 結構可以使 Layout 性能提高。然而，你的程序的每個組件和 Layout 都需要初始化、佈局和繪製。例如，嵌套的 LinearLayout 可能會使得 View 的層級結構過深。此外，嵌套使用了 `layout_weight` 參數的 LinearLayout 的計算量會尤其大，因為每個子元素都需要被測量兩次。這對需要多次重複 `inflate` 的 Layout 尤其需要注意，比如嵌套在 ListView 或 GridView 時。

本課中，你將學習使用 [Hierarchy Viewer](#) 和 [Layoutopt](#) 來檢查和優化 Layout。

## 檢查 Layout

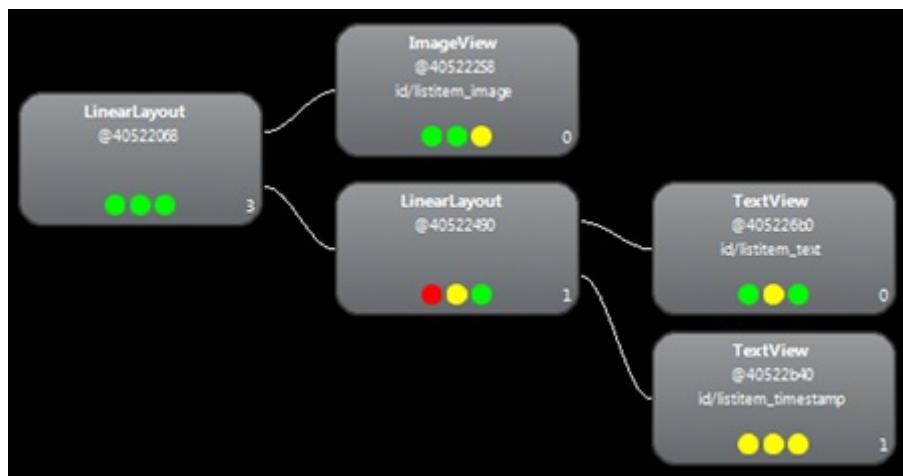
Android SDK 工具中有一個叫做 [Hierarchy Viewer](#) 的工具，能夠在程序運行時分析 Layout。你可以用這個工具找到 Layout 的性能瓶頸。

Hierarchy Viewer 會讓你選擇設備或者模擬器上正在運行的進程，然後顯示其 Layout 的樹型結構。每個塊上的交通燈分別代表了它在測量、佈局和繪畫時的性能，幫你找出瓶頸部分。

比如，下圖是 ListView 中一個列表項的 Layout。列表項裏，左邊放一個小位圖，右邊是兩個層疊的文字。像這種需要被多次 `inflate` 的 Layout，優化它們會有事半功倍的效果。



`hierarchyviewer` 這個工具在 `<sdk>/tools/` 中。當打開時，它顯示一張可使用設備的列表，和它正在運行的組件。點擊 `Load View Hierarchy` 來查看所選組件的層級。比如，下圖就是前一個圖中所示 Layout 的層級關係。



圖中，你可以看到一個三層結構，其中右下角的 TextView 在佈局的時候有問題。點擊其中的項會顯示每個步驟所花費的時間。這樣，誰花了多長時間在什麼哪個步驟上面，就清晰可見了。

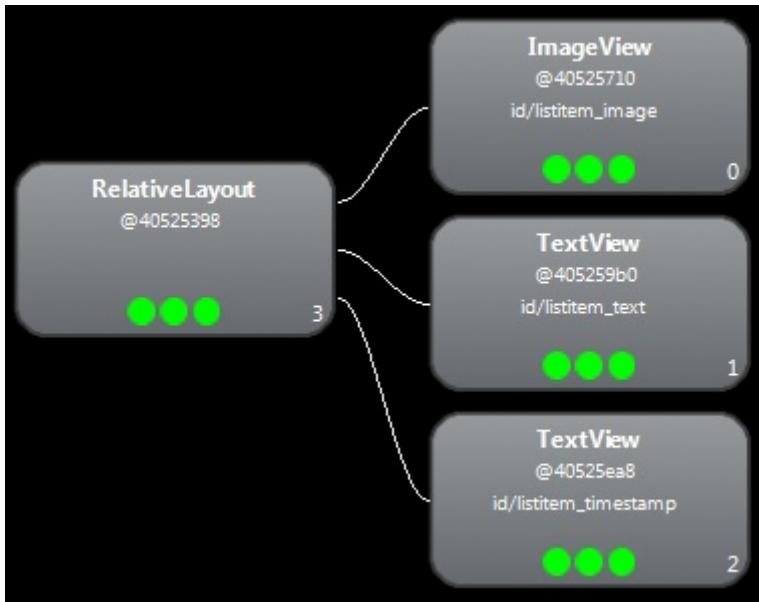


可以看到，渲染一個完整的列表項的時間就是：

- 測量: 0.977ms
- 佈局: 0.167ms
- 繪製: 2.717ms

## 修正 Layout

上面的 Layout 由於有這個嵌套的 LinearLayout 導致性能太慢，可能的解決辦法是將 Layout 層級扁平化——變淺變寬，而不是又窄又深。RelativeLayout 作為根節點時就可以達到目的。所以，當換成基於 RelativeLayout 的設計時，你的 Layout 變成了兩層。新的 Layout 變成這樣：



現在渲染列表項的時間：

- 測量: 0.598ms
- 佈局: 0.110ms
- 繪製: 2.146ms

可能看起來是很小的進步，但是由於它對列表中每個項都有效，這個時間要翻倍。

這個時間的主要差異是由於在 `LinearLayout` 中使用 `layout_weight` 所致，因為會減慢“測量”的速度。這只是一個正確使用各種 Layout 的例子，當你使用 `layout_weight` 時有必要慎重。

## 使用 Lint

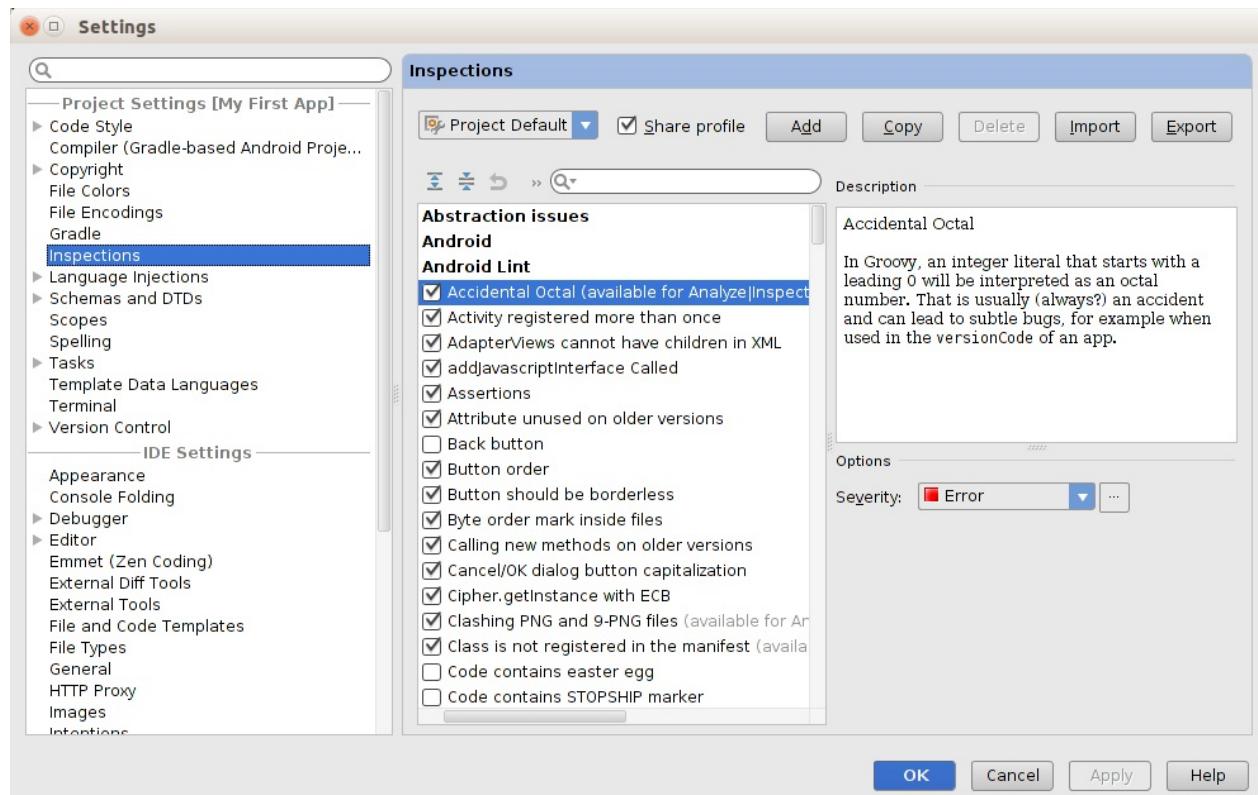
大部分叫做 lint 的編程工具，都是類似於代碼規範的檢測工具。比如JSLint，CSSLinkt， JSONLint 等等。譯者注。

運行 Lint 工具來檢查 Layout 可能的優化方法，是個很好的實踐。Lint 已經取代了 Layoutopt 工具，它擁有更強大的功能。Lint 中包含的一些檢測規則有：

- 使用compound drawable —— 用一個 `drawable` 替代一個包含 `ImageView` 和 `TextView` 的 `LinearLayout` 時會更有效率。
- 合併根 frame —— 如果 `FrameLayout` 是 Layout 的根節點，並且沒有使用 padding 或者背景等，那麼用 `merge` 標籤替代他們會稍微高效些。
- 沒用的子節點 —— 一個沒有子節點或者背景的 Layout 應該被去掉，來獲得更扁平的層級
- 沒用的父節點 —— 一個節點如果沒有兄弟節點，並且它不是 `ScrollView` 或根節點，沒有背景，這樣的節點應該直接被子節點取代，來獲得更扁平的層級
- 太深的 Layout —— Layout 的嵌套層數太深對性能有很大影響。嘗試使用更扁平的 Layout，比如 `RelativeLayout` 或 `GridLayout` 來提高性能。一般最多不超過10層。

另一個使用 Lint 的好處就是，它內置於 Android Studio 中。Lint 在你導編譯程序時自動運行。Android Studio 中，你可以為單獨的 build variant 或者所有 variant 運行 lint。

你也可以在 Android Studio 中管理檢測選項，在 File > Settings > Project Settings 中。檢測配置頁面會顯示支持的檢測項目。



Lint 有自動修復、提示建議和直接跳轉到問題處的功能。

# 使用`<include>`標籤重用layouts

編寫:allenlsy - 原文:<http://developer.android.com/training/improving-layouts/reusing-layouts.html>

雖然 Android 提供很多小的可重用的交互組件，你仍然可能需要重用複雜一點的組件，這也許會用到 Layout。為了高效重用整個的 Layout，你可以使用 `<include/>` 和 `<merge/>` 標籤把其他 Layout 嵌入當前 Layout。

重用 Layout 非常強大，它讓你可以創建複雜的可重用 Layout。比如，一個 yes/no 按鈕面板，或者帶有文字的自定義進度條。這也意味着，任何在多個 Layout 中重複出現的元素可以被提取出來，被單獨管理，再添加到 Layout 中。所以，雖然可以添加一個自定義 View 來實現單獨的 UI 組件，你可以更簡單的直接重用某個 Layout 文件。

## 創建可重用 Layout

如果你已經知道你需要重用的 Layout，就先創建一個新的 XML 文件並定義 Layout。比如，以下是一個來自 G-Kenya codelab 的 Layout，定義了一個需要添加到每個 Activity 中的標題欄（titlebar.xml）：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/titlebar_bg">

    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/gafricalogo" />

</FrameLayout>
```

根節點 View 就是你想添加入的 Layout 類型。

## 使用 `<include>` 標籤

使用 `<include>` 標籤，可以在 Layout 中添加可重用的組件。比如，這裏有一個來自 G-Kenya codelab 的 Layout 需要包含上面的那個標題欄：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/app_bg"
    android:gravity="center_horizontal">

    <include layout="@layout/titlebar" />

    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:padding="10dp" />

    ...
</LinearLayout>
```

你也可以覆寫被添加的 Layout 的所有 Layout 參數（任何 `android:layout_*` 屬性），通過在 `<include/>` 中聲明他們來完成。比如：

```
<include android:id="@+id/news_title"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    layout="@layout/title"/>
```

然而，如果你要在 `<include>` 中覆寫某些屬性，你必須先覆寫 `android:layout_height` 和 `android:layout_width`。

## 使用 `<merge>` 標籤

`<merge />` 標籤在你嵌套 Layout 時取消了 UI 層級中冗餘的 ViewGroup。比如，如果你有一個 Layout 是一個豎直方向的 LinearLayout，其中包含兩個連續的 View 可以在別的 Layout 中重用，那麼你會做一個 LinearLayout 來包含這兩個 View，以便重用。不過，當使用一個 LinearLayout 作為另一個 LinearLayout 的根節點時，這種嵌套 LinearLayout 的方式除了減慢你的 UI 性能外沒有任何意義。

為了避免這種情況，你可以用 `<merge>` 元素來替代可重用 Layout 的根節點。例如：

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/add"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/delete"/>

</merge>
```

現在，當你要將這個 Layout 包含到另一個 Layout 中時（並且使用了 `<include>` 標籤），系統會忽略 `<merge>` 標籤，直接把兩個 Button 放到 Layout 中 `<include>` 的所在位置。

# 按需加載視圖

編寫:allenlsy - 原文:<http://developer.android.com/training/improving-layouts/loading-on-demand.html>

有時你的 Layout 會用到不怎麼重用的複雜視圖。不管它是列表項 細節，進度顯示器，或是撤銷時的提示信息，你可以僅在需要的時候載入它們，提高 UI 渲染速度。

## 定義 ViewStub

**ViewStub** 是一個輕量的視圖，不需要大小信息，也不會在被加入的 Layout 中繪製任何東西。每個 ViewStub 只需要設置 `android:layout` 屬性來指定需要被 inflate 的 Layout 類型。

以下 ViewStub 是一個半透明的進度條覆蓋層。功能上講，它應該只在新的數據項被導入到應用程序時可見。

```
<ViewStub  
    android:id="@+id/stub_import"  
    android:inflatedId="@+id/panel_import"  
    android:layout="@layout/progress_overlay"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom" />
```

## 載入 ViewStub Layout

當你要載入用 ViewStub 聲明的 Layout 時，要麼用 `setVisibility(View.VISIBLE)` 設置它的可見性，要麼調用其 `inflate()` 方法。

```
((ViewStub) findViewById(R.id.stub_import)).setVisibility(View.VISIBLE);  
// or  
View importPanel = ((ViewStub) findViewById(R.id.stub_import)).inflate();
```

注意：`inflate()` 方法會在渲染完成後返回被 `inflate` 的視圖，所以如果你需要和這個 Layout 交互的話，你不需要再調用 `findViewById()` 去查找這個元素，。

一旦 ViewStub 可見或是被 `inflate` 了，ViewStub 元素就不存在了。取而代之的是被 `inflate` 的 Layout，其 id 是 ViewStub 上的 `android:inflatedId` 屬性。（ViewStub 的 `android:id` 屬性僅在 ViewStub 可見以前可用）

注意：ViewStub 的一個缺陷是，它目前不支持使用 `<merge/>` 標籤的 Layout。

# 使得ListView滑動順暢

編寫:allenlsy - 原文:<http://developer.android.com/training/improving-layouts/smooth-scrolling.html>

保持程序流暢的關鍵，是讓主線程（UI 線程）不要進行大量運算。你要確保在其他線程執行磁盤讀寫、網絡讀寫或是 SQL 操作等。為了測試你的應用的狀態，你可以啓用 [StrictMode](#)。

## 使用後臺線程

你應該把主線程中的耗時間的操作，提取到一個後臺線程（也叫做“工人線程”，英文 worker thread）中，使得主線程只關注 UI 繪畫。很多時候，使用 [AsyncTask](#) 是一個簡單的在主線程以外進行操作的方法。系統使用一個全局隊列來排列所有的 [AsyncTask](#)，並且一次運行其中的 [execute\(\)](#) 方法。這個行為是全局的，這意味着你不需要考慮自己定義線程池的事情。

在以下的例子中，一個 [AsyncTask](#) 被用於在後臺線程載入圖片，並在載入完成後把圖片放入 UI。當圖片正在載入時，它還顯示一個進度轉盤。

```
// Using an AsyncTask to load the slow images in a background thread
new AsyncTask<ViewHolder, Void, Bitmap>() {
    private ViewHolder v;

    @Override
    protected Bitmap doInBackground(ViewHolder... params) {
        v = params[0];
        return mFakeImageLoader.getImage();
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        super.onPostExecute(result);
        if (v.position == position) {
            // If this item hasn't been recycled already, hide the
            // progress and set and show the image
            v.progress.setVisibility(View.GONE);
            v.icon.setVisibility(View.VISIBLE);
            v.icon.setImageBitmap(result);
        }
    }
}.execute(holder);
```

從 Android 3.0 (API level 11) 開始，[AsyncTask](#) 有個新特性，那就是它可以在多個 CPU 核上運行。你可以調用 [executeOnExecutor\(\)](#) 來自動根據核數，在多核上執行任務，而不是調用 [execute\(\)](#)。

## 在 View Holder 中填入視圖對象

你的代碼可能在 ListView 滑動時經常使用 [findViewById\(\)](#)，這樣會降低性能。即使是 Adapter 返回一個用於回收的 [inflate](#) 後的視圖，你仍然需要查看這個元素並更新它。避免頻繁調用 [findViewById\(\)](#) 的方法之一，就是使用 View Holder（視圖佔位符）設計模式。

一個 ViewHolder 對象存儲了他的標簽下的每個視圖。這樣你不用頻繁查找這個元素。第一，你需要創建一個類來存儲你會用到的視圖。比如：

```
static class ViewHolder {
```

```
    TextView text;
    TextView timestamp;
    ImageView icon;
    ProgressBar progress;
    int position;
}
```

然後，在 Layout 的類中生成一個 ViewHolder 對象：

```
ViewHolder holder = new ViewHolder();
holder.icon = (ImageView) convertView.findViewById(R.id.listitem_image);
holder.text = (TextView) convertView.findViewById(R.id.listitem_text);
holder.timestamp = (TextView) convertView.findViewById(R.id.listitem_timestamp);
holder.progress = (ProgressBar) convertView.findViewById(R.id.progress_spinner);
convertView.setTag(holder);
```

這樣你就可以輕鬆獲取每個視圖，而不是用 `findViewById()` 來不斷查找視圖，節省了寶貴的運算時間。

# 優化電池壽命

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/monitoring-device-state/index.html>

顯然，手持設備的電量使用情況需要引起很大的重視。通過這一系列的課程，你將學會如何根據設備的狀態來改變App的某些行爲與功能。

通過在失去網絡連接時關閉後臺更新服務，在剩餘電量較低時減少更新數據的頻率等操作，你可以在不影響用戶體驗的前提下，確保App對電池壽命的影響減到最小。

## 課程

---

### 檢測電量與充電狀態

---

學習如何通過判斷與檢測當前電池電量以及充電狀態的變化，改變應用程序的更新頻率。

### 判斷並監測設備的底座狀態與類型

---

設備使用習慣的區別也會影響到刷新頻率的優化措施，這節課中將學習如何判斷與監測底座狀態及其種類來改變應用程序的行爲。

### 判斷並檢測網絡連接狀態

---

在沒有連接到互聯網的情況下，你是無法在線更新應用的。這一節課將學習如何根據網絡的連接狀態，改變後臺更新的頻率，以及如何在高帶寬傳輸任務開始前，判斷網絡連接類型(Wi-Fi/數據連接)。

### 按需操縱BroadcastReceiver

---

在Manifest清單文件中聲明的BroadcastReceiver可以在運行時切換其開啓狀態，這樣一來，我們就可以根據當前設備的狀態，禁用那些沒有必要開啓的BroadcastReceiver。在這一節課將學習如何通過切換這些BroadcastReceiver的開啓狀態，以及如何根據設備的狀態延遲某一操作的執行時機，來提高應用的效率。

# 監測電池的電量與充電狀態

編寫:kesenhoo - 原文:<http://developer.android.com/training/monitoring-device-state/battery-monitoring.html>

當你想通過改變後臺更新操作的頻率來減少對電池壽命的影響時，那麼首先需要檢查當前電量與充電狀態。

執行應用更新對電池壽命的影響是與電量和充電狀態密切相關的。當使用交流電對設備充電時，更新操作的影響可以忽略不計，所以在大多數情況下，如果使用壁式充電器對設備進行充電，我們可以將刷新頻率設置到最大。相反的，如果設備沒有在充電狀態，那麼我們就需要儘量減少設備的更新操作來延長電池的續航能力。

同樣的，如果我們監測到電量即將耗盡時，那麼應該儘可能降低甚至停止更新操作。

## 判斷當前充電狀態

首先來看一下應該如何確定當前的充電狀態。BatteryManager會廣播一個帶有電池與充電詳情的Sticky Intent

因為廣播的是一個sticky Intent，所以不需要註冊BroadcastReceiver。僅僅只需要調用一個以 `null` 作為Receiver參數的 `registerReceiver()` 方法就可以了。如下面的代碼片段中展示的那樣，它返回了保存當前電池信息的Intent。你也可以在這裏傳入一個實際的BroadcastReceiver對象，但這並不是必須的。

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);
```

我們可以提取出當前的充電狀態，以及設備處於充電時，是通過USB還是交流充電器充電的。

```
// Are we charging / charged?
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
                     status == BatteryManager.BATTERY_STATUS_FULL;

// How are we charging?
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;
```

通常，我們可以在設備使用交流充電時最大化後臺更新頻率，在使用USB充電時降低更新頻率，在非充電狀態時，將更新頻率進一步降低。

## 監測充電狀態的改變

充電狀態隨時可能改變，所以我們應該檢查充電狀態的改變來調整更新頻率。

BatteryManager會在設備連接或者斷開充電器的時候廣播一個Action。即使應用沒有運行，我們也應該接收這些事件的廣播，主要原因是因為這些事件會影響到應用啓動（從而進行更新）的頻率，因此我們應該在Manifest文件裏面註冊一個BroadcastReceiver來監聽含有ACTION\_POWER\_CONNECTED 與 ACTION\_POWER\_DISCONNECTED的Intent。

```
<receiver android:name=".PowerConnectionReceiver">
```

```
<intent-filter>
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
    <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
</intent-filter>
</receiver>
```

我們可以在該BroadcastReceiver的實現中，提取出當前的充電狀態，如下所示：

```
public class PowerConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
        boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
            status == BatteryManager.BATTERY_STATUS_FULL;

        int chargePlug = intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
        boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
        boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;
    }
}
```

## 判斷當前電池電量

在一些情況下，獲取到當前電池電量也很有幫助。我們可以在獲知電量少於某個級別的時候減少後臺的更新頻率。我們可以通過電池狀態Intent獲取到電池電量與容量等信息，如下所示：

```
int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);

float batteryPct = level / (float)scale;
```

## 檢測電量的有效改變

我們不能不停地監測電池狀態，實際上這也是不必要的。通常來說，不間斷地監測電量信息對電池的影響會遠大於應用本身對電池的影響。所以我們應該僅監測電量的一些顯著性變化，特別是當設備進入或者離開低電量狀態時。

在下面的Manifest文件片段中，BroadcastReceiver僅僅監聽 ACTION\_BATTERY\_LOW 與 ACTION\_BATTERY\_OKAY，這樣它就只會在設備電量進入低電量或者離開低電量的時候被觸發。

```
<receiver android:name=".BatteryLevelReceiver">
<intent-filter>
    <action android:name="android.intent.action.ACTION_BATTERY_LOW"/>
    <action android:name="android.intent.action.ACTION_BATTERY_OKAY"/>
</intent-filter>
</receiver>
```

通常我們都需要在進入低電量的情況下，關閉所有後臺更新來維持設備的續航，因為這個時候做任何更新等操作都極有可能是無用的，因為也許在你還沒來得及處理更新的數據時，設備就因電量耗盡而自動關機了。

在很多時候，用戶往往會將設備放入某種底座中充電（譯註：比如車載的底座式充電器），在下一節課程當中，我們將會學習如何確定當前的底座狀態，以及如何監聽設備底座的變化。

# 判斷並監測設備的底座狀態與類型

編寫:kesenhoo - 原文:<http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html>

Android設備可以放置在許多不同的底座中，包括車載底座，家庭底座還有數字信號底座以及模擬信號底座等。由於許多底座會向設備充電，因此底座狀態通常與充電狀態密切相關。

你的應用類型決定了底座類型會對更新頻率產生怎樣的影響。對於一個體育類應用，可以讓設備在筆記本底座狀態下增加更新的頻率，或者當設備在車載底座狀態下停止更新。相反的，如果你的後臺服務用來更新交通數據，你也可以選擇在車載底座模式下最大化更新的頻率。

底座狀態也是以Sticky Intent方式來廣播的，這樣可以通過查詢Intent裏面的數據來判斷目前設備是否放置在底座中，以及底座的類型。

## 判斷當前底座狀態

底座狀態的具體信息會以Extra數據的形式，包含在具有ACTION\_DOCK\_EVENT這一Action的某個Sticky廣播中，因此，你不需要為其註冊一個BroadcastReceiver。如下所示，僅需要將 null 作為參數傳遞給registerReceiver()方法就可以了：

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_DOCK_EVENT);
Intent dockStatus = context.registerReceiver(null, ifilter);
```

你可以從 EXTRA\_DOCK\_STATE 這一Extra數據中，提取出當前的底座狀態：

```
int dockState = battery.getIntExtra(EXTRA_DOCK_STATE, -1);
boolean isDocked = dockState != Intent.EXTRA_DOCK_STATE_UNDOCKED;
```

## 判斷當前底座類型

如果設備被放置在了底座中，那麼它可以有下面四種底座類型：

- Car
- Desk
- Low-End (Analog) Desk
- High-End (Digital) Desk

注意最後兩種底座類型僅在API Level 11及以後版本的Android系統中才被支持。如果你只在乎底座的類型而不管它是數字的還是模擬的，那麼可以僅監測三種類型：

```
boolean isCar = dockState == EXTRA_DOCK_STATE_CAR;
boolean isDesk = dockState == EXTRA_DOCK_STATE_DESK ||
                dockState == EXTRA_DOCK_STATE_LE_DESK ||
                dockState == EXTRA_DOCK_STATE_HE_DESK;
```

## 監測底座狀態或者類型的改變

---

當設備被放置在或者拔出底座時，系統會發出一個具有ACTION\_DOCK\_EVENT這一Action的廣播。為了監聽底座狀態的變化，我們只需要在應用的Manifest文件中註冊一個BroadcastReceiver，如下所示：

```
<action android:name="android.intent.action.ACTION_DOCK_EVENT"/>
```

之於該BroadcastReceiver的具體實現，可以參考前面提到的那些方法，以此來提取出當前的底座類型和狀態。

# 判斷並監測網絡連接狀態

編寫:kesenhoo - 原文:<http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html>

重複鬧鐘和後臺服務最常見的功能之一，是用來從網絡上獲取應用更新，存儲數據或者執行大文件的下載。但是如果沒有獲得網絡連接，或者連接的速度太慢以至於下載無法完成，那麼就沒有必要喚醒設備並執行那些更新等操作了。

我們可以使用`ConnectivityManager`來檢查設備是否連接到網絡，以及網絡的類型（譯註：通過網絡的連接狀況改變，相應的改變app的行爲，減少無謂的操作，從而延長設備的續航能力）。

## 判斷當前是否有網絡連接

如果沒有網絡連接，那麼就沒有必要做那些需要聯網的事情。下面的代碼片段展示瞭如何通過`ConnectivityManager`檢查當前活動的網絡類型，並確定它是否可以連接到互聯網：

```
ConnectivityManager cm =  
    (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);  
  
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();  
boolean isConnected = activeNetwork != null &&  
    activeNetwork.isConnectedOrConnecting();
```

## 判斷連接網絡的類型

我們還可以獲取到當前的網絡連接類型。

設備通常可以有移動網絡，WiMax，Wi-Fi與以太網連接等類型。通過查詢當前活動的網絡類型，可以根據網絡的帶寬對更新頻率進行調整：

```
boolean isWiFi = activeNetwork.getType() == ConnectivityManager.TYPE_WIFI;
```

移動網絡的使用費會比Wi-Fi更高，所以多數情況下，如果設備正在使用移動網絡，我們應該減少應用的更新頻率；同樣地，還應該臨時地掛起一些文件下載任務直到有Wi-Fi連接時再繼續下載。

如果已經關閉了更新操作，那麼需要監聽網絡連接的變化，這樣就可以在建立了互聯網訪問之後，重新恢復它們。

## 監聽網絡連接的變化

當網絡連接發生改變時，`ConnectivityManager`會廣播`CONNECTIVITY_ACTION`（`android.net.conn.CONNECTIVITY_CHANGE`）的Action消息。我們可以在Manifest文件裏面註冊一個BroadcastReceiver，來監聽這些變化，並適當地恢復（或掛起）你的後臺更新：

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
```

設備的網絡變化可能會比較頻繁，因此每當你在移動網絡與Wi-Fi之間切換的時候，這一廣播就會被觸發。因此，我們

可以僅在之前的更新或者下載任務被掛起的時候去監聽這一廣播（用來恢復那些任務）。通常我們可以在開始更新前檢查一下網絡連接，如果當前沒有連接到互聯網，那麼就將更新任務掛起，直到連接恢復。

上述方法會涉及到Broadcast Receiver開啓狀態的切換，這一內容會在下一節課中展開。

# 按需操控BroadcastReceiver

編寫:kesenhoo - 原文:<http://developer.android.com/training/monitoring-device-state/manifest-receivers.html>

監測設備狀態變化最簡單的方法，是為你所要監聽的每一個狀態創建一個BroadcastReceiver，並在Manifest文件中註冊它們。之後就可以在每一個BroadcastReceiver中，根據當前設備的狀態調整一些計劃任務。

上述方法的副作用是：一旦你的接收器收到了廣播，應用就會喚醒設備。喚醒的頻率可能會遠高於需要的頻率。

更好的方法是在程序運行時開啓或者關閉BroadcastReceiver。這樣的話，你就可以讓這些接收器僅在需要的時候被激活。

## 切換是否開啓接收器以提高效率

我們可以使用PackageManager來切換任何一個在Manifest裏面定義好的組件的開啓狀態。通過下面的方法可以開啓或者關閉任何一個BroadcastReceiver：

```
ComponentName receiver = new ComponentName(context, myReceiver.class);
PackageManager pm = context.getPackageManager();
pm.setComponentEnabledSetting(receiver,
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP)
```

使用這種技術，如果我們確定網絡連接已經斷開，那麼可以在這個時候關閉除了監聽網絡狀態變化的接收器之外的其它所有接收器。

相反的，一旦重新建立網絡連接，我們可以停止監聽網絡連接的改變，而僅僅在執行需要聯網的操作之前判斷當前網絡是否可以用。

同樣地，你可以使用上面的技術來暫緩一個需要更高帶寬的下載任務。這僅需要啓用一個監聽網絡連接變化的BroadcastReceiver，並在連接到Wi-Fi時，初始化下載任務。

# 多線程操作

---

編寫:AllenZheng1991 - 原文:<http://developer.android.com/training/multiple-threads/index.html>

把一個相對耗時且數據操作複雜的任務分割成多個小的操作，然後分別運行在多個線程上，這能夠提高完成任務的速度和效率。在多核CPU的設備上，系統可以並行運行多個線程，而不需要讓每個子操作等待CPU的時間片切換。例如，如果要解碼大量的圖片文件並以縮略圖的形式把圖片顯示在屏幕上，當你把每個解碼操作單獨用一個線程去執行時，會發現速度快了很多。

這個章節會向你展示如何在一個Android應用中創建和使用多線程，以及如何使用線程池對象（thread pool object）。你還將瞭解到如何使得代碼運行在指定的線程中，以及如何讓你創建的線程和UI線程進行通信。

## Sample Code

---

點擊下載：[ThreadSample](#)

## 課程

---

### 在一個線程中執行一段特定的代碼

學習如何通過實現[Runnable](#)接口定義一個線程類，讓你寫的代碼能在單獨的一個線程中執行。

### 為多線程創建線程池

學習如何創建一個能管理線程池和任務隊列的對象，需要使用一個叫[ThreadPoolExecutor](#)的類。

### 在線程池中的一個線程裏執行代碼

學習如何讓線程池裏的一個線程執行一個任務。

### 與UI線程通信

學習如何讓線程池裏的一個普通線程與UI線程進行通信。

# 在一個線程中執行一段特定的代碼

編寫:AllenZheng1991 - 原文:<http://developer.android.com/training/multiple-threads/define-runnable.html>

這一課向你展示瞭如何通過實現 `Runnable` 接口得到一個能在重寫的 `Runnable.run()` 方法中執行一段代碼的單獨的線程。另外你可以傳遞一個 `Runnable` 對象到另一個對象，然後這個對象可以把它附加到一個線程，並執行它。一個或多個執行特定操作的 `Runnable` 對象有時也被稱為一個任務。

`Thread` 和 `Runnable` 只是兩個基本的線程類，通過他們能發揮的作用有限，但是他們是強大的Android線程類的基礎類，例如Android中的 `HandlerThread`, `AsyncTask` 和 `IntentService` 都是以它們為基礎。`Thread` 和 `Runnable` 同時也是 `ThreadPoolExecutor` 類的基礎。`ThreadPoolExecutor` 類能自動管理線程和任務隊列，甚至可以並行執行多個線程。

## 定義一個實現 `Runnable` 接口的類

直接了當的方法是通過實現 `Runnable` 接口去定義一個線程類。例如：

```
public class PhotoDecodeRunnable implements Runnable {  
    ...  
    @Override  
    public void run() {  
        /*  
         * 把你想要在線程中執行的代碼寫在這裏  
         */  
        ...  
    }  
    ...  
}
```

## 實現 `run()` 方法

在一個類裏，`Runnable.run()` 包含執行了的代碼。通常在 `Runnable` 中執行任何操作都是可以的，但需要記住的是，因為 `Runnable` 不會在 UI 線程中運行，所以它不能直接更新 UI 對象，例如 `View` 對象。為了與 UI 對象進行通信，你必須使用另一項技術，在 [與 UI 線程進行通信](#) 這一課中我們會對其進行描述。

在 `Runnable.run()` 方法的開始的地方通過調用參數為 `THREAD_PRIORITY_BACKGROUND` 的 `Process.setThreadPriority()` 方法來設置線程使用的是後臺運行優先級。這個方法減少了通過 `Runnable` 創建的線程和 UI 線程之間的資源競爭。

你還應該通過在 `Runnable` 自身中調用 `Thread.currentThread()` 來存儲一個引用到 `Runnable` 對象的線程。

下面這段代碼展示瞭如何創建 `run()` 方法：

```
class PhotoDecodeRunnable implements Runnable {  
    ...  
    /*  
     * 定義要在這個任務中執行的代碼  
     */  
    @Override  
    public void run() {  
        // 把當前的線程變成後臺執行的線程  
        android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_BACKGROUND);  
        ...  
    }  
}
```

```
* 在PhotoTask實例中存儲當前線程，以至於這個實例能中斷這個線程
*/
mPhotoTask.setImageDecodeThread(Thread.currentThread());
...
}

...
}
```

# # # ### ## # 爲多線程創建管理器

編寫:AllenZheng1991 - 原文:<http://developer.android.com/training/multiple-threads/create-threadpool.html>

在前面的課程中展示瞭如何在單獨的一個線程中執行一個任務。如果你的線程只想執行一次，那麼上一課的內容已經能滿足你的需要了。

如果你想在一個數據集中重複執行一個任務，而且你只需要一個執行運行一次。這時，使用一個IntentService將能滿足你的需求。為了在資源可用的時候自動執行任務，或者允許不同的任務同時執行（或前後兩者），你需要提供一個管理線程的集合。為了做這個管理線程的集合，使用一個ThreadPoolExecutor實例，當一個線程在它的線程池中變得不受約束時，它會運行隊列中的一個任務。為了能執行這個任務，你所需要做的就是把它加入到這個隊列。

一個線程池能運行多個並行的任務實例，因此你要能保證你的代碼是線程安全的，從而你需要給會被多個線程訪問的變量附上同步代碼塊(synchronized block)。當一個線程在對一個變量進行寫操作時，通過這個方法將能阻止另一個線程對該變量進行讀取操作。典型的，這種情況會發生在靜態變量上，但同樣它也能突然發生在任意一個只實例化一次。為了學到更多的相關知識，你可以閱讀[進程與線程](#)這一API指南。

## 定義線程池類

在自己的類中實例化ThreadPoolExecutor類。在這個類裏需要做以下事：

### 1. 為線程池使用靜態變量

為了有一個單一控制點用來限制CPU或涉及網絡資源的Runnable類型，你可能需要有一個能管理所有線程的線程池，且每個線程都會是單個實例。比如，你可以把這個作爲一部分添加到你的全局變量的聲明中去：

```
public class PhotoManager {  
    ...  
    static {  
        ...  
        // Creates a single static instance of PhotoManager  
        sInstance = new PhotoManager();  
    }  
    ...  
}
```

### 2. 使用私有構造方法

讓構造方法私有從而保證這是一個單例，這意味着你不需要在同步代碼塊(synchronized block)中額外訪問這個類：

```
public class PhotoManager {  
    ...  
    /**  
     * Constructs the work queues and thread pools used to download  
     * and decode images. Because the constructor is marked private,  
     * it's unavailable to other classes, even in the same package.  
     */  
    private PhotoManager() {  
        ...  
    }  
}
```

### 3. 通過調用線程池類裏的方法開啓你的任務

在線程池類中定義一個能添加任務到線程池隊列的方法。例如：

```
public class PhotoManager {  
    ...  
    // Called by the PhotoView to get a photo  
    static public PhotoTask startDownload(  
        PhotoView imageView,  
        boolean cacheFlag) {  
        ...  
        // Adds a download task to the thread pool for execution  
        sInstance.  
        mDownloadThreadPool.  
        execute(downloadTask.getHTTPDownloadRunnable());  
        ...  
    }  
}
```

#### 4. 在構造方法中實例化一個Handler，且將它附加到你APP的UI線程。

一個Handler允許你的APP安全地調用UI對象（例如 View對象）的方法。大多數UI對象只能從UI線程安全的代碼中被修改。這個方法將會在[與UI線程進行通信\(Communicate with the UI Thread\)](#)這一課中進行詳細的描述。例如：

```
private PhotoManager() {  
    ...  
    // Defines a Handler object that's attached to the UI thread  
    mHandler = new Handler(Looper.getMainLooper()) {  
        /*  
         * handleMessage() defines the operations to perform when  
         * the Handler receives a new Message to process.  
         */  
        @Override  
        public void handleMessage(Message inputMessage) {  
            ...  
        }  
        ...  
    }  
}
```

## 確定線程池的參數

一旦有了整體的類結構，你可以開始定義線程池了。為了初始化一個ThreadPoolExecutor對象，你需要提供以下數值：

### 1. 線程池的初始化大小和最大的大小

這個是指最初分配給線程池的線程數量，以及線程池中允許的最大線程數量。在線程池中擁有的線程數量主要取決於你的設備的CPU內核數。

這個數字可以從系統環境中獲得：

```
public class PhotoManager {  
    ...  
    /*  
     * Gets the number of available cores  
     * (not always the same as the maximum number of cores)  
     */  
    private static int NUMBER_OF_CORES =  
        Runtime.getRuntime().availableProcessors();  
}
```

這個數字可能並不反映設備的物理核心數量，因為一些設備根據系統負載關閉了一個或多個CPU內核，對於這樣的設備，`availableProcessors()`方法返回的是處於活動狀態的內核數量，可能少於設備的實際內核總數。

## 2.線程保持活動狀態的持續時間和時間單位

這個是指線程被關閉前保持空閒狀態的持續時間。這個持續時間通過時間單位值進行解譯，是[TimeUnit\(\)](#)中定義的常量之一。

## 3.一個任務隊列

這個傳入的隊列由[ThreadPoolExecutor](#)獲取的[Runnable](#)對象組成。為了執行一個線程中的代碼，一個線程池管理者從先進先出的隊列中取出一個[Runnable](#)對象且把它附加到一個線程。當你創建線程池時需要提供一個隊列對象，這個隊列對象類必須實現[BlockingQueue](#)接口。為了滿足你的APP的需求，你可以選擇一個Android SDK中已經存在的隊列實現類。為了學習更多相關的知識，你可以看一下[ThreadPoolExecutor](#)類的概述。下面是一個使用[LinkedBlockingQueue](#)實現的例子：

```
public class PhotoManager {  
    ...  
    private PhotoManager() {  
        ...  
        // A queue of Runnables  
        private final BlockingQueue<Runnable> mDecodeWorkQueue;  
        ...  
        // Instantiates the queue of Runnables as a LinkedBlockingQueue  
        mDecodeWorkQueue = new LinkedBlockingQueue<Runnable>();  
        ...  
    }  
    ...  
}
```

# 創建一個線程池

為了創建一個線程池，可以通過調用[ThreadPoolExecutor\(\)](#)構造方法初始化一個線程池管理者對象，這樣就能創建和管理一組可約束的線程了。如果線程池的初始化大小和最大大小相同，[ThreadPoolExecutor](#)在實例化的時候就會創建所有的線程對象。例如：

```
private PhotoManager() {  
    ...  
    // Sets the amount of time an idle thread waits before terminating  
    private static final int KEEP_ALIVE_TIME = 1;  
    // Sets the Time Unit to seconds  
    private static final TimeUnit KEEP_ALIVE_TIME_UNIT = TimeUnit.SECONDS;  
    // Creates a thread pool manager  
    mDecodeThreadPool = new ThreadPoolExecutor(  
        NUMBER_OF_CORES,           // Initial pool size  
        NUMBER_OF_CORES,           // Max pool size  
        KEEP_ALIVE_TIME,  
        KEEP_ALIVE_TIME_UNIT,  
        mDecodeWorkQueue);  
}
```

# 啓動與停止線程池中的線程

編寫:AllenZheng1991 - 原文:<http://developer.android.com/training/multiple-threads/run-code.html>

在前面的課程中向你展示瞭如何去定義一個可以管理線程池且能在他們中執行任務代碼的類。在這一課中我們將向你展示如何在線程池中執行任務代碼。為了達到這個目的，你需要把任務添加到線程池的工作隊列中去，當一個線程變成可運行狀態時，ThreadPoolExecutor從工作隊列中取出一個任務，然後在該線程中執行。

這節課同時也向你展示瞭如何去停止一個正在執行的任務，這個任務可能在剛開始執行時是你想要的，但後來發現它所做的工作並不是你所需要的。你可以取消線程正在執行的任務，而不是浪費處理器的運行時間。例如你正在從網絡上下載圖片且對下載的圖片進行了緩存，當檢測到正在下載的圖片在緩存中已經存在時，你可能希望停止這個下載任務。當然，這取決於你編寫APP的方式，因為可能壓在你啓動下載任務之前無法獲知是否需要啓動這個任務。

## 啓動線程池中的線程執行任務

為了在一個特定的線程池的線程裏開啓一個任務，可以通過調用ThreadPoolExecutor.execute()，它需要提供一個Runnable類型的參數，這個調用會把該任務添加到這個線程池中的工作隊列。當一個空閒的線程進入可執行狀態時，線程管理者從工作隊列中取出等待時間最長的那個任務，並且在線程中執行它。

```
public class PhotoManager {  
    public void handleState(PhotoTask photoTask, int state) {  
        switch (state) {  
            // The task finished downloading the image  
            case DOWNLOAD_COMPLETE:  
                // Decodes the image  
                mDecodeThreadPool.execute(  
                    photoTask.getPhotoDecodeRunnable());  
                ...  
            }  
            ...  
        }  
    }  
}
```

當ThreadPoolExecutor在一個線程中開啓一個Runnable後，它會自動調用Runnable的run()方法。

## 中斷正在執行的代碼

為了停止執行一個任務，你必須中斷執行這個任務的線程。在準備做這件事之前，當你創建一個任務時，你需要存儲處理該任務的線程。例如：

```
class PhotoDecodeRunnable implements Runnable {  
    // Defines the code to run for this task  
    public void run() {  
        /*  
         * Stores the current Thread in the  
         * object that contains PhotoDecodeRunnable  
         */  
        mPhotoTask.setImageDecodeThread(Thread.currentThread());  
        ...  
    }  
}
```

想要中斷一個線程，你可以調用[Thread.interrupt\(\)](#)。需要注意的是這些線程對象都被系統控制，系統可以在你的APP進程之外修改他們。因為這個原因，在你要中斷一個線程時，你需要把這段代碼放在一個同步代碼塊中對這個線程的訪問加鎖來解決這個問題。例如：

```
public class PhotoManager {
    public static void cancelAll() {
        /*
         * Creates an array of Runnables that's the same size as the
         * thread pool work queue
         */
        Runnable[] runnableArray = new Runnable[mDecodeWorkQueue.size()];
        // Populates the array with the Runnables in the queue
        mDecodeWorkQueue.toArray(runnableArray);
        // Stores the array length in order to iterate over the array
        int len = runnableArray.length;
        /*
         * Iterates over the array of Runnables and interrupts each one's Thread.
         */
        synchronized (sInstance) {
            // Iterates over the array of tasks
            for (int runnableIndex = 0; runnableIndex < len; runnableIndex++) {
                // Gets the current thread
                Thread thread = runnableArray[taskArrayIndex].mThread;
                // if the Thread exists, post an interrupt to it
                if (null != thread) {
                    thread.interrupt();
                }
            }
        }
        ...
    }
}
```

在大多數情況下，通過調用[Thread.interrupt\(\)](#)能立即中斷這個線程，然而他只能停止那些處於等待狀態的線程，卻不能中斷那些佔據CPU或者耗時的連接網絡的任務。爲了避免拖慢系統速度或造成系統死鎖，在嘗試執行耗時操作之前，你應該測試當前是否存在處於掛起狀態的中斷請求：

```
/*
 * Before continuing, checks to see that the Thread hasn't
 * been interrupted
 */
if (Thread.interrupted()) {
    return;
}
...
// Decodes a byte array into a Bitmap (CPU-intensive)
BitmapFactory.decodeByteArray(
    imageBuffer, 0, imageBuffer.length, bitmapOptions);
...
```

# 與UI線程通信

編寫:AllenZheng1991 - 原文:<http://developer.android.com/training/multiple-threads/communicate-ui.html>

在前面的課程中你學習瞭如何在一個被`ThreadPoolExecutor`管理的線程中開啓一個任務。最後這一節課將會向你展示如何從執行的任務中發送數據給運行在UI線程中的對象。這個功能允許你的任務可以做後臺工作，然後把得到的結果數據轉移給UI元素使用，例如位圖數據。

任何一個APP都有自己特定的一個線程用來運行UI對象，比如`View`對象，這個線程我們稱之為UI線程。只有運行在UI線程中的對象能訪問運行在其它線程中的對象。因為你的任務執行的線程來自一個線程池而不是執行在UI線程，所以他們不能訪問UI對象。為了把數據從一個後臺線程轉移到UI線程，需要使用一個運行在UI線程裏的`Handler`。

## 在UI線程中定義一個Handler

`Handler`屬於Android系統的線程管理框架的一部分。一個`Handler`對象用於接收消息和執行處理消息的代碼。一般情況下，如果你為一個新線程創建了一個`Handler`，你還需要創建一個`Handler`，讓它與一個已經存在的線程關聯，用於這兩個線程之間的通信。如果你把一個`Handler`關聯到UI線程，處理消息的代碼就會在UI線程中執行。

你可以在一個用於創建你的線程池的類的構造方法中實例化一個`Handler`對象，並把它定義為全局變量，然後通過使用`Handler (Looper)`這一構造方法實例化它，用於關聯到UI線程。`Handler(Looper)`這一構造方法需要傳入了一個`Looper`對象，它是Android系統的線程管理框架中的另一部分。當你在一個特定的`Looper`實例的基礎上去實例化一個`Handler`時，這個`Handler`與`Looper`運行在同一個線程裏。例如：

```
private PhotoManager() {  
    ...  
    // Defines a Handler object that's attached to the UI thread  
    mHandler = new Handler(Looper.getMainLooper()) {  
        ...  
    };  
}
```

在這個`Handler`裏需要重寫`handleMessage()`方法。當這個`Handler`接收到由另外一個線程管理的`Handler`發送過來的新消息時，Android系統會自動調用這個方法，而所有線程對應的`Handler`都會收到相同信息。例如：

```
/*  
 * handleMessage() defines the operations to perform when  
 * the Handler receives a new Message to process.  
 */  
@Override  
public void handleMessage(Message inputMessage) {  
    // Gets the image task from the incoming Message object.  
    PhotoTask photoTask = (PhotoTask) inputMessage.obj;  
    ...  
}  
...  
}
```

下一部分將向你展示如何用`Handler`轉移數據。

## 把數據從一個任務中轉移到UI線程

為了從一個運行在後臺線程的任務對象中轉移數據到UI線程中的一個對象，首先需要存儲任務對象中的數據和UI對象的

引用；接下來傳遞任務對象和狀態碼給實例化Handler的那個對象。在這個對象裏，發送一個包含任務對象和狀態的Message給Handler也運行在UI線程中，所以它可以把數據轉移到UI線程。

## 在任務對象中存儲數據

比如這裏有一個Runnable，它運行在一個編碼了一個Bitmap且存儲這個Bitmap到父類PhotoTask對象裏的後臺線程。這個Runnable同樣也存儲了狀態碼DECODE\_STATE\_COMPLETED。

```
// A class that decodes photo files into Bitmaps
class PhotoDecodeRunnable implements Runnable {

    ...
    PhotoDecodeRunnable(PhotoTask downloadTask) {
        mPhotoTask = downloadTask;
    }
    ...
    // Gets the downloaded byte array
    byte[] imageBuffer = mPhotoTask.getByteBuffer();
    ...
    // Runs the code for this task
    public void run() {
        ...
        // Tries to decode the image buffer
        returnBitmap = BitmapFactory.decodeByteArray(
            imageBuffer,
            0,
            imageBuffer.length,
            bitmapOptions
        );
        ...
        // Sets the ImageView Bitmap
        mPhotoTask.setImage(returnBitmap);
        // Reports a status of "completed"
        mPhotoTask.handleDecodeState(DECODE_STATE_COMPLETED);
        ...
    }
    ...
}
```

PhotoTask類還包含一個用於給ImageView顯示Bitmap的handler。雖然Bitmap和ImageView的引用在同一個對象中，但你不能把這個Bitmap分配給ImageView去顯示，因為它們並沒有運行在UI線程中。

這時，下一步應該發送這個狀態給 PhotoTask 對象。

## 發送狀態取決於對象層次

PhotoTask是下一個層次更高的對象，它包含將要展示數據的編碼數據和View對象的引用。它會收到一個來自PhotoDecodeRunnable的狀態碼，並把這個狀態碼單獨傳遞到一個包含線程池和Handler實例的對象：

```
public class PhotoTask {
    ...
    // Gets a handle to the object that creates the thread pools
    sPhotoManager = PhotoManager.getInstance();
    ...
    public void handleDecodeState(int state) {
        int outState;
        // Converts the decode state to the overall state.
        switch(state) {
            case PhotoDecodeRunnable.DECODE_STATE_COMPLETED:
                outState = PhotoManager.TASK_COMPLETE;
                break;
            ...
        }
    }
}
```

```

    ...
    // Calls the generalized state method
    handleState(outState);
}

...
// Passes the state to PhotoManager
void handleState(int state) {
/*
 * Passes a handle to this task and the
 * current state to the class that created
 * the thread pools
 */
sPhotoManager.handleState(this, state);
}

...
}

```

## 轉移數據到UI

從PhotoTask對象那裏，PhotoManager對象收到了一個狀態碼和一個PhotoTask對象的handler。因為狀態碼是TASK\_COMPLETE，所以創建一個Message應該包含狀態和任務對象，然後把它發送給Handler：

```

public class PhotoManager {
    ...

    // Handle status messages from tasks
    public void handleState(PhotoTask photoTask, int state) {
        switch (state) {
            ...
            // The task finished downloading and decoding the image
            case TASK_COMPLETE:
                /*
                 * Creates a message for the Handler
                 * with the state and the task object
                 */
                Message completeMessage =
                    mHandler.obtainMessage(state, photoTask);
                completeMessage.sendToTarget();
                break;
            ...
        }
    ...
}

```

最終，Handler.handleMessage()會檢查每個傳入進來的Message，如果狀態碼是TASK\_COMPLETE，這時任務就完成了，而傳入的Message裏的PhotoTask對象裏同時包含一個Bitmap和一個ImageView。因為Handler.handleMessage()運行在UI線程裏，所以它能安全地轉移Bitmap數據給ImageView：

```

private PhotoManager() {
    ...
    mHandler = new Handler(Looper.getMainLooper()) {
        @Override
        public void handleMessage(Message inputMessage) {
            // Gets the task from the incoming Message object.
            PhotoTask photoTask = (PhotoTask) inputMessage.obj;
            // Gets the ImageView for this task
            PhotoView localView = photoTask.getPhotoView();
            ...
            switch (inputMessage.what) {
                ...
                // The decoding is done
                case TASK_COMPLETE:
                    /*
                     * Moves the Bitmap from the task
                     * to the View
                     */

```

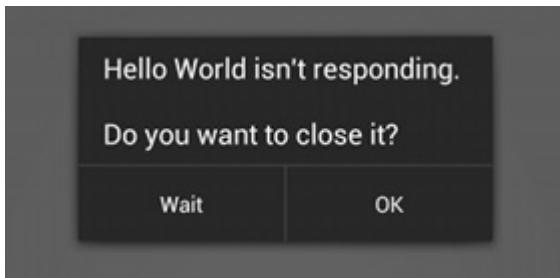
```
        localView.setImageBitmap(photoTask.getImage());
        break;
    ...
    default:
        /*
         * Pass along other messages from the UI
         */
        super.handleMessage(inputMessage);
    }
}
...
}
}
...
}
}
```

# 避免出現程序無響應ANR(Keeping Your App Responsive)

編寫:kesenhoo - 原文:<http://developer.android.com/training/articles/perf-anr.html>

可能你寫的代碼在性能測試上表現良好，但是你的應用仍然有時候會反應遲緩(sluggish)，停頓(hang)或者長時間卡死(freeze)，或者是應用處理輸入的數據花費時間過長。對於你的應用來說最糟糕的事情是出現"程序無響應(Application Not Responding)" (ANR)的警示框。

在Android中，系統通過顯示ANR警示框來保護程序的長時間無響應。對話框如下：



此時，你的應用已經經歷過一段時間的無法響應了，因此系統提供用戶可以退出應用的選擇。為你的程序提供良好的響應性是至關重要的，這樣才能夠避免系統為用戶顯示ANR的警示框。

這節課描述了Android系統是如何判斷一個應用不可響應的。這節課還會提供程序編寫的指導原則，確保你的程序保持響應性。

## 是什麼導致了ANR?(What Triggers ANR?)

通常來說，系統會在程序無法響應用戶的輸入事件時顯示ANR。例如，如果一個程序在UI線程執行I/O操作(通常是網絡請求或者是文件讀寫)，這樣系統就無法處理用戶的輸入事件。或者是應用在UI線程花費了太多的時間用來建立一個複雜的在內存中的數據結構，又或者是在一個遊戲程序的UI線程中執行了一個複雜耗時的計算移動的操作。確保那些計算操作高效是很重要的，不過即使是最高效的代碼也是需要花時間執行的。

對於你的應用中任何可能長時間執行的操作，你都不應該執行在UI線程。你可以創建一個工作線程，把那些操作都執行在工作線程中。這確保了UI線程(這個線程會負責處理UI事件)能夠順利執行，也預防了系統因代碼僵死而崩潰。因為UI線程是和類級別相關聯的，你可以把相應性作為一個類級別(class-level)的問題(相比來說，代碼性能則屬於方法級別(method-level)的問題)

在Android中，程序的響應性是由Activity Manager與Window Manager系統服務來負責監控的。當系統監測到下面的條件之一時會顯示ANR的對話框:

- 對輸入事件(例如硬件點擊或者屏幕觸摸事件)，5秒內都無響應。
- BroadReceiver不能夠在10秒內結束接收到任務。

## 如何避免ANRs(How to Avoid ANRs)

Android程序通常是執行在默認的UI線程(也就是main線程)中的。這意味著在UI線程中執行的任何長時間的操作都可能觸發ANR，因為程序沒有給自己處理輸入事件或者broadcast事件的機會。

因此，任何執行在UI線程的方法都應該儘可能的簡短快速。特別是，在activity的生命週期的關鍵方法onCreate()與onResume()方法中應該儘可能的做比較少的事情。類似網絡或者DB操作等可能長時間執行的操作，或

者是類似調整bitmap大小等需要長時間計算的操作，都應該執行在工作線程中。(在DB操作中，可以通過異步的網絡請求)。

為了執行一個長時間的耗時操作而創建一個工作線程最方便高效的方式是使用 `AsyncTask`。只需要繼承`AsyncTask`並實現 `doInBackground()` 方法來執行任務即可。為了把任務執行的進度呈現給用戶，你可以執行 `publishProgress()` 方法，這個方法會觸發 `onProgressUpdate()` 的回調方法。在 `onProgressUpdate()` 的回調方法中(它執行在UI線程)，你可以執行通知用戶進度的操作，例如：

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    // Do the long-running work in here
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    // This is called each time you call publishProgress()
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    // This is called when doInBackground() is finished
    protected void onPostExecute(Long result) {
        showNotification("Downloaded " + result + " bytes");
    }
}
```

爲了能夠執行這個工作線程，只需要創建一個實例並執行 `execute()`：

```
new DownloadFilesTask().execute(url1, url2, url3);
```

相比起`AsycnTask`來說，創建自己的線程或者`HandlerThread`稍微複雜一點。如果你想這樣做，你應該通過 `Process.setThreadPriority()` 並傳遞 `THREAD_PRIORITY_BACKGROUND` 來設置線程的優先級爲"background"。如果你不通過這個方式來給線程設置一個低的優先級，那麼這個線程仍然會使得你的應用顯得卡頓，因爲這個線程默認與UI線程有着同樣的優先級。

如果你實現了`Thread`或者`HandlerThread`，請確保你的UI線程不會因爲等待工作線程的某個任務而去執行`Thread.wait()`或者`Thread.sleep()`。UI線程不應該去等待工作線程完成某個任務，你的UI線程應該提供一個`Handler`給其他工作線程，這樣工作線程能夠通過這個`Handler`在任務結束的時候通知UI線程。使用這樣的方式來設計你的應用程序可以使得你的程序UI線程保持響應性，以此來避免ANR。

`BroadcastReceiver`有特定執行時間的限制說明了`broadcast receivers`應該做的是：簡短快速的任務，避免執行費時的操作，例如保存數據或者註冊一個`Notification`。正如在UI線程中執行的方法一樣，程序應該避免在`broadcast receiver`中執行費時的長任務。但不是採用通過工作線程來執行複雜的任務的方式，你的程序應該啓動一個`IntentService`來響應`intent broadcast`的長時間任務。

Tip: 你可以使用`StrictMode`來幫助尋找因爲不小心加入到UI線程的潛在的長時間執行的操作，例如網絡或者DB相關的任務。

## 增加響應性(Reinforce Responsiveness)

通常來說，100ms - 200ms是用戶能夠察覺到卡頓的上限。這樣的話，下面有一些避免ANR的技巧：

- 如果你的程序需要響應正在後臺加載的任務，在你的UI中可以顯示ProgressBar來顯示進度。
- 對遊戲程序，在工作線程執行計算的任務。
- 如果你的程序在啓動階段有一個耗時的初始化操作，可以考慮顯示一個閃屏，要麼儘快的顯示主界面，然後馬上顯示一個加載的對話框，異步加載數據。無論哪種情況，你都應該顯示一個進度信息，以免用戶感覺程序有卡頓的情況。
- 使用性能測試工具，例如Systrace與Traceview來判斷程序中影響響應性的瓶頸。

# JNI Tips

---

編寫:[pedant](#) - 原文:<http://developer.android.com/training/articles/perf-jni.html>

JNI全稱Java Native Interface。它為託管代碼（使用Java編程語言編寫）與本地代碼（使用C/C++編寫）提供了一種交互方式。它是與廠商無關的（**vendor-neutral**），支持從動態共享庫中加載代碼，雖然這樣會稍顯麻煩，但有時這是相當有效的。

如果你對JNI還不是太熟悉，可以先通讀[Java Native Interface Specification](#)這篇文章來對JNI如何工作以及哪些特性可用有個大致的印象。這種接口的一些方面不能立即一讀就顯而易見，所以你會發現接下來的幾個章節很有用處。

## JavaVM 及 JNIEnv

---

JNI定義了兩種關鍵數據結構，“JavaVM”和“JNIEnv”。它們本質上都是指向函數表指針的指針（在C++版本中，它們被定義為類，該類包含一個指向函數表的指針，以及一系列可以通過這個函數表間接地訪問對應的JNI函數的成員函數）。JavaVM提供“調用接口（invocation interface）”函數，允許你創建和銷燬一個JavaVM。理論上你可以在一個進程中擁有多個JavaVM對象，但安卓只允許一個。

JNIEnv提供了大部分JNI功能。你定義的所有本地函數都會接收JNIEnv作為第一個參數。

JNIEnv是用作線程局部存儲。因此，你不能在線程間共享一個JNIEnv變量。如果在一段代碼中沒有其它辦法獲得它的JNIEnv，你可以共享JavaVM對象，使用GetEnv來取得該線程下的JNIEnv（如果該線程有一個JavaVM的話；見下面的AttachCurrentThread）。

JNIEnv和JavaVM的在C聲明是不同於在C++的聲明。頭文件“jni.h”根據它是以C還是以C++模式包含來提供不同的類型定義（typedefs）。因此，不建議把JNIEnv參數放到可能被兩種語言引入的頭文件中（換一句話說：如果你的頭文件需要#define \_\_cplusplus，你可能不得不在任何涉及到JNIEnv的內容處都要做些額外的工作）。

## 線程

---

所有的線程都是Linux線程，由內核統一調度。它們通常從託管代碼中啓動（使用Thread.start），但它們也能夠在其他任何地方創建，然後連接（attach）到JavaVM。例如，一個用pthread\_create啓動的線程能夠使用JNI AttachCurrentThread 或 AttachCurrentThreadAsDaemon函數連接到JavaVM。在一個線程成功連接（attach）之前，它沒有JNIEnv，不能夠調用JNI函數。

連接一個本地環境創建的線程會觸發構造一個java.lang.Thread對象，然後其被添加到主線程羣組（main ThreadGroup），以讓調試器可以探測到。對一個已經連接的線程使用AttachCurrentThread不做任何操作（no-op）。

安卓不能中止正在執行本地代碼的線程。如果正在進行垃圾回收，或者調試器已發出了中止請求，安卓會在下一次調用JNI函數的時候中止線程。

連接過的（attached）線程在它們退出之前必須通過JNI調用DetachCurrentThread。如果你覺得直接這樣編寫不太優雅，在安卓2.0（Eclair）及以上，你可以使用pthread\_key\_create來定義一個析構函數，它將會在線程退出時被調用，你可以在那兒調用DetachCurrentThread（使用生成的key與pthread\_setspecific將JNIEnv存儲到線程局部空間內；這樣JNIEnv能夠作為參數傳入到析構函數當中去）。

## jclass, jmethodID, jfieldID

---

如果你想在本地代碼中訪問一個對象的字段（field），你可以像下面這樣做：

- 對於類，使用`FindClass`獲得類對象的引用
- 對於字段，使用`GetFieldId`獲得字段ID
- 使用對應的方法（例如`GetIntField`）獲取字段下面的值

類似地，要調用一個方法，你首先得獲得一個類對象的引用，然後是方法ID（method ID）。這些ID通常是指向運行時內部數據結構。查找到它們需要些字符串比較，但一旦你實際去執行它們獲得字段或者做方法調用是非常快的。

如果性能是你看重的，那麼一旦查找出這些值之後在你的本地代碼中緩存這些結果是非常有用的。因為每個進程當中的JavaVM是存在限制的，存儲這些數據到本地靜態數據結構中是非常合理的。

類引用（class reference），字段ID（field ID）以及方法ID（method ID）在類被卸載前都是有效的。如果與一個類加載器（ClassLoader）相關的所有類都能夠被垃圾回收，但是這種情況在安卓上是罕見甚至不可能出現，只有這時類才被卸載。注意雖然`jclass`是一個類引用，但是必須要調用`NewGlobalRef`保護起來（見下個章節）。

當一個類被加載時如果你想緩存些ID，而後當這個類被卸載後再次載入時能夠自動地更新這些緩存ID，正確做法是在對應的類中添加一段像下面的代碼來初始化這些ID：

```
/*
 * 我們在一個類初始化時調用本地方法來緩存一些字段的偏移信息
 * 這個本地方法查找並緩存你感興趣的class/field/method ID
 * 失敗時拋出異常
 */
private static native void nativeInit();

static {
    nativeInit();
}
```

在你的C/C++代碼中創建一個`nativeClassInit`方法以完成ID查找的工作。當這個類被初始化時這段代碼將會執行一次。當這個類被卸載後而後再次載入時，這段代碼將會再次執行。

## 局部和全局引用

每個傳入本地方法的參數，以及大部分JNI函數返回的每個對象都是“局部引用”。這意味着它只在當前線程的當前方法執行期間有效。即使這個對象本身在本地方法返回之後仍然存在，這個引用也是無效的。

這同樣適用於所有 `jobject`的子類，包括`jclass`，`jstring`，以及`jarray`（當JNI擴展檢查是打開的時候，運行時會警告你對大部分對象引用的誤用）。

如果你想持有一個引用更長的時間，你就必須使用一個全局（“global”）引用了。`NewGlobalRef`函數以一個局部引用作為參數並且返回一個全局引用。全局引用能夠保證在你調用`DeleteGlobalRef`前都是有效的。

這種模式通常被用在緩存一個從`FindClass`返回的`jclass`對象的時候，例如：

```
jclass localClass = env->FindClass("MyClass");
jclass globalClass = reinterpret_cast<jclass>(env->NewGlobalRef(localClass));
```

所有的JNI方法都接收局部引用和全局引用作為參數。相同對象的引用卻可能具有不同的值。例如，用相同對象連續地調用`NewGlobalRef`得到返回值可能是不同的。為了檢查兩個引用是否指向的是同一個對象，你必須使用`IsSameObject`

函數。絕不要在本地代碼中用`= =`符號來比較兩個引用。

得出的結論就是你絕不要在本地代碼中假定對象的引用是常量或者是唯一的。代表一個對象的32位值從方法的一次調用到下一次調用可能有不同的值。在連續的調用過程中兩個不同的對象卻可能擁有相同的32位值。不要使用`jobject`的值作為key。

開發者需要“不過度分配”局部引用。在實際操作中這意味着如果你正在創建大量的局部引用，或許是通過對象數組，你應該使用`DeleteLocalRef`手動地釋放它們，而不是寄希望JNI來為你做這些。實現上只預留了16個局部引用的空間，所以如果你需要更多，要麼你刪掉以前的，要麼使用`EnsureLocalCapacity/PushLocalFrame`來預留更多。

注意`jfieldID`和`jmethodID`是映射類型（opaque types），不是對象引用，不應該被傳入到`NewGlobalRef`。原始數據指針，像`GetStringUTFChars`和`GetByteArrayElements`的返回值，也都不是對象（它們能夠在線程間傳遞，並且在調用對應的`Release`函數之前都是有效的）。

還有一種不常見的情況值得一提，如果你使用`AttachCurrentThread`連接（attach）了本地進程，正在運行的代碼在線程分離（detach）之前決不會自動釋放局部引用。你創建的任何局部引用必須手動刪除。通常，任何在循環中創建局部引用的本地代碼可能都需要做一些手動刪除。

## UTF-8、UTF-16 字符串

---

Java編程語言使用UTF-16格式。為了便利，JNI也提供了支持變形UTF-8（Modified UTF-8）的方法。這種變形編碼對於C代碼是非常有用的，因為它將\0000編碼成0xc0 0x80，而不是0x00。最愜意的事情是你能在具有C風格的以\0結束的字符串上計數，同時兼容標準的libc字符串函數。不好的一面是你不能傳入隨意的UTF-8數據到JNI函數而還指望它正常工作。

如果可能的話，直接操作UTF-16字符串通常更快些。安卓當前在調用`GetStringChars`時不需要拷貝，而`GetStringUTFChars`需要一次分配並且轉換為UTF-8格式。注意UTF-16字符串不是以零終止字符串，\0000是被允許的，所以你需要像對jchar指針一樣地處理字符串的長度。

不要忘記`Release`你`Get`的字符串。這些字符串函數返回`jchar`或者`jbyte`，都是指向基本數據類型的C格式的指針而不是局部引用。它們在`Release`調用之前都保證有效，這意味著當本地方法返回時它們並不主動釋放。

傳入`NewStringUTF`函數的數據必須是變形UTF-8格式。一種常見的錯誤情況是，從文件或者網絡流中讀取出的字符數據，沒有過濾直接使用`NewStringUTF`處理。除非你確定數據是7位的ASCII格式，否則你需要剔除超出7位ASCII編碼範圍（high-ASCII）的字符或者將它們轉換為對應的變形UTF-8格式。如果你沒那樣做，UTF-16的轉換結果可能不會是你想要的結果。JNI擴展檢查將會掃描字符串，然後警告你那些無效的數據，但是它們將不會發現所有潛在的風險。

## 原生類型數組

---

JNI提供了一系列函數來訪問數組對象中的內容。對象數組的訪問只能一次一條，但如果原生類型數組以C方式聲明，則能夠直接進行讀寫。

為了讓接口更有效率而不受VM實現的制約，`GetArrayElements`系列調用允許運行時返回一個指向實際元素的指針，或者是分配些內存然後拷貝一份。不論哪種方式，返回的原始指針在相應的`Release`調用之前都保證有效（這意味著，如果數據沒被拷貝，實際的數組對象將會受到牽制，不能重新成為整理堆空間的一部分）。你必須釋放（Release）每個你通過`Get`得到的數組。同時，如果`Get`調用失敗，你必須確保你的代碼在之後不會去嘗試調用`Release`來釋放一個空指針（NULL pointer）。

你可以用一個非空指針作為`isCopy`參數的值來決定數據是否會被拷貝。這相當有用。

Release類的函數接收一個mode參數，這個參數的值可選的有下面三種。而運行時具體執行的操作取決於它返回的指針是指向真實數據還是拷貝出來的那份。

- 0
  - 真實的：實際數組對象不受到牽制
  - 拷貝的：數據將會複製回去，備份空間將會被釋放。
- JNI\_COMMIT
  - 真實的：不做任何操作
  - 拷貝的：數據將會複製回去，備份空間將不會被釋放。
- JNI\_ABORT
  - 真實的：實際數組對象不受到牽制.之前的寫入不會被取消。
  - 拷貝的：備份空間將會被釋放；裏面所有的變更都會丟失。

檢查isCopy標識的一個原因是對一個數組做出變更後確認你是否需要傳入JNI\_COMMIT來調用Release函數。如果你交替地執行變更和讀取數組內容的代碼，你也許可以跳過無操作（no-op）的JNI\_COMMIT。檢查這個標識的另一個可能的原因是使用JNI\_ABORT可以更高效。例如，你也許想得到一個數組，適當地修改它，傳入部分到其他函數中，然後丟掉這些修改。如果你知道JNI是為你做了一份新的拷貝，就沒有必要再創建另一份“可編輯的（editable）”的拷貝了。如果JNI傳給你的是原始數組，這時你就需要創建一份你自己的拷貝了。

另一個常見的錯誤（在示例代碼中出現過）是認為當isCopy是false時你就可以不調用Release。實際上是沒有這種情況的。如果沒有分配備份空間，那麼初始的內存空間會受到牽制，位置不能被垃圾回收器移動。

另外注意JNI\_COMMIT標識沒有釋放數組，你最終需要使用一個不同的標識再次調用Release。

## 區間數組

當你想做的只是拷出或者拷進數據時，可以選擇調用像GetArrayElements和GetStringChars這類非常有用的函數。想想下面：

```
jbyte* data = env->GetByteArrayElements(array, NULL);
if (data != NULL) {
    memcpy(buffer, data, len);
    env->ReleaseByteArrayElements(array, data, JNI_ABORT);
}
```

這裏獲取到了數組，從當中拷貝出開頭的len個字節元素，然後釋放這個數組。根據代碼的實現，Get函數將會牽制或者拷貝數組的內容。上面的代碼拷貝了數據（為了可能的第二次），然後調用Release；這當中JNI\_ABORT確保不存在第三份拷貝了。

另一種更簡單的實現方式：

```
env->GetByteArrayRegion(array, 0, len, buffer);
```

這種方式有幾個優點：

- 只需要調用一個JNI函數而不是兩個，減少了開銷。
- 不需要指針或者額外的拷貝數據。
- 減少了開發人員犯錯的風險-在某些失敗之後忘記調用Release不存在風險。

類似地，你能使用SetArrayRegion函數拷貝數據到數組，使用GetStringRegion或者GetStringUTFRegion從String中拷貝字符。

## 異常

---

當異常發生時你一定不能調用大部分的JNI函數。你的代碼收到異常（通過函數的返回值，ExceptionCheck，或者ExceptionOccurred），然後返回，或者清除異常，處理掉。

當異常發生時你被允許調用的JNI函數有：

- DeleteGlobalRef
- DeleteLocalRef
- DeleteWeakGlobalRef
- ExceptionCheck
- ExceptionClear
- ExceptionDescribe
- ExceptionOccurred
- MonitorExit
- PopLocalFrame
- PushLocalFrame
- ReleaseArrayElements
- ReleasePrimitiveArrayCritical
- ReleaseStringChars
- ReleaseStringCritical
- ReleaseStringUTFChars

許多JNI調用能夠拋出異常，但通常提供一種簡單的方式來檢查失敗。例如，如果NewString返回一個非空值，你不需要檢查異常。然而，如果你調用一個方法（使用一個像CallObjectMethod的函數），你必須一直檢查異常，因為當一個異常拋出時它的返回值將不會是有效的。

注意中斷代碼拋出的異常不會展開本地調用堆棧信息，Android也還不支持C++異常。JNI Throw和ThrowNew指令僅僅是在當前線程中放入一個異常指針。從本地代碼返回到託管代碼時，異常將會被注意到，得到適當的處理。

本地代碼能夠通過調用ExceptionCheck或者ExceptionOccurred捕獲到異常，然後使用ExceptionClear清除掉。通常，拋棄異常而不處理會導致些問題。

沒有內建的函數來處理Throwable對象自身，因此如果你想得到異常字符串，你需要找出Throwable Class，然後查找到getMessage "()Ljava/lang/String;"的方法ID，調用它，如果結果非空，使用GetStringUTFChars，得到的結果你可以傳到printf(3) 或者其它相同功能的函數輸出。

## 擴展檢查

---

JNI的錯誤檢查很少。錯誤發生時通常會導致崩潰。Android也提供了一種模式，叫做CheckJNI，這當中JavaVM和JNIEnv函數表指針被換成了函數表，它在調用標準實現之前執行了一系列擴展檢查的。

額外的檢查包括：

- 數組：試圖分配一個長度為負的數組。
- 壞指針：傳入一個不完整jarray/jclass/jobject/jstring對象到JNI函數，或者調用JNI函數時使用空指針傳入到一個不能為空的參數中去。

- 類名：傳入了除 “java/lang/String” 之外的類名到JNI函數。
- 關鍵調用：在一個 “關鍵的(critical)” get和它對應的release之間做出JNI調用。
- 直接的ByteBuffers：傳入不正確的參數到NewDirectByteBuffer。
- 異常：當一個異常發生時調用了JNI函數。
- JNIEnv：在錯誤的線程中使用一個JNIEnv。
- jfieldIDs：使用一個空jfieldID，或者使用jfieldID設置了一個錯誤類型的值到字段（比如說，試圖將一個 StringBuilder賦給String類型的域），或者使用一個靜態字段下的jfieldID設置到一個實例的字段（instance field）反之亦然，或者使用的一個類的jfieldID卻來自另一個類的實例。
- jmethodIDs：當調用Call\*Method函數時時使用了類型錯誤的jmethodID：不正確的返回值，靜態/非靜態的不匹配，this的類型錯誤（對於非靜態調用）或者錯誤的類（對於靜態類調用）。
- 引用：在類型錯誤的引用上使用了DeleteGlobalRef/DeleteLocalRef。
- 釋放模式：調用release使用一個不正確的釋放模式（其它非 0，JNI\_ABORT，JNI\_COMMIT的值）。
- 類型安全：從你的本地代碼中返回了一個不兼容的類型（比如說，從一個聲明返回String的方法卻返回了 StringBuilder）。
- UTF-8：傳入一個無效的變形UTF-8字節序列到JNI調用。

（方法和域的可訪問性仍然沒有檢查：訪問限制對於本地代碼並不適用。）

有幾種方法去啓用CheckJNI。

如果你正在使用模擬器，CheckJNI默認是打開的。

如果你有一臺root過的設備，你可以使用下面的命令序列來重啓運行時（runtime），啓用CheckJNI。

```
adb shell stop  
adb shell setprop dalvik.vm.checkjni true  
adb shell start
```

隨便哪一種，當運行時（runtime）啓動時你將會在你的日誌輸出中見到如下的字符：

```
D AndroidRuntime: CheckJNI is ON
```

如果你有一臺常規的設備，你可以使用下面的命令：

```
adb shell setprop debug.checkjni 1
```

這將不會影響已經在運行的app，但是從那以後啓動的任何app都將打開CheckJNI（改變屬性為其它值或者只是重啓都將會再次關閉CheckJNI）。這種情況下，你將會在下一次app啓動時，在日誌輸出中看到如下字符：

```
D Late-enabling CheckJNI
```

## 本地庫

你可以使用標準的System.loadLibrary方法來從共享庫中加載本地代碼。在你的本地代碼中較好的做法是：

- 在一個靜態類初始化時調用System.loadLibrary（見之前的一個例子中，當中就使用了nativeClassInit）。參數是“未加修飾（undecorated）”的庫名稱，因此要加載“libfubar.so”，你需要傳入“fubar”。
- 提供一個本地函數：jint JNI\_OnLoad(JavaVM vm, void reserved)
- 在JNI\_OnLoad中，註冊所有你的本地方法。你應該聲明方法為“靜態的（static）”因此名稱不會佔據設備上符號表的空間。

JNI\_OnLoad函數在C++中的寫法如下：

```
jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env;
    if (vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6) != JNI_OK) {
        return -1;
    }

    // 使用env->FindClass得到 jclass
    // 使用env->RegisterNatives註冊本地方法

    return JNI_VERSION_1_6;
}
```

你也可以使用共享庫的全路徑來調用System.load。對於Android app，你也許會發現從context對象中得到應用私有數據存儲的全路徑是非常有用的。

上面是推薦的方式，但不是僅有的實現方式。顯式註冊不是必須的，提供一個JNI\_OnLoad函數也不是必須的。你可以使用基於特殊命名的“發現（discovery）”模式來註冊本地方法（更多細節見：[JNI spec](#)），雖然這並不可取。因為如果一個方法的簽名錯誤，在這個方法實際第一次被調用之前你是不會知道的。

關於JNI\_OnLoad另一點注意的是：任何你在JNI\_OnLoad中對FindClass的調用都發生在用作加載共享庫的類加載器的上下文（context）中。一般FindClass使用與“調用棧”頂部方法相關的加載器，如果當中沒有加載器（因為線程剛剛連接）則使用“系統（system）”類加載器。這就使得JNI\_OnLoad成為一個查尋及緩存類引用很便利的地方。

## 64位機問題

Android當前設計為運行在32位的平臺上。理論上它也能夠構建為64位的系統，但那不是現在的目標。當與本地代碼交互時，在大多數情況下這不是你需要擔心的，但是如果你打算存儲指針變量到對象的整型字段（integer field）這樣的本地結構中，這就變得非常重要了。為了支持使用64位指針的架構，你需要使用long類型而不是int類型的字段來存儲你的本地指針。

## 不支持的特性/向後兼容性

除了下面的例外，支持所有的JNI 1.6特性：

- DefineClass沒有實現。Android不使用Java字節碼或者class文件，因此傳入二進制class數據將不會有效。

對Android以前老版本的向後兼容性，你需要注意：

- 本地函數的動態查找 在Android 2.0(Eclair)之前，在搜索方法名稱時，字符“\$”不會轉換為對應的“\_00024”。要使它正常工作需要使用顯式註冊方式或者將本地方法的聲明移出內部類。
- 分離線程 在Android 2.0(Eclair)之前，使用pthread\_key\_create析構函數來避免“退出前線程必須分離”檢查是否可行的（運行時(runtime)也使用了一個pthread key析構函數，因此這是一場看誰先被調用的競賽）。
- 全局弱引用 在Android 2.0(Eclair)之前，全局弱引用沒有被實現。如果試圖使用它們，老版本將完全不兼容。你可

以使用Android平臺版本號常量來測試系統的支持性。在Android 4.0 (Ice Cream Sandwich)之前，全局弱引用只能傳給NewLocalRef, NewGlobalRef, 以及DeleteWeakGlobalRef（強烈建議開發者在使用全局弱引用之前都為它們創建強引用hard reference，所以這不應該在所有限制當中）。從Android 4.0 (Ice Cream Sandwich)起，全局弱引用能夠像其它任何JNI引用一樣使用了。

- 局部引用 在Android 4.0 (Ice Cream Sandwich)之前，局部引用實際上是直接指針。Ice Cream Sandwich為了更好地支持垃圾回收添加了間接指針，但這並不意味着很多JNI bug在老版本上不存在。更多細節見[JNI Local Reference Changes in ICS](#)。
- 使用GetObjectRefType獲得引用類型 在Android 4.0 (Ice Cream Sandwich)之前，使用直接指針（見上面）的後果就是正確地實現GetObjectRefType是不可能的。我們可以使用依次檢測全局弱引用表，參數，局部表，全局表的方式來代替。第一次匹配到你的直接指針時，就表明你的引用類型是當前正在檢測的類型。這意味着，例如，如果你在一個全局jclass上使用GetObjectRefType，而這個全局jclass碰巧與作為靜態本地方法的隱式參數傳入的jclass一樣的，你得到的結果是JNILocalRefType而不是JNIGlobalRefType。

## FAQ: 為什麼出現了UnsatisfiedLinkError?

當使用本地代碼開發時經常會見到像下面的錯誤：

```
java.lang.UnsatisfiedLinkError: Library foo not found
```

有時候這表示和它提示的一樣---未找到庫。但有些時候庫確實存在但不能被dlopen(3)找開，更多的失敗信息可以參見異常詳細說明。

你遇到“library not found”異常的常見原因可能有這些：

- 庫文件不存在或者不能被app訪問到。使用adb shell ls -l 檢查它的存在性和權限。
- 庫文件不是用NDK構建的。這就導致設備上並不存在它所依賴的函數或者庫。

另一種UnsatisfiedLinkError錯誤像下面這樣：

```
java.lang.UnsatisfiedLinkError: myfunc
    at Foo.myfunc(Native Method)
    at Foo.main(Foo.java:10)
```

在日誌中，你會發現：

```
w/dalvikvm( 880): No implementation found for native LFoo;.myfunc ()V
```

這意味着運行時嘗試匹配一個方法但是沒有成功，這種情況常見的原因有：

- 庫文件沒有得到加載。檢查日誌輸出中關於庫文件加載的信息。
- 由於名稱或者簽名錯誤，方法不能匹配成功。這通常是由於：
  - 對於方法的懶查尋，使用extern "C"和對應的可見性(JNIEXPORT)來聲明C++函數沒有成功。注意Ice Cream Sandwich之前的版本，JNIEXPORT宏是不正確的，因此對新版本的GCC使用舊的jni.h頭文件將不會有效。你可以使用arm-eabi-nm查看它們出現在庫文件裏的符號。如果它們看上去比較凌亂（像\_Z15Java\_Foo\_myfuncP7\_JNIEnvP7\_jclass這樣而不是Java\_Foo\_myfunc），或者符號類型是小寫的“t”而不是一個大寫的“T”，這時你就需要調整聲明瞭。

- 對於顯式註冊，在進行方法簽名時可能犯了些小錯誤。確保你傳入到註冊函數的簽名能夠完全匹配上日志文件裏提示的。記住“B”是byte，“Z”是boolean。在簽名中類名組件是以“L”開頭的，以“;”結束的，使用“/”來分隔包名/類名，使用“\$”符來分隔內部類名稱（比如說，Ljava/util/Map\$Entry;）。

使用javah來自動生成JNI頭文件也許能幫助你避免這些問題。

## FAQ: 為什麼FindClass不能找到我的類？

確保類名字符串有正確的格式。JNI類名稱以包名開始，然後使用左斜槓來分隔，比如java/lang/String。如果你正在查找一個數組類，你需要以對應數目的綰括號開頭，使用“L”和“;”將類名兩頭包起來，所以一個一維字符串數組應該寫成[Ljava/lang/String;。

如果類名稱看上去正確，你可能運行時遇到了類加載器的問題。FindClass想在與你代碼相關的類加載器中開始查找指定的類。檢查調用堆棧，可能看起像：

```
Foo.myfunc(Native Method)
Foo.main(Foo.java:10)
dalvik.system.NativeStart.main(Native Method)
```

最頂層的方法是Foo.myfunc。FindClass找到與類Foo相關的ClassLoader對象然後使用它。

這通常正是你所想的。如果你創建了自己的線程那麼就會遇到麻煩（也許是調用了pthread\_create然後使用AttachCurrentThread進行了連接）。現在跟蹤堆棧可能像下面這樣：

```
dalvik.system.NativeStart.run(Native Method)
```

最頂層的方法是NativeStart.run，它不是你應用內的方法。如果你從這個線程中調用FindClass，JavaVM將會啓動“系統（system）”的而不是與你應用相關的加載器，因此試圖查找應用內定義的類都將會失敗。

下面有幾種方法可以解決這個問題：

- 在JNI\_OnLoad中使用FindClass查尋一次，然後為後面的使用緩存這些類引用。任何在JNI\_OnLoad當中執行的FindClass調用都使用與執行System.loadLibrary的函數相關的類加載器（這個特例，讓庫的初始化更加方便了）。如果你的app代碼正在加載庫文件，FindClass將會使用正確的類加載器。
- 傳入類實例到一個需要它的函數，你的本地方法聲明必須帶有一個Class參數，然後傳入Foo.class。
- 在合適的地方緩存一個ClassLoader對象的引用，然後直接發起loadClass調用。這需要額外些工作。

## FAQ: 使用本地代碼怎樣共享原始數據？

也許你會遇到這樣一種情況，想從你的託管代碼或者本地代碼訪問一大塊原始數據的緩衝區。常見例子包括對bitmap或者聲音文件的處理。這裏有兩種基本實現方式。

你可以將數據存儲到byte[]。這允許你從託管代碼中快速地訪問。然而，在本地代碼端不能保證你不去拷貝一份就直接能夠訪問數據。在某些實現中，GetByteArrayElements和GetPrimitiveArrayCritical將會返回指向在維護堆中的原始數據的真實指針，但是在另外一些實現中將在本地堆空間分配一塊緩衝區然後拷貝數據過去。

還有一種選擇是將數據存儲在一塊直接字節緩衝區（direct byte buffer），可以使用

`java.nio.ByteBuffer.allocateDirect`或者`NewDirectByteBuffer` JNI函數創建buffer。不像常規的byte緩衝區，它的存儲空間將不會分配在程序維護的堆空間上，總是可以從本地代碼直接訪問（使用`GetDirectBufferAddress`得到地址）。依賴於直接字節緩衝區訪問的實現方式，從託管代碼訪問原始數據將會非常慢。

選擇使用哪種方式取決於兩個方面：

1.大部分的數據訪問是在Java代碼還是C/C++代碼中發生？

2.如果數據最終被傳到系統API，那它必須是怎樣的形式（例如，如果數據最終被傳到一個使用`byte[]`作為參數的函數，在直接的`ByteBuffer`中處理或許是不明智的）？

如果通過上面兩種情況仍然不能明確區分的，就使用直接字節緩衝區（`direct byte buffer`）形式。它們的支持是直接構建到JNI中的，在未來的版本中性能可能會得到提升。

# SMP(Symmetric Multi-Processor) Primer for Android

編寫:kesenhoo - 原文:<http://developer.android.com/training/articles/smp.html>

從Android 3.0開始，系統針對多核CPU架構的機器做了優化支持。這份文檔介紹了針對多核系統應該如何編寫C，C++以及Java程序。這裏只是作為Android應用開發者的入門教程，並不會深入討論這個話題，並且我們會把討論範圍集中在ARM架構的CPU上。

如果你並沒有時間學習整篇文章，你可以跳過前面的理論部分，直接查看實踐部分。但是我們並不建議這樣做。

## 0)簡要介紹

SMP的全稱是“Symmetric Multi-Processor”。它表示的是一種雙核或者多核CPU的設計架構。在幾年前，所有的Android設備都還是單核的。

大多數的Android設備已經有了多個CPU，但是通常來說，其中一個CPU負責執行程序，其他的CPU則處理設備硬件的相關事務（例如，音頻）。這些CPU可能有着不同的架構，運行在上面的程序無法在內存中彼此進行溝通交互。

目前大多數售賣的Android設備都是SMP架構的，這使得軟件開發者處理問題更加複雜。對於多線程的程序，如果多個線程執行在不同的內核上，這會使得程序更加容易發生race conditions。更糟糕的是，基於ARM架構的SMP比起x86架構來說，更加複雜，更難進行處理。那些在x86上測試通過的程序可能會在ARM上崩潰。

下面我們會介紹為何會這樣以及如何做才能夠使得你的代碼行為正常。

## 1)理論篇

這裏會快速並且簡單的介紹這個複雜的主題。其中一些部分並不完整，但是並沒有出現錯誤或者誤導。

查看文章末尾的進一步閱讀可以瞭解這個主題的更多知識。

### 1.1)內存一致性模型(Memory consistency models)

內存一致性模型(Memory consistency models)通常也被叫做“memory models”，描述了硬件架構如何確保內存訪問的一致性。例如，如果你對地址A進行了一個賦值，然後對地址B也進行了賦值，那麼內存一致性模型就需要確保每一個CPU都需要知道剛纔的操作賦值與操作順序。

這個模型通常被程序員稱為：順序一致性(sequential consistency)，請從文章末尾的進一步閱讀查看Adve & Gharachorloo這篇文章。

- 所有的內存操作每次只能執行一個。
- 所有的操作，在單核CPU上，都是順序執行的。

如果你關注一段代碼在內存中的讀寫操作，在sequentially-consistent的CPU架構上，是按照期待的順序執行的。It's possible that the CPU is actually reordering instructions and delaying reads and writes, but there is no way for code running on the device to tell that the CPU is doing anything other than execute instructions in a straightforward manner. (We're ignoring memory-mapped device driver I/O for the moment.)

To illustrate these points it's useful to consider small snippets of code, commonly referred to as litmus tests. These are assumed to execute in program order, that is, the order in which the instructions appear here is the

order in which the CPU will execute them. We don't want to consider instruction reordering performed by compilers just yet.

Here's a simple example, with code running on two threads:

Thread 1 Thread 2 A = 3 B = 5 reg0 = B reg1 = A

Thread 1	Thread 2
A = 3 B = 5	reg0 = B reg1 = A

In this and all future litmus examples, memory locations are represented by capital letters (A, B, C) and CPU registers start with "reg". All memory is initially zero. Instructions are executed from top to bottom. Here, thread 1 stores the value 3 at location A, and then the value 5 at location B. Thread 2 loads the value from location B into reg0, and then loads the value from location A into reg1. (Note that we're writing in one order and reading in another.)

Thread 1 and thread 2 are assumed to execute on different CPU cores. You should always make this assumption when thinking about multi-threaded code.

Sequential consistency guarantees that, after both threads have finished executing, the registers will be in one of the following states:

Registers	States
reg0=5, reg1=3	possible (thread 1 ran first)
reg0=0, reg1=0	possible (thread 2 ran first)
reg0=0, reg1=3	possible (concurrent execution)
reg0=5, reg1=0	never

To get into a situation where we see B=5 before we see the store to A, either the reads or the writes would have to happen out of order. On a sequentially-consistent machine, that can't happen.

Most uni-processors, including x86 and ARM, are sequentially consistent. Most SMP systems, including x86 and ARM, are not.

### 1.1.1)Processor consistency

### 1.1.2)CPU cache behavior

### 1.1.3)Observability

### 1.1.4)ARM's weak ordering

## 1.2)Data memory barriers

### 1.2.1)Store/store and load/load

### 1.2.2)Load/store and store/load

### 1.2.3)Barrier instructions

## 1.2.4)Address dependencies and causal consistency

## 1.2.5)Memory barrier summary

## 1.3)Atomic operations

### 1.3.1)Atomic essentials

### 1.3.2)Atomic + barrier pairing

### 1.3.3)Acquire and release

## 2)實踐篇

---

調試內存一致性(memory consistency)的問題非常困難。如果內存柵欄(memory barrier)導致一些代碼讀取到陳舊的數據，你將無法通過調試器檢查內存dumps文件來找出原因。By the time you can issue a debugger query, the CPU cores will have all observed the full set of accesses, and the contents of memory and the CPU registers will appear to be in an “impossible” state.

### 2.1)What not to do in C

#### 2.1.1)C/C++ and “volatile”

#### 2.1.2)Examples

### 2.2)在Java中不應該做的事

我們沒有討論過Java語言的一些相關特性，因此我們首先來簡要的看下那些特性。

#### 2.2.1)Java中的“synchronized”與“volatile”關鍵字

“synchronized”關鍵字提供了Java一種內置的鎖機制。每一個對象都有一個相對應的“monitor”，這個監聽器可以提供互斥的訪問。

“synchronized”代碼段的實現機制與自旋鎖(spin lock)有着相同的基礎結構：他們都是從獲取到CAS開始，以釋放CAS結束。這意味着編譯器(compilers)與代碼優化器(code optimizers)可以輕鬆的遷移代碼到“synchronized”代碼段中。一個實踐結果是：你不能判定synchronized代碼段是執行在這段代碼下面一部分的前面，還是這段代碼上面一部分的後面。更進一步，如果一個方法有兩個synchronized代碼段並且鎖住的是同一個對象，那麼在這兩個操作的中間代碼都無法被其他的線程所檢測到，編譯器可能會執行“鎖粗化lock coarsening”並且把這兩者綁定到同一個代碼塊上。

另外一個相關的關鍵字是“volatile”。在Java 1.4以及之前的文檔中是這樣定義的：volatile聲明和對應的C語言中的一樣可不靠。從Java 1.5開始，提供了更有力的保障，甚至和synchronization一樣具備強同步的機制。

volatile的訪問效果可以用下面這個例子來說明。如果線程1給volatile字段做了賦值操作，線程2緊接着讀取那個字段的值，那麼線程2是被確保能夠查看到之前線程1的任何寫操作。更通常的情況是，任何線程對那個字段的寫操作對於線程2來說都是可見的。實際上，寫volatile就像是釋放件監聽器，讀volatile就像是獲取監聽器。

非volatile的訪問有可能因為照顧volatile的訪問而需要做順序的調整。例如編譯器可能會往上移動一個非volatile加載操作，但是不會往下移動。Volatile之間的訪問不會因為彼此而做出順序的調整。虛擬機會注意處理如何的內存柵欄(memory barriers)。

當加載與保存大多數的基礎數據類型，他們都是原子的atomic，對於long以及double類型的數據則不具備原子型，除非他們被聲明為volatile。即使是在單核處理器上，併發多線程更新非volatile字段值也還是不確定的。

## 2.2.2) Examples

下面是一個錯誤實現的單步計數器(monotonic counter)的示例: ([Java theory and practice: Managing volatility](#)).

```
class Counter {  
    private int mValue;  
  
    public int get() {  
        return mValue;  
    }  
    public void incr() {  
        mValue++;  
    }  
}
```

假設get()與incr()方法是被多線程調用的。然後我們想確保當get()方法被調用時，每一個線程都能夠看到當前的數量。最引人注目的問題是mValue++實際上包含了下面三個操作。

1. reg = mValue
2. reg = reg + 1
3. mValue = reg

如果兩個線程同時在執行 incr() 方法，其中的一個更新操作會丟失。為了確保正確的執行 ++ 的操作，我們需要把 incr() 方法聲明為“synchronized”。這樣修改之後，這段代碼才能夠在單核多線程的環境中正確的執行。

然而，在SMP的系統下還是會執行失敗。不同的線程通過 get() 方法獲取到得值可能是不一樣的。因為我們是使用通常的加載方式來讀取這個值的。我們可以通過聲明 get() 方法為synchronized的方式來修正這個錯誤。通過這些修改，這樣的代碼纔是正確的了。

不幸的是，我們有介紹過有可能發生的鎖競爭(lock contention)，這有可能會傷害到程序的性能。除了聲明 get() 方法為synchronized之外，我們可以聲明 mValue 為“volatile”。(請注意 incr() 必須使用synchronize) 現在我們知道 volatile的mValue的寫操作對於後續的讀操作都是可見的。incr() 將會稍稍有點變慢，但是 get() 方法將會變得更加快速。因此讀操作多於寫操作時，這會是一個比較好的方案。(請參考AtomicInteger.)

下面是另外一個示例，和之前的C示例有點類似：

```
class MyGoodies {  
    public int x, y;  
}  
class MyClass {  
    static MyGoodies sGoodies;  
  
    void initGoodies() { // runs in thread 1  
        MyGoodies goods = new MyGoodies();  
        goods.x = 5;  
        goods.y = 10;  
        sGoodies = goods;  
    }  
  
    void useGoodies() { // runs in thread 2  
        if (sGoodies != null) {  
            int i = sGoodies.x; // could be 5 or 0  
            ...  
        }  
    }  
}
```

這段代碼同樣存在着問題，`sGoodies = goods` 的賦值操作有可能在 `goods` 成員變量賦值之前被察覺到。如果你使用 `volatile` 聲明 `sGoodies` 變量，你可以認為 `load` 操作為 `atomic_acquire_load()`，並且把 `store` 操作認為是 `atomic_release_store()`。

(請注意僅僅是 `sGoodies` 的引用本身為 `volatile`，訪問它的內部字段並不是這樣的。賦值語句 `z = sGoodies.x` 會執行一個 `volatile load` `MyClass.sGoodies` 的操作，其後會伴隨一個 `non-volatile` 的 `load` 操作：：`sGoodies.x`。如果你設置了一個本地引用 `MyGoodies localGoods = sGoodies`, `z = localGoods.x`，這將不會執行任何 `volatile loads`。)

另外一個在 Java 程序中更加常用的範式就是臭名昭著的“double-checked locking”：

```
class MyClass {
    private Helper helper = null;

    public Helper getHelper() {
        if (helper == null) {
            synchronized (this) {
                if (helper == null) {
                    helper = new Helper();
                }
            }
        }
        return helper;
    }
}
```

上面的寫法是為了獲得一個 `MyClass` 的單例。我們只需要創建一次這個實例，通過 `getHelper()` 這個方法。為了避免兩個線程會同時創建這個實例。我們需要對創建的操作加 `synchronize` 機制。然而，我們不想要為了每次執行這段代碼的時候都為“`synchronized`”付出額外的代價，因此我們僅僅在 `helper` 對象為空的時候加鎖。

在單核系統上，這是不能正常工作的。JIT 編譯器會破壞這件事情。請查看 [4\)Appendix](#) 的“‘Double Checked Locking is Broken’ Declaration”獲取更多的信息，或者是 Josh Bloch’s Effective Java 書中的 Item 71 (“Use lazy initialization judiciously”)。

在 SMP 系統上執行這段代碼，引入了一個額外的方式會導致失敗。把上面那段代碼換成 C 的語言實現如下：

```
if (helper == null) {
    // acquire monitor using spinlock
    while (atomic_acquire_cas(&this.lock, 0, 1) != success)
        ;
    if (helper == null) {
        newHelper = malloc(sizeof(Helper));
        newHelper->x = 5;
        newHelper->y = 10;
        helper = newHelper;
    }
    atomic_release_store(&this.lock, 0);
}
```

此時問題就更加明顯了：`helper` 的 `store` 操作發生在 `memory barrier` 之前，這意味着其他的線程能夠在 `store x/y` 之前觀察到非空的值。

你應該嘗試確保 `store helper` 執行在 `atomic_release_store()` 方法之後。通過重新排序代碼進行加鎖，但是這是無效的，因為往上移動的代碼，編譯器可以把它移動回原來的位置：在 `atomic_release_store()` 前面。（這裏沒有讀懂，下次再回讀）

有 2 個方法可以解決這個問題：

- 刪除外層的檢查。這確保了我們不會在synchronized代碼段之外做任何的檢查。
- 聲明helper為volatile。僅僅這樣一個小小的修改，在前面示例中的代碼就能夠在Java 1.5及其以後的版本中正常工作。

下面的示例演示了使用volatile的2各重要問題：

```
class MyClass {
    int data1, data2;
    volatile int vol1, vol2;

    void setValues() {    // runs in thread 1
        data1 = 1;
        vol1 = 2;
        data2 = 3;
    }

    void useValues1() {    // runs in thread 2
        if (vol1 == 2) {
            int l1 = data1;    // okay
            int l2 = data2;    // wrong
        }
    }

    void useValues2() {    // runs in thread 2
        int dummy = vol2;
        int l1 = data1;    // wrong
        int l2 = data2;    // wrong
    }
}
```

請注意 `useValues1()`，如果thread 2還沒有察覺到 `vol1` 的更新操作，那麼它也無法知道 `data1` 或者 `data2` 被設置的操作。一旦它觀察到了 `vol1` 的更新操作，那麼它也能夠知道`data1`的更新操作。然而，對於 `data2` 則無法做任何猜測，因為store操作是在volatile store之後發生的。

`useValues2()` 使用了第2個volatile字段：`vol2`，這會強制VM生成一個memory barrier。這通常不會發生。為了建立一個恰當的“happens-before”關係，2個線程都需要使用同一個volatile字段。在thread 1中你需要知道`vol2`是在`data1/data2`之後被設置的。(The fact that this doesn't work is probably obvious from looking at the code; the caution here is against trying to cleverly “cause” a memory barrier instead of creating an ordered series of accesses.)

## 2.3)What to do

### 2.3.1)General advice

在C/C++中，使用 `pthread` 操作，例如mutexes與semaphores。他們會使用合適的memory barriers，在所有的Android平臺上提供正確有效的行為。請確保正確這些技術，例如在沒有獲得對應的mutex的情況下賦值操作需要很謹慎。

避免直接使用atomic方法。如果locking與unlocking之間沒有競爭，locking與unlocking一個pthread mutex 分別需要一個單獨的atomic操作。如果你需要一個lock-free的設計，你必須在開始寫代碼之前瞭解整篇文檔的要點。（或者是尋找一個已經為SMP ARM設計好的庫文件）。

Be extremely circumspect with "volatile" in C/C++. It often indicates a concurrency problem waiting to happen.

In Java, the best answer is usually to use an appropriate utility class from the `java.util.concurrent` package. The code is well written and well tested on SMP.

Perhaps the safest thing you can do is make your class immutable. Objects from classes like String and Integer

hold data that cannot be changed once the class is created, avoiding all synchronization issues. The book Effective Java, 2nd Ed. has specific instructions in “Item 15: Minimize Mutability”. Note in particular the importance of declaring fields “final” (Bloch).

If neither of these options is viable, the Java “synchronized” statement should be used to guard any field that can be accessed by more than one thread. If mutexes won’t work for your situation, you should declare shared fields “volatile”, but you must take great care to understand the interactions between threads. The volatile declaration won’t save you from common concurrent programming mistakes, but it will help you avoid the mysterious failures associated with optimizing compilers and SMP mishaps.

The Java Memory Model guarantees that assignments to final fields are visible to all threads once the constructor has finished — this is what ensures proper synchronization of fields in immutable classes. This guarantee does not hold if a partially-constructed object is allowed to become visible to other threads. It is necessary to follow safe construction practices.(Safe Construction Techniques in Java).

### 2.3.2)Synchronization primitive guarantees

The pthread library and VM make a couple of useful guarantees: all accesses previously performed by a thread that creates a new thread are observable by that new thread as soon as it starts, and all accesses performed by a thread that is exiting are observable when a join() on that thread returns. This means you don’t need any additional synchronization when preparing data for a new thread or examining the results of a joined thread.

Whether or not these guarantees apply to interactions with pooled threads depends on the thread pool implementation.

In C/C++, the pthread library guarantees that any accesses made by a thread before it unlocks a mutex will be observable by another thread after it locks that same mutex. It also guarantees that any accesses made before calling signal() or broadcast() on a condition variable will be observable by the woken thread.

Java language threads and monitors make similar guarantees for the comparable operations.

### 2.3.3)Upcoming changes to C/C++

The C and C++ language standards are evolving to include a sophisticated collection of atomic operations. A full matrix of calls for common data types is defined, with selectable memory barrier semantics (choose from relaxed, consume, acquire, release, acq\_rel, seq\_cst).

See the Further Reading section for pointers to the specifications.

## 3)Closing Notes

---

While this document does more than merely scratch the surface, it doesn’t manage more than a shallow gouge. This is a very broad and deep topic. Some areas for further exploration:

- Learn the definitions of happens-before, synchronizes-with, and other essential concepts from the Java Memory Model. (It’s hard to understand what “volatile” really means without getting into this.)
- Explore what compilers are and aren’t allowed to do when reordering code. (The JSR-133 spec has some great examples of legal transformations that lead to unexpected results.)
- Find out how to write immutable classes in Java and C++. (There’s more to it than just “don’t change anything after construction”.)
- Internalize the recommendations in the Concurrency section of Effective Java, 2nd Edition. (For example,

- you should avoid calling methods that are meant to be overridden while inside a synchronized block.)
- Understand what sorts of barriers you can use on x86 and ARM. (And other CPUs for that matter, for example Itanium's acquire/release instruction modifiers.)
  - Read through the `java.util.concurrent` and `java.util.concurrent.atomic` APIs to see what's available.
  - Consider using concurrency annotations like `@ThreadSafe` and `@GuardedBy` (from `net.jcip.annotations`).

The Further Reading section in the appendix has links to documents and web sites that will better illuminate these topics.

## 4)Appendix

---

### 4.1)SMP failure example

### 4.2)Implementing synchronization stores

### 4.3)Further reading

# 保護安全與隱私的最佳策略

---

編寫:craftsmanBai - <http://z1ng.net> - 原文:<http://developer.android.com/training/best-security.html>

下面的課程教你如何確保應用程序數據的安全。

## 安全要點

---

怎樣執行多個任務的同時確保應用程序數據和用戶數據的安全。

## HTTPS和SSL的安全

---

如何確保應用程序在進行網絡傳輸時是安全的。

## 更新你的Security Provider對抗SSL漏洞攻擊

---

如何使用和更新Google Play services security provider來對抗SSL漏洞攻擊。

## 企業級開發

---

如何為企業級應用程序實施設備管理策略。

# 安全要點

編寫:[craftsmanBai - http://z1ng.net](http://z1ng.net) - 原文:<http://developer.android.com/training/articles/security-tips.html>

Android的安全性體現在系統顯著地減少了應用程序安全問題帶來的影響。你可以在默認的系統設置和文件權限設置的環境下建立應用，避免了一堆頭疼的安全問題。

幫助你建立應用的部分核心安全特性如下：

- Android應用程序沙盒，將你的應用數據和代碼同其他程序隔開。
- 具有魯棒性實現了常見安全功能的應用框架，例如密碼學應用，權限控制，安全IPC
- 使用ASLR, NX, ProPolice, safe\_iop, OpenBSD dlmalloc, OpenBSD calloc, Linux mmap\_min\_addr等技術，減少了常見內存管理錯誤。
- 加密文件系統可以保護丟失或被盜走的設備數據。
- 用戶權限控制限制訪問系統詳細情況和用戶數據。
- 應用程序權限以單個應用為基礎控制其數據。

儘管如此，熟悉Android安全特性仍然很重要。遵守這些習慣將其作為優秀的代碼風格，能夠減少無意間給用戶帶來的安全問題。

## 數據存儲

對於一個Android的應用程序來說，最為常見的安全問題是存放在設備上的數據能否被其他應用獲取。在設備上存放數據基本方式有三種：

### 使用內存儲器

默認情況下，你在[內存儲器](#)中創建的文件只有你的應用可以訪問。這種機制被Android加強了並且對於大多數應用程序都是有效的。你應該避免在IPC文件中使用[MODE\\_WORLD\\_WRITEABLE](#)或者[MODE\\_WORLD\\_READABLE](#)模式，因為它們不為特殊程序提供限制數據訪問的功能，它們也不對數據格式進行任何控制。如果你想與其他應用的進程共享數據，可以使用[content provider](#)，它給其他應用提供了可讀寫權限以及逐項動態獲取權限。

如果想對敏感數據進行特別保護，你可以使用應用程序無法直接獲取的密鑰來加密本地文件。例如，密鑰可以存放在[密鑰庫](#)而非設備上，使用用戶密碼進行保護。儘管這種方式無法在具有root權限時監視用戶輸入的密碼的情況下保護數據，但是它可以為未進行[文件系統加密](#)已丟失的設備提供保護。

### 使用外部存儲器

創建於[外部存儲](#)的文件，比如SD卡，是全局可讀寫的。由於外部存儲器可被用戶移除並且能夠被任何應用修改，因此不應使用外部存儲存儲應用的敏感信息。當處理來自外部存儲器的數據時，應用程序應該執行輸入驗證（參看輸入驗證章節）我們強烈建議應用在動態加載之前不要把可執行文件或class文件存儲到外部存儲器中。如果一個應用從外部存儲器檢索可執行文件，那麼在動態加載之前它們應該進行簽名與加密驗證。

### 使用Content Providers

[ContentProviders](#)提供一個結構存儲機制，可以限制你自己的應用或者導出給其他應用程序允許訪問。如果你不打算為其他應用提供訪問你的[ContentProvider](#)功能，那麼在manifest中標記他們為[android:exported=false](#)即可。要建立

一個給其他應用使用而導出的Content Provider，你可以為讀寫操作指定一個單一的permission，或者在manifest中為讀寫操作指定確切的許可。我們強烈建議你限制權限給手頭要求完成的任務。記住，通常顯示新功能稍後加入許可比把許可撤走並打斷已經存在的用戶更容易。

如果你使用Content Provider僅在自己的應用中共享數據，使用簽名級別android:protectionLevel的許可是更可取的。簽名許可不需要用戶確認，當應用使用同樣的密鑰獲取數據時，這提供了更好的用戶體驗，也更好地控制了Content Provider數據的訪問。Content Providers也可以通過聲明android:grantUriPermissions並在觸發組件的Intent對象中使用FLAG\_GRANT\_READ\_URI\_PERMISSION和FLAG\_GRANT\_WRITE\_URI\_PERMISSION標誌提供更細緻的訪問。這些許可的作用域可以通過grant-uri-permission進一步限制。當訪問一個ContentProvider時，使用參數化的查詢方法，比如query(), update()和delete()來避免來自不信任源潛在的SQL注入。注意，如果提交方法之前的selection是通過連接用戶數據建立的，使用參數化的方法是不夠的。不要對“寫”權限有一個錯誤的觀念。考慮“寫”權限允許sql語句，使得部分數據使用創造性的WHERE語句並且解析結果變為可能。例如：入侵者可能在通話記錄中通過修改一條記錄來檢測某個特定存在的電話號碼，只要那個電話號碼已經存在。如果content provider數據有可預見的結構，提供“寫”權限也許等同於同時提供了“讀寫”權限。

## 使用許可

---

因為安卓沙盒將應用程序隔開，程序必須顯式地共享資源和數據。它們通過聲明他們需要的權限來獲取額外的功能，而基本的沙盒不提供這些功能，比如相機訪問設備。

### 請求許可

我們建議最小化應用請求的許可數量，不具有訪問敏感資料的權限可以減少無意中濫用這些權限的風險，可以增加用戶接受度，並且減少應用可被攻擊者攻擊利用。

如果你的應用可以設計成不需要任何許可，那最好不過。例如：與其請求訪問設備信息來建立一個標識，不如建立一個GUID（這個例子在Handling User Data中有說明）。

除了請求許可之外，你的應用可以使用permissions來保護可能會暴露給其他應用的安全敏感的IPC：比如ContentProvider。通常來說，我們建議使用訪問控制而不是用戶權限確認許可，因為許可會使用戶感到困惑。例如，考慮在權限設置上為應用間的IPC通信使用單一開發者提供的簽名保護級別

不要泄漏受許可保護的數據。只有當應用通過IPC暴露數據纔會發生這種情況，因為它具有特殊權限，卻不要求任何客戶端的IPC接口有那樣的權限。更多細節帶來的潛在影響以及這種問題發生的頻率在USENIX: [http://www.cs.berkeley.edu/~afelt/felt\\_usenixsec2011.pdf](http://www.cs.berkeley.edu/~afelt/felt_usenixsec2011.pdf)研究論文中都有說明。

### 創建許可

通常，你應該力求建立擁有儘量少許可的應用，直至滿足你的安全需要。建立一個新的許可對於大多數應用相對少見，因為系統定義的許可覆蓋很多情況。在適當的地方使用已經存在的許可執行訪問檢查。

如果必須建立一個新的許可，考慮能否使用signature protection level來完成你的任務。簽名許可對用戶是透明的並且只允許相同開發者簽名的應用訪問，與應用執行許可檢查一樣。如果你建立一個dangerous protection level，那麼用戶需要決定是否安裝這個應用。這會使其他開發者困惑，也使用戶困惑。

如果你要建立一個危險的許可，則會有多種複雜情況需考慮：

- 對於用戶將要做出的安全決定，許可需要用字符串對其進行簡短的表述。
- 許可字符串必須保證語言的國際化。
- 用戶可能對一個許可感到困惑或者知曉風險而選擇不安裝應用
- 當許可的創造未安裝的時候，應用可能要求許可。

上面每一個因素都為應用開發者帶來了重要的非技術挑戰，同時也使用戶感到困惑，這也是我們不建議使用危險許可的原因。

## 使用網絡

---

網絡交易具有很高的安全風險，因為它涉及到傳送私人的數據。人們對移動設備的隱私關注日益加深，特別是當設備進行網絡交易時，因此應用採取最佳方式保護用戶數據安全極為重要。

### 使用IP網絡

android下的網絡與Linux環境下的差別並不很大。主要考慮的是確保對敏感數據採用了適當的協議，比如使用[HTTPS進行網絡傳輸](#)。我們在任何支持HTTPS的服務器上更願意使用HTTPS而不是HTTP，因為移動設備頻繁連接不安全的網絡，比如公共WiFi熱點。

認證加密socket級別的通信可通過使用[SSL Socket](#)類輕鬆實現。由於Android設備使用WiFi連接不安全網絡的頻率，對於所有應用來說，使用安全網絡是極力鼓勵支持的。

我們發現部分應用使用[localhost](#)端口處理敏感的IPC。不鼓勵這種方法是因為這些接口可被設備上的其他應用訪問。相反，你應該在可認證的地方使用android IPC機制，例如[Service](#)（比使用迴環還糟的是綁定INADDR\_ANY，因為你的應用可能收到來自任何地方來的請求，我們也已經見識過了）。

一個有必要重複的常見議題是，確保不信任從HTTP或者其他不安全協議下載的數據。這包括在[WebView](#)中的輸入驗證和對於http的任何響應。

### 使用電話網絡

SMS協議是Android開發者使用最頻繁的電話協議，主要為用戶與用戶之間的通信設計，但對於想要傳送數據的應用來說並不合適。由於SMS的限制性，我們強烈建議使用[Google Cloud Messaging \(GCM\)](#)和IP網絡從web服務器發送數據消息給用戶設備應用。

很多開發者沒有意識到SMS在網絡上或者設備上是不加密的，也沒有牢固驗證。特別是任何SMS接收者應該預料到惡意用戶也許已經給你的應用發送了SMS：不要指望未驗證的SMS數據執行敏感操作。你也應該注意到SMS在網絡上也許會遭到冒名頂替並且/或者攔截，對於Android設備本身，SMS消息是通過廣播intent傳遞的，所以他們也許會被其他擁有[READ\\_SMS](#)許可的應用截獲。

## 輸入驗證

---

無論應用運行在什麼平臺上，功能不完善的輸入驗證是最常見的影響應用安全問題之一。Android有平臺級別的對策，用於減少應用的公開輸入驗證問題，你應該在可能的地方使用這些功能。同樣需要注意的是，選擇類型安全的語言能減少輸入驗證問題。

如果你使用native代碼，那麼任何從文件讀取的，通過網絡接收的，或者通過IPC接收的數據都有可能引發安全問題。最常見的問題是[buffer overflows](#), [use after free](#), 和[off-by-one](#)。Android提供安全機制比如ASLR和DEP以減少這些漏洞的可利用性，但是沒有解決基本的問題。小心處理指針和管理緩存可以預防這些問題。

動態、基於字符串的語言，比如JavaScript和SQL，都常受到由轉義字符和[腳本注入](#)帶來的輸入驗證問題。

如果你使用提交到SQL Database或者Content Provider的數據，SQL注入也許是個問題。最好的防禦是使用參數化的查詢，就像ContentProviders中討論的那樣。限制權限為只讀或者只寫可以減少SQL注入的潛在危害。

如果你不能使用上面提到的安全功能，我們強烈建議使用結構嚴謹的數據格式並且驗證符合期望的格式。黑名單策略與

替換危險字符是有效的，但這些技術在實踐中是易錯的並且當錯誤可能發生的時候應該儘量避免。

## 處理用戶數據

---

通常來說，處理用戶數據安全最好的方法是最小化獲取敏感數據用戶個人數據的API使用。如果你對數據進行訪問並且可以避免存儲或傳輸，那就不要存儲和傳輸數據。最後，思考是否有一種應用邏輯可能被實現為使用hash或者不可逆形式的數據。例如，你的應用也許使用一個email地址的hash作為主鍵，避免傳輸或存儲email地址，這減少無意間泄漏數據的機會，並且也能減少攻擊者嘗試利用你的應用的機會。

如果你的應用訪問私人數據，比如密碼或者用戶名，記住司法也許要求你提供一個使用和存儲這些數據的隱私策略的解釋。所以遵守最小化訪問用戶數據最佳的安全實踐也許只是簡單的服從。

你也應該考慮你應用是否會疏忽暴露個人信息給其他方，比如廣告第三方組件或者你應用使用的第三方服務。如果你不知道為什麼一個組件或者服務請求個人信息，那麼就不要提供給它。通常來說，通過減少應用訪問個人信息，會減少這個區域潛在的問題。

如果必須訪問敏感數據，評估這個信息是否必須要傳到服務器，或者是否可以被客戶端操作。考慮客戶端上使用敏感數據運行的任何代碼，避免傳輸用戶數據 確保不會無意間通過過渡自由的IPC、world writable文件、或網絡socket暴露用戶數據給其他設備上的應用。這裏有一個泄漏權限保護數據的特別例子，在[Requesting Permissions](#)章節中討論。

如果需要GUID，建立一個大的、唯一的數字並保存它。不要使用電話標識，比如與個人信息相關的電話號碼或者IMEI。這個話題在[Android Developer Blog](#)中有更詳細的討論。

應用開發者應謹慎的把log寫到機器上。在Android中，log是共享資源，一個帶有[READ\\_LOGS](#)許可的應用可以訪問。即使電話log數據是臨時的並且在重啓之後會擦除，不恰當地記錄用戶信息會無意間泄漏用戶數據給其他應用。

## 使用WebView

---

因為[WebView](#)能包含HTML和JavaScript瀏覽網絡內容，不恰當的使用會引入常見的web安全問題，比如[跨站腳本攻擊](#)（JavaScript注入）。Android採取一些機制通過限制WebView的能力到應用請求功能最小化來減少這些潛在的問題。

如果你的應用沒有在WebView內直接使用JavaScript，不要調用[setJavaScriptEnabled\(\)](#)。某些樣本代碼使用這種方法，可能會導致在產品應用中改變用途：所以如果不需要的話移除它。默認情況下WebView不執行JavaScript，所以跨站腳本攻擊不會產生。

使用[addJavaScriptInterface\(\)](#)要特別的小心，因為它允許JavaScript執行通常保留給Android應用的操作。只把[addJavaScriptInterface\(\)](#)暴露給可靠的輸入源。如果不受信任的輸入是被允許的，不受信任的JavaScript也許會執行Android方法。總得來說，我們建議只把[addJavaScriptInterface\(\)](#)暴露給你應用內包含的JavaScript。

如果你的應用通過WebView訪問敏感數據，你也許想要使用[clearCache\(\)](#)方法來刪除任何存儲到本地的文件。服務端的header，比如no-cache，能用於指示應用不應該緩存特定的內容。

## 處理證書

通常來說，我們建議請求用戶證書頻率最小化--使得釣魚攻擊更明顯，並且降低其成功的可能。取而代之使用授權令牌然後刷新它。

可能的情況下，用戶名和密碼不應該存儲到設備上，而使用用戶提供的用戶名和密碼執行初始認證，然後使用一個短暫的、特定服務的授權令牌。可以被多個應用訪問的service應該使用[AccountManager](#)訪問。如果可能的話，使用AccountManager類來執行基於雲的服務並且不把密碼存儲到設備上。

使用AccountManager獲取Account之後，進入任何證書前檢查CREATOR，這樣你就不會因為疏忽而把證書傳遞給錯誤的應用。

如果證書只是用於你創建的應用，那麼你能使用checkSignature()驗證訪問AccountManager的應用。或者，如果一個應用要使用證書，你可以使用KeyStore來儲存。

## 使用密碼學

---

除了採取數據隔離之外，支持完整的文件系統加密，並且提供安全交流通道。Android提供大量加密算法來保護數據。

通常來說，嘗試使用最高級別的已存在framework的實現來支持，如果你需要安全的從一個已知的位置收回一個文件，一個簡單的HTTPS URI也許就足夠了，並且這部分不要求任何加密知識。如果你需要一個安全隧道，考慮使用HttpsURLConnection或者SSLocket要比使用你自己的協議好。

如果你發現你的確需要實現你自己的協議，我們強烈建議你不要自己實現加密算法。使用已經存在的加密算法，比如Cipher類中提供的AES或者RSA。

使用一個安全的隨機數生成器（SecureRandom）來初始化加密的key（KeyGenerator）。使用一個不安全隨機數生成器生成的key嚴重削弱算法的優點，而且可能遭到離線攻擊。

如果你需要存儲一個key來重複使用，使用類似於KeyStore的機制，提供長期儲存和檢索加密key的功能。

## 使用進程間通信

---

一些Android應用試圖使用傳統的Linux技術實現IPC，比如網絡socket和共享文件。我們強烈鼓勵使用Android系統IPC功能，比如Intent，Binder，Messenger和BroadcastReceiver。Android IPC機制允許你為每一個IPC機制驗證連接到你的IPC和設置安全策略的應用的身份。

很多安全元素通過IPC機制共享。Broadcast Receiver, Activity, 和Service都在應用的manifest中聲明。如果你的IPC機制不打算給其他應用使用，設置 android:exported 屬性為false。這對於同一個UID內包含多個進程的應用，或者在開發後期決定不想通過IPC暴露功能並且不想重寫代碼的時候非常有用。

如果你的IPC打算讓別的應用訪問，你可以通過使用Permission標記設置一個安全策略。如果IPC是使用同一個密鑰簽名的獨立的應用間的，使用signature更好一些。

## 使用意圖

Intent是Android中異步IPC機制的首選。根據你應用的需求，你也許使用sendBroadcast(),sendOrderedBroadcast()或者直接的intent來指定一個應用組件。

注意，有序廣播可以被接收者“消費”，所以他們也許不會被發送到所有的應用中。如果你要發送一個intent給指定的receiver，這個intent必須被直接的發送給這個receiver。

intent的發送者能在發送的時候驗證接受者是否有一個許可指定了一个non-Null Permission。只有有那個許可的應用纔會收到這個intent。如果廣播intent內的數據是敏感的，你應該考慮使用許可來保證惡意應用沒有恰當的許可無法註冊接收那些消息。這種情況下，可以考慮直接執行這個receiver而不是發起一個廣播。

注意：intent過濾器不能作為安全特性--組件可被intent顯式調用，可能會沒有符合intent過濾器的數據。你應該在intent接收器內執行輸入驗證，確認對於調用接收者，服務、或活動來說格式正確合理。

## 使用服務

[Service](#)經常被用於為其他應用提供服務。每個service類必須在它的manifest文件進行相應的聲明。

默認情況下，Service不能被導出和被其他應用執行。如果你加入了任何intent過濾器到服務蛇呢光明中，那麼它默認為可以被導出。最好明確聲明[android:exported](#)元素來確定它按照你設想的運行。可以使用[android:permission](#)保護Service。這樣做，其他應用在他們自己的manifest文件中將需要聲明相應的元素來啓動、停止或者綁定到這個service上。

一個Service可以使用許可保護單獨的IPC調用，在執行調用前通過調用[checkCallingPermission\(\)](#)來實現。我們建議使用manifest中聲明的許可，因為那些是不容易監管的。

## 使用binder和messenger接口

在Android中，[Binders](#)和[Messenger](#)是RPC-style IPC的首選機制。必要的話，他們提供一個定義明確的接口，促進彼此的端點認證。

我們強烈鼓勵在一定程度上設計不要求指定許可檢查的接口。Binder和[Messenger](#)不在應用的manifest中聲明，因此你不能直接在Binder上應用聲明的許可。它們在應用的manifest中繼承許可聲明，[Service](#)或者[Activity](#)內實現了許可。如果你打算創建一個接口，在一個指定binder接口上要求認證和/或者訪問控制，這些控制必須在Binder和[Messenger](#)的接口中明確添加代碼。

如果提供一個需要訪問控制的接口，使用[checkCallingPermission\(\)](#)來驗證調用者是否擁有必要許可。由於你的應用的id已經被傳遞到別的接口，因此代表調用者訪問一個Service之前這尤其重要。如果調用一個Service提供的接口，如果你沒有對給定的Service訪問許可，[bindService\(\)](#)請求也許會失敗。如果調用你自己的應用提供的本地接口，使用[clearCallingIdentity\(\)](#)來進行內部安全檢查是有用的。

更多關於用服務運行IPC的信息，參見[Bound Services](#)

## 利用廣播接收機

[Broadcast receivers](#)是用來處理通過[intent](#)發起的異步請求。

默認情況下，receiver是導出的，並且可以被任何其他應用執行。如果你的[BroadcastReceiver](#)打算讓其他應用使用，你也許想在應用的manifest文件中使用元素對receiver使用安全許可。這將阻止沒有恰當許可的應用發送intent給這個[BroadcastReceiver](#)。

## 動態加載代碼

我們極為不鼓勵從應用文件外加載代碼。由於代碼注入或者代碼篡改這樣做顯著增加了應用暴露的可能，同事也增加了版本管理和應用測試的複雜性。最終可能造成無法驗證應用的行為，因此在某些環境下應該被限制。

如果你的應用確實動態加載了代碼，最重要的事情是記住運行動態加載的代碼與應用具有相同的安全許可。用戶決定安裝你的應用是基於你的id，他們期望你提供任何運行在應用內部的代碼，包括動態加載的代碼。

動態加載代碼主要的風險在於代碼來源於可確認的源頭。如果這個模塊是之間直接包含在你的應用中，那麼它們不能被其他應用修改，不論代碼是本地庫或者是使用[DexClassLoader](#)加載的類這都是事實。我們見過很多應用實例嘗試從不安全的地方加載代碼，比如從網絡上通過非加密的協議或者從world writable位置（比如外部存儲）下載數據。這些地方會允許網絡上其他人在傳輸過程中修改其內容，或者允許用戶設備上的其他應用修改其內容。

## 在虛擬機器安全性

Dalvik是安卓的運行時虛擬機(VM).Dalvik是特別為安卓建立的，但許多其他虛擬機相關的安全代碼的也適用於安卓。一

般來說，你不應該關心與自己有關的虛擬機的安全問題。你的應用程序在一個安全的沙盒環境下運行，所以系統上的其他進程無法訪問你的代碼或私人數據。

如果你想更深入瞭解虛擬機的安全問題，我們建議您熟悉一些現有文獻的主題。推薦兩個比較流行的資源：

- <http://www.securingjava.com/toc.html>
- [https://www.owasp.org/index.php/Java\\_Security\\_Resources](https://www.owasp.org/index.php/Java_Security_Resources)

這個文檔集中於安卓與其他VM環境不同地方。對於有在其他環境下有VM編程經驗開發者來說，這裏有兩個普遍的問題可能對於編寫Android應用來說有些不同：

- 一些虛擬機，比如JVM或者.net，擔任一個安全的邊界作用，代碼與底層操作系統隔離。在Android上，Dalvik VM不是一個安全邊界：應用沙箱是在系統級別實現的，所以Dalvik可以在同一個應用與native代碼相互操作，沒有任何安全約束。
- 已知的手機上的存儲限制，對來發者來說，想要建立模塊化應用和使用動態類加載是很常見的。要這麼做的時候需要考慮兩個資源：一是在哪裏恢復你的應用邏輯，二是在哪裏存儲它們。不要從未驗證的資源使用動態類加載器，比如不安全的網絡資源或者外部存儲，因為那些代碼可能被修改為包含惡意行為。

## 本地代碼的安全

---

一般來說，對於大多數應用開發，我們鼓勵開發者使用Android SDK而不是使用[Android NDK] (<http://developer.android.com/tools/sdk/ndk/index.html>) 的native代碼。編譯native代碼的應用更為複雜，移植性差，更容易包含常見的內存崩潰錯誤，比如緩衝區溢出。

Android使用Linux內核編譯並且與Linux開發相似，如果你打算使用native代碼，安全策略尤其有用。這篇文檔討論的最佳策略實在太少了，但最受歡迎的資源之一“Secure Programming for Linux and Unix HOWTO”，在這裏可以找到<http://www.dwheeler.com/secure-programs/Android>。

與大多數Linux環境的一個重要區別是應用沙箱。在Android中，所有的應用運行在應用沙箱中，包括用native代碼編寫的應用。在最基本的級別中，與Linux相似，對於開發者來說最好的方式是知道每個應用被分配一個權限非常有限的唯一UID。這裏討論的比Android Security Overview中更細節化，你應該熟悉應用許可，即使你使用的是native代碼。

# 使用HTTPS與SSL

編寫:craftsmanBai - <http://z1ng.net> - 原文: <http://developer.android.com/training/articles/security-ssl.html>

SSL，傳輸層安全(TSL)，是一個常見的用來加密客戶端和服務器通信的模塊。但是應用程序錯誤地使用SSL可能會導致應用程序的數據在網絡中被惡意攻擊者攔截。為了幫助你確保這種情況不在你的應用中發生，這篇文章主要說明使用網絡安全協議常見的陷阱和使用Public-Key Infrastructure(PKI)時一些值得關注的問題。

## 概念

一個典型的SSL使用場景是，服務器配置中包含了一個證書，有匹配的公鑰和私鑰。作為SSL客戶端和服務端握手的一部分，服務端通過使用public-key cryptography(公鑰加密算法)進行證書簽名來證明它有私鑰。

然而，任何人都可以生成他們自己的證書和私鑰，因此一次簡單的握手不能證明服務端具有匹配證書公鑰的私鑰。一種解決這個問題的方法是讓客戶端擁有一套或者更多可信賴的證書。如果服務端提供的證書不在其中，那麼它將不能得到客戶端的信任。

這種簡單的方法有一些缺陷。服務端應該根據時間升級到強壯的密鑰(key rotation)，更新證書中的公鑰。不幸的是，現在客戶端應用需要根據服務端配置的變化來進行更新。如果服務端不在應用程序開發者的控制下，問題將變得更加麻煩，比如它是一個第三方網絡服務。如果程序需要和任意的服務器進行對話，例如web瀏覽器或者email應用，這種方法也會帶來問題。

為瞭解決這個問題，服務端通常配置了知名的發行者證書(稱為Certificate Authorities(CAs)).提供的平臺通常包含了一系列知名可信賴的CAs。Android4.2(Jelly Bean)包含了超過100CAs並在每個發行版中更新。和服務端相似的是，一個CA擁有一個證書和一個私鑰。當為一個服務端發佈頒發證書的時候，CA用它的私鑰為服務端簽名。客戶端可以通過服務端擁有被已知平臺CA簽名的證書來確認服務端。

然而，使用CAs又帶來了其他的問題。因為CA為許多服務端證書簽名，你仍然需要其他的方法來確保你對話的是你想要的服務器。為瞭解決這個問題，使用CA簽名的證書通過特殊的名字如 gmail.com 或者帶有通配符的域名如 \*.google.com 來確認服務端。下面這個例子會使這些概念具體化一些。openssl工具的客戶端命令關注Wikipedia服務端證書信息。端口為443，因為默認為HTTPS。這條命令將open s\_client的輸出發送給openssl x509，根據X.509 standard格式化證書中的內容。特別的是，這條命令需要subject，包含服務端名字和issuer來確認CA。

```
$ openssl s_client -connect wikipedia.org:443 | openssl x509 -noout -subject -issuer
subject= /serialNumber=sOrr2rKpMVP70Z6E9BT5reY008SJEyv/C=US/O=*.wikipedia.org/OU=GT03314600/OU=See www.rapidssl.com/resou
issuer= /C=US/O=GeoTrust, Inc./CN=RapidSSL CA
```

可以看到RapidSSL CA頒發給匹配\*.wikipedia.org的服務端證書。

## 一個HTTP的例子

假設你有一個知名CA頒發證書的web服務器，你可以使用下面的代碼發送一個安全請求:

```
URL url = new URL("https://wikipedia.org");
URLConnection urlConnection = url.openConnection();
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

是的，它就是這麼簡單。如果你想要修改HTTP的請求，你可以把它扔到 `HttpURLConnection`。Android關於`HttpURLConnection`文檔中還有更貼切的例子關於怎樣去處理請求、響應頭、posting的內容、cookies管理、使用代理、抓responses等等。但是就這些確認證書和域名的細節而言，Android框架已經通過API來為你考慮這些細節。下面是其他需要關注的問題。

## 服務器普通問題的驗證

假設從`getInputStream()`接受內容，會拋出一個異常：

```
javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found
    at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl.startHandshake(OpenSSLSocketImpl.java:374)
    at libcore.net.http.HttpConnection.setupSecureSocket(HttpConnection.java:209)
    at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.makeSslConnection(HttpsURLConnectionImpl.java:478)
    at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.connect(HttpsURLConnectionImpl.java:433)
    at libcore.net.http.HttpEngine.sendSocketRequest(HttpEngine.java:290)
    at libcore.net.http.HttpEngine.sendRequest(HttpEngine.java:240)
    at libcore.net.http.HttpURLConnectionImpl.getResponse(HttpURLConnectionImpl.java:282)
    at libcore.net.http.HttpURLConnectionImpl.getInputStream(HttpURLConnectionImpl.java:177)
    at libcore.net.http.HttpsURLConnectionImpl.getInputStream(HttpsURLConnectionImpl.java:271)
```

這種情況發生的原因包括：

- 1.頒佈證書給服務器的CA不是知名的。
- 2.服務器證書不是CA簽名的而是自己簽名的。
- 3.服務器配置缺失了中間CA

下面將會分別討論當你和服務器安全連接時如何去解決這些問題。

## 無法識別證書機構

在這種情況中，`SSLHandshakeException`異常產生的原因是你有一個不被系統信任的CA。可能是你的證書來源於新CA而不被安卓信任，也可能是你的應用運行版本較老沒有CA。更多的時候，一個CA不知名是因為它不是公開的CA，而是政府，公司，教育機構等組織私有的。

幸運的是，你可以教會`HttpsURLConnection`學會信任特殊的CA。過程可能會讓人感到有一些費解，下面這個例子是從`InputStream`中獲得特殊的CA，使用它去創建一個密鑰庫，用來創建和初始化`TrustManager`。`TrustManager`是系統用來驗證服務器證書的，這些證書通過使用`TrustManager`信任的CA和密鑰庫中的密鑰創建。給定一個新的`TrustManager`，下面這個例子初始化了一個新的`SSLContext`，提供了一個`SSLContextFactory`，你可以覆蓋來自`HttpsURLConnection`的默認`SSLContextFactory`。這樣連接時會使用你的CA來進行證書驗證。

下面是一個華盛頓的大學的組織性的CA的使用例子

```
// Load CAs from an InputStream
// (could be from a resource or ByteArrayInputStream or ...)
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// From https://www.washington.edu/itconnect/security/ca/load-der.crt
InputStream caInput = new BufferedInputStream(new FileInputStream("load-der.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
```

```

        System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
    } finally {
        caInput.close();
    }

    // Create a KeyStore containing our trusted CAs
    String keyStoreType = KeyStore.getDefaultType();
    KeyStore keyStore = KeyStore.getInstance(keyStoreType);
    keyStore.load(null, null);
    keyStore.setCertificateEntry("ca", ca);

    // Create a TrustManager that trusts the CAs in our KeyStore
    String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
    TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
    tmf.init(keyStore);

    // Create an SSLContext that uses our TrustManager
    SSLContext context = SSLContext.getInstance("TLS");
    context.init(null, tmf.getTrustManagers(), null);

    // Tell the URLConnection to use a SocketFactory from our SSLContext
    URL url = new URL("https://certs.cac.washington.edu/CAtest/");
    HttpsURLConnection urlConnection =
        (HttpsURLConnection)url.openConnection();
    urlConnection.setSSLSocketFactory(context.getSocketFactory());
    InputStream in = urlConnection.getInputStream();
    copyInputStreamToOutputStream(in, System.out);

```

使用一個常用的瞭解你CA的TrustManager，系統可以確認你的服務器證書來自於一個可信任的發行者。

注意:許多網站提供了簡陋的二選一方案是否安裝TrustManager。如果你這樣做還不如不加密通訊過程，因為任何人都可以在公共wifi熱點下，使用偽裝成你的服務器的代理發送你的用戶流量，進行DNS欺騙，來攻擊你的用戶。然後攻擊者便可記錄用戶密碼和其他個人資料。這種方式奏效是因為攻擊者可以生成一個證書，並且缺少可以驗證該證書是否來自受信任的來源的TrustManager。你的應用可以同任何人會話。所以不這樣做，暫時的也不行。如果你能始終讓你的應用信任服務器證書的發行者，那麼你可以這麼做。

## 自簽名服務器證書

第二種SSLHandshakeException取決於自簽名證書，意味着服務器就是它自己的CA。這同未知證書權威機構類似，因此你同樣可以用前面部分中提到的方法。

你可以創建你自己的TrustManager，這一次直接信任服務器證書。有之前提到的將你的應用直接捆綁證書的所有缺點，但是可以安全的執行。然而你應該小心確保你的自簽名證書擁有合適的強密鑰。到2012年，一個2048位65537指數位一年到期的RSA簽名是合理的。當輪換密鑰時，你應該查看權威機構(比如NIST)的建議(recommendation)來瞭解哪種密鑰是合適的。

## 缺少中間證書頒發機構

第三種SSLHandshakeException情況的產生於缺少中間CA。大多數公開的CA不直接給服務器簽名。相反，他們使用它們主要的機構(簡稱根認證機構)證書來給中間認證機構簽名，他們這樣做，因此根認證機構可以離線存儲減少危險。然而，操作系統典型的比如安卓只信任直接地根認證機構，在服務器證書(由中間證書頒發機構簽名)和證書驗證者(只知道根認證機構)之間留下了一個缺口。為瞭解決這個問題，服務器並不在SSL握手的過程中只向客戶端發送它的證書，而是一系列的從服務器到必經的任何中間機構到達根認證機構的證書。

下面是一個 mail.google.com證書鏈，以openssl\_s\_client命令顯示：

```
$ openssl s_client -connect mail.google.com:443
---
```

```
Certificate chain
0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=mail.google.com
 i:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
1 s:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
 i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority
---
```

這裏顯示了一臺服務器發送了一個Thawte SGC CA為mail.google.com頒發的證書，Thawte SGC CA是一箇中間證書頒發機構，Thawte SGC CA的證書由被安卓信任的Verisign CA頒發。然而，配置一臺服務器不包括中間證書機構是不常見的。例如，一臺服務器導致安卓瀏覽器的錯誤和應用的異常：

```
$ openssl s_client -connect egov.uscis.gov:443
---
Certificate chain
0 s:/C=US/ST=District Of Columbia/L=Washington/O=U.S. Department of Homeland Security/OU=United States Citizenship and Immigration Services/CN=egov.uscis.gov
 i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms of use at https://www.verisign.com/rpa (c)10/CN=VeriSign Class 3 Public Primary Certification Authority, Inc.
---
```

更有趣的是，用大所屬桌面瀏覽器訪問這臺服務器不會導致類似於完全未知CA的或者自簽名的服務器證書導致的錯誤。這是因為大多數桌面瀏覽器緩存隨着時間的推移信任中間證書機構。一旦瀏覽器訪問並且從一個網站瞭解到的一箇中間證書機構，下一次它將不需要中間證書機構包含證書鏈。

一些站點故意這樣做目的是讓二級服務器用來提供資源服務。比如，他們可能會讓他們的主HTML頁面用一臺擁有全部證書鏈的服務器來提供，但是像圖片，CSS,或者JavaScript等這樣的資源用不包含CA的服務器來提供，以此節省帶寬。不幸的是，有時這些服務器可能會提供一個在應用中調用的web服務。這裏有兩種解決這些問題的方法：

- 配置服務器使它包含服務器鏈中的中間證書頒發機構
- 或者，像對待不知名的CA一樣對待中間CA，並且創建一個TrustManager來直接信任它，就像在前兩節中做的那樣。

## 驗證主機名常見問題

就像在文章開頭提到的那樣，有兩個關鍵的部分來確認SSL的連接。第一個是確認證書來源於信任的源，這也是前一個部分關注的焦點。這一部分關注第二部分：確保你當前對話的服務器有正確的證書。當情況不是這樣時，你可能會看到這樣的典型錯誤：

```
java.io.IOException: Hostname 'example.com' was not verified
    at libcore.net.http.HttpConnection.verifySecureSocketHostname(HttpConnection.java:223)
    at libcore.net.http.HttpsURLConnectionImpl$HttpsEngine.connect(HttpsURLConnectionImpl.java:446)
    at libcore.net.http.HttpEngine.sendSocketRequest(HttpEngine.java:290)
    at libcore.net.http.HttpEngine.sendRequest(HttpEngine.java:240)
    at libcore.net.http.HttpURLConnectionImpl.getResponse(HttpURLConnectionImpl.java:282)
    at libcore.net.http.HttpURLConnectionImpl.getInputStream(HttpURLConnectionImpl.java:177)
    at libcore.net.http.HttpsURLConnectionImpl.getInputStream(HttpsURLConnectionImpl.java:271)
```

服務器配置錯誤可能會導致這種情況發生。服務器配置了一個證書，這個證書沒有匹配的你想連接的服務器的subject或者命名空間中二選一的subject。一個證書被許多不同的服務器使用是可能的。比如，使用 `openssl s_client -connect google.com:443 |openssl x509 -text` 查看google證書，你可以看到一個subject支持 `google.com`, `youtube.com`, `*.android.com`或者其他的。這種錯誤只會發生在你在連接的服務器名稱沒有被證書列為可接受。

不幸的是另外一種原因也會導致這種情況發生：[虛擬化服務](#)。當用HTTP同時擁有一個以上主機名的服務器共享時，web服務器可以從 HTTP/1.1請求中找到客戶端需要的目標主機名。不行的是，使用HTTPS會使情況變得複雜，因為服務

器必須知道在發現HTTP請求前返回哪一個證書。為瞭解決這個問題，新版本的SSL，特別是TLSV.1.0和之後的版本，支持服務器名指示(SNI)，允許SSL客戶端為服務端指定目標主機名，從而返回正確的證書。幸運的是，從安卓2.3開始，[HttpsURLConnection](#)支持SNI。不幸的是，Apache HTTP客戶端不這樣，這也是我們不鼓勵用它的原因之一。如果你需要支持安卓2.2或者更老的版本或者Apache HTTP客戶端，一個解決方法是建立一個可選的虛擬化服務並且使用特別的端口，這樣服務端就能夠清楚該返回哪一個證書。

採用不使用你的虛擬服務的主機名[HostnameVerifier](#)而不是服務器默認的來替換，是很重要的選擇。

注意：替換[HostnameVerifier](#)可能會非常危險，如果另外一個虛擬服務不在你的控制下，中間人攻擊可能會直接使流量到達另外一臺服務器而超出你的預想。如果你仍然確定你想覆蓋主機名驗證，這裏有一個為單[URLConnection](#)替換驗證過程的例子

```
// Create an HostnameVerifier that hardwires the expected hostname.  
// Note that is different than the URL's hostname:  
// example.com versus example.org  
HostnameVerifier hostnameVerifier = new HostnameVerifier() {  
    @Override  
    public boolean verify(String hostname, SSLSession session) {  
        HostnameVerifier hv =  
            HttpsURLConnection.getDefaultHostnameVerifier();  
        return hv.verify("example.com", session);  
    }  
};  
  
// Tell the URLConnection to use our HostnameVerifier  
URL url = new URL("https://example.org/");  
HttpsURLConnection urlConnection =  
    (HttpsURLConnection)url.openConnection();  
urlConnection.setHostnameVerifier(hostnameVerifier);  
InputStream in = urlConnection.getInputStream();  
copyInputStreamToOutputStream(in, System.out);
```

但是請記住，如果你發現你在替換主機名驗證，特別是虛擬服務，另外一個虛擬主機不在你的控制的情況是非常危險的，你應該找到一個避免這種問題產生的託管管理。

## 關於直接使用SSL Socket的警告

到目前為止，這些例子聚焦於使用[HttpsURLConnection](#)上。有時一些應用需要讓SSL和HTTP分開。舉個例子，一個email應用可能會使用SSL的變種，SMTP,POP3,IMAP等。在那些例子中，應用程序會想使用[SSLSocket](#)直接連接，與[HttpsURLConnection](#)做的方法相似。這種技術到目前為止處理了證書驗證問題，也應用於SSLSocket中。事實上，當使用常規的TrustManager時，傳遞給[HttpsURLConnection](#)的是SSLSocketFactory。如果你需要一個帶常規的SSLSocket的TrustManager，採取下面的步驟使用SSLSocketFactory來創建你的SSLSocket。注意：SSLSocket不具有主機名驗證功能。它取決於它自己的主機名驗證，通過傳入預期的主機名調用[getDefalutHostNameVerifier\(\)](#)。進一步需要注意的是，當發生錯誤時，[HostnameVerifier.verify\(\)](#)不知道拋出異常，而是返回一個布爾值，你需要進一步明確的檢查。下面是一個演示的方法。這個例子演示了當它連接gmail.com 443端口並且沒有SNI支持的時候，你將會收到一個mail.google.com的證書。你需要確保證書的確是mail.google.com的。

```
// Open SSLSocket directly to gmail.com  
SocketFactory sf = SSLSocketFactory.getDefault();  
SSLSocket socket = (SSLSocket) sf.createSocket("gmail.com", 443);  
HostnameVerifier hv = HttpsURLConnection.getDefaultHostnameVerifier();  
SSLSession s = socket.getSession();  
  
// Verify that the certificate hostname is for mail.google.com  
// This is due to lack of SNI support in the current SSLSocket.  
if (!hv.verify("mail.google.com", s)) {  
    throw new SSLHandshakeException("Expected mail.google.com, "  
        "found " + s.getPeerPrincipal());
```

```
}

// At this point SSLSocket performed certificate verification and
// we have performed hostname verification, so it is safe to proceed.

// ... use socket ...
socket.close();
```

## 黑名單

SSL 主要依靠CA來確認證書來自正確無誤服務器和域名的所有者。少數情況下，CA被欺騙，或者在[Comodo](#)和[DigiNotar](#)的例子中，一個主機名的證書被頒發給了除了服務器和域名的擁有者之外的人，導致被破壞。

為了減少這着危險，安卓可以將一些黑名單或者整個CA列入黑名單。儘管名單是以前是嵌入操作系統的，從安卓4.2開始，這個名單在以後的方案中可以遠程更新。

## 阻塞

一個應用可以通過阻塞技術保護它自己免於受虛假證書的欺騙。這是簡單運用使用未知CA的例子，限制應用信任的CA僅來自被應用使用的服務器。阻止了來自系統中另外一百多個CA的欺騙而導致的應用安全通道的破壞。

## 客戶端驗證

這篇文章聚焦在SSL的使用者同服務器的安全對話上。SSL也支持服務端通過驗證客戶端的證書來確認客戶端的身份。這種技術也與TrustManager的特性相似。可以參考在[HttpsURLConnection](#)文檔中關於創建一個常規的[KeyManager](#)的討論。

## nogotofail：網絡流量安全測試工具

對於已知的TLS／SSL漏洞和錯誤，nogotofail提供了一個簡單的方法來確認你的應用程序是安全的。它是一個自動化的、強大的、用於測試網絡的安全問題可擴展性的工具，任何設備的網絡流量都可以通過它。 nogotofail主要應用於三種場景：

- 發現錯誤和漏洞。
- 驗證修補程序和等待迴歸。
- 瞭解應用程序和設備產生的交通。

nogotofail 可以工作在Android，iOS，Linux，Windows，Chrome OS，OSX環境下，事實上任何需要連接到Internet的設備都可以。Android和Linux環境下有簡單易用獲取通知的客戶端配置設置，以及本身可以作為靶機，部署為一個路由器，VPN服務器，或代理。 你可以在nogotofail開源項目訪問該工具。

# 更新你的Security Provider來對抗SSL漏洞利用

編寫:craftsmanBai - <http://z1ng.net> - 原文: <http://developer.android.com/training/articles/security-gms-provider.html>

安卓依靠security provider保障網絡通信安全。然而有時默認的security provider存在安全漏洞。為了防止這些漏洞被利用，Google Play services 提供了一個自動更新設備的security provider的方法來對抗已知的漏洞。通過調用Google Play services方法，可以確保你的應用運行在可以抵抗已知漏洞的設備上。

舉個例子，OpenSSL的漏洞(CVE-2014-0224)會導致中間人攻擊，在通信雙方不知情的情況下解密流量。Google Play services 5.0提供了一個補丁，但是必須確保應用安裝了這個補丁。通過調用Google Play services方法，可以確保你的應用運行在可抵抗攻擊的安全設備上。

注意：更新設備的security provider不是更新[android.net.SSLCertificateSocketFactory](#).比起使用這個類，我們更鼓勵應用開發者使用融入密碼學的高級方法。大多數應用可以使用類似[HttpsURLConnection](#)，[HttpClient](#)，[AndroidHttpClient](#)這樣的API，而不必去設置[TrustManager](#)或者創建一個[SSLCertificateSocketFactory](#)。

## 使用ProviderInstaller給Security Provider打補丁

使用providerinstaller類來更新設備的security provider。你可以通過調用該類的方法[installIfNeeded\(\)](#)(或者[installIfNeededAsync\(\)](#))來驗證security provider是否為最新的(必要的話更新它)。

當你調用[installIfNeeded](#)時，[providerinstaller](#)會做以下事情：

- 如果設備的Provider成功更新(或已經是最新的)，該方法返回正常。
- 如果設備的Google Play services 庫已經過時了，這個方法拋出[googleplayservicesrepairableexception](#)異常表明無法更新Provider。應用程序可以捕獲這個異常並向用戶彈出合適的對話框提示更新Google Play services。
- 如果產生了不可恢復的錯誤，該方法拋出[googleplayservicesnotavailableexception](#)表示它無法更新Provider。應用程序可以捕獲異常並選擇合適的行動，如顯示標準問題解決流程圖。

[installIfNeededAsync](#)方法類似，但它不拋出異常，而是通過相應的回調方法，以提示成功或失敗。

如果[installIfNeeded](#)需要安裝一個新的Provider，可能耗費30-50毫秒（較新的設備）到350毫秒（舊設備）。如果security provider已經是最新的，該方法需要的時間量可以忽略不計。為了避免影響用戶體驗：

- 線程加載後立即在後臺網絡線程中調用[installIfNeeded](#)，而不是等待線程嘗試使用網絡。（多次調用該方法沒有害處，如果安全提供程序不需要更新它會立即返回。）
- 如果用戶體驗會受線程阻塞的影響——比如從UI線程中調用，那麼使用[installIfNeededAsync\(\)](#)調用該方法的異步版本。（當然，如果你要這樣做，在嘗試任何安全通信之前必須等待操作完成。[providerinstaller](#)調用監聽者的[onProviderInstalled\(\)](#)方法發出成功信號。）

警告：如果[providerinstaller](#)無法安裝更新Provider，您的設備security provider會容易受到已知漏洞的攻擊。你的程序等同於所有HTTP通信未被加密。一旦Provider更新，所有安全API（包括SSL API）的調用會經過它(但這並不適用於[android.net.sslcertificatesocketfactory](#)，面對cve-2014-0224這種漏洞仍然是脆弱的)。

## 同步修補

修補security provider最簡單的方法就是調用同步方法[installIfNeeded\(\)](#)。如果用戶體驗不會被線程阻塞影響的話，這種方法很合適。

舉個例子，這裏有一個sync adapter會更新security provider。由於它運行在後臺，因此在等待security provider更新的時候線程阻塞是可以的。sync adapter調用[installifneeded\(\)](#)更新security provider。如果返回正常，sync adapter可以確保security provider是最新的。如果返回異常，sync adapter可以採取適當的行動（如提示用戶更新Google Play services）。

```
/**  
 * Sample sync adapter using {@link ProviderInstaller}.  
 */  
public class SyncAdapter extends AbstractThreadingSyncAdapter {  
  
    ...  
  
    // This is called each time a sync is attempted; this is okay, since the  
    // overhead is negligible if the security provider is up-to-date.  
    @Override  
    public void onPerformSync(Account account, Bundle extras, String authority,  
        ContentProviderClient provider, SyncResult syncResult) {  
        try {  
            ProviderInstaller.installIfNeeded(getApplicationContext());  
        } catch (GooglePlayServicesRepairableException e) {  
  
            // Indicates that Google Play services is out of date, disabled, etc.  
  
            // Prompt the user to install/update/enable Google Play services.  
            GooglePlayServicesUtil.showErrorNotification(  
                e.getConnectionStatusCode(), getApplicationContext());  
  
            // Notify the SyncManager that a soft error occurred.  
            syncResult.stats.numIOExceptions++;  
            return;  
  
        } catch (GooglePlayServicesNotAvailableException e) {  
            // Indicates a non-recoverable error; the ProviderInstaller is not able  
            // to install an up-to-date Provider.  
  
            // Notify the SyncManager that a hard error occurred.  
            syncResult.stats.numAuthExceptions++;  
            return;  
        }  
  
        // If this is reached, you know that the provider was already up-to-date,  
        // or was successfully updated.  
    }  
}
```

## 異步修補

更新security provider可能耗費350毫秒（舊設備）。如果在一個會直接影響用戶體驗的線程中更新，如UI線程，那麼你不會希望進行同步更新，因為這可能導致應用程序或設備凍結直到操作完成。因此你應該使用異步方法[installifneededasync\(\)](#)。方法通過調用回調函數來反饋其成功或失敗。例如，下面是一些關於更新security provider在UI線程中的活動的代碼。調用[installifneededasync\(\)](#)來更新security provider，並指定自己為監聽器接收成功或失敗的通知。如果security provider是最新的或更新成功，會調用[onproviderinstalled\(\)](#)方法，並且知道通信是安全的。如果security provider無法更新，會調用[onproviderinstallfailed\(\)](#)方法，並採取適當的行動（如提示用戶更新Google Play services）

```
/**  
 * Sample activity using {@link ProviderInstaller}.  
 */  
public class MainActivity extends Activity  
    implements ProviderInstaller.ProviderInstallListener {  
  
    private static final int ERROR_DIALOG_REQUEST_CODE = 1;  
  
    private boolean mRetryProviderInstall;  
  
    //Update the security provider when the activity is created.  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ProviderInstaller.installIfNeededAsync(this, this);  
    }  
  
    /**  
     * This method is only called if the provider is successfully updated  
     * (or is already up-to-date).  
     */  
    @Override  
    protected void onProviderInstalled() {  
        // Provider is up-to-date, app can make secure network calls.  
    }  
  
    /**  
     * This method is called if updating fails; the error code indicates  
     * whether the error is recoverable.  
     */  
    @Override  
    protected void onProviderInstallFailed(int errorCode, Intent recoveryIntent) {  
        if (GooglePlayServicesUtil.isUserRecoverableError(errorCode)) {  
            // Recoverable error. Show a dialog prompting the user to  
            // install/update/enable Google Play services.  
            GooglePlayServicesUtil.showErrorDialogFragment(  
                errorCode,  
                this,  
                ERROR_DIALOG_REQUEST_CODE,  
                new DialogInterface.OnCancelListener() {  
                    @Override  
                    public void onCancel(DialogInterface dialog) {  
                        // The user chose not to take the recovery action  
                        onProviderInstallerNotAvailable();  
                    }  
                });  
        } else {  
            // Google Play services is not available.  
            onProviderInstallerNotAvailable();  
        }  
    }  
  
    @Override  
    protected void onActivityResult(int requestCode, int resultCode,  
        Intent data) {  
        super.onActivityResult(requestCode, resultCode, data);  
        if (requestCode == ERROR_DIALOG_REQUEST_CODE) {  
            // Adding a fragment via GooglePlayServicesUtil.showErrorDialogFragment  
            // before the instance state is restored throws an error. So instead,  
            // set a flag here, which will cause the fragment to delay until  
            // onPostResume.  
            mRetryProviderInstall = true;  
        }  
    }  
  
    /**  
     * On resume, check to see if we flagged that we need to reinstall the  
     * provider.  
     */  
    @Override  
    protected void onPostResume() {  
        super.onPostResume();  
        if (mRetryProviderInstall) {
```

```
// We can now safely retry installation.  
ProviderInstall.installIfNeededAsync(this, this);  
}  
mRetryProviderInstall = false;  
}  
  
private void onProviderInstallerNotAvailable() {  
    // This is reached if the provider cannot be updated for some reason.  
    // App should consider all HTTP communication to be vulnerable, and take  
    // appropriate action.  
}  
}
```

# 使用設備管理策略增強安全性

編寫:craftsmanBai - <http://z1ng.net> - 原文: <http://developer.android.com/training/enterprise/device-management-policy.html>

Android 2.2(API Level 8)之後，Android平臺通過設備管理API提供系統級的設備管理能力。

在這一小節中，你將學到如何通過使用設備管理策略創建安全敏感的應用程序。比如某應用可被配置為：在給用戶顯示受保護的內容之前，確保已設置一個足夠強度的鎖屏密碼。

## 定義並聲明你的策略

首先，你需要定義多種在功能層面提供支持的策略。這些策略可以包括屏幕鎖密碼強度、密碼過期時間以及加密等等方面。

你須在res/xml/device\_admin.xml中聲明選擇的策略集，它將被應用強制實行。在Android manifest也需要引用聲明的策略集。

每個聲明的策略對應DevicePolicyManager中一些相關設備的策略方法（例如定義最小密碼長度或最少大寫字母字符數）。如果一個應用嘗試調用XML中沒有對應策略的方法，程序在會運行時拋出一個SecurityException異常。

如果應用程序試圖管理其他策略，那麼強制鎖force-lock之類的其他權限就會發揮作用。正如你將看到的，作為設備管理權限激活過程的一部分，聲明策略的列表會在系統屏幕上顯示給用戶。如下代碼片段在res/xml/device\_admin.xml中聲明瞭密碼限制策略：

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-policies>
        <limit-password />
    </uses-policies>
</device-admin>
```

在Android manifest引用XML策略聲明：

```
<receiver android:name=".Policy$PolicyAdmin"
    android:permission="android.permission.BIND_DEVICE_ADMIN">
    <meta-data android:name="android.app.device_admin"
        android:resource="@xml/device_admin" />
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
```

## 創建一個設備管理接受端

創建一個設備管理廣播接收端（broadcast receiver），可以接收到與你聲明的策略有關的事件通知。也可以對應用程序有選擇地重寫回調函數。

在同樣的應用程序（Device Admin）中，當設備管理（device administrator）權限被用戶設為禁用時，已配置好的策略就會從共享偏好設置（shared preference）中擦除。

你應該考慮實現與你的應用業務邏輯相關的策略。例如，你的應用可以採取一些措施來降低安全風險，如：刪除設備上的敏感數據，禁用遠程同步，對管理員的通知提醒等等。

為了讓廣播接收端能夠正常工作，請務必在Android manifest中註冊下面代碼片段所示內容。

```
<receiver android:name=".Policy$PolicyAdmin">
    android:permission="android.permission.BIND_DEVICE_ADMIN">
        <meta-data android:name="android.app.device_admin"
            android:resource="@xml/device_admin" />
        <intent-filter>
            <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
        </intent-filter>
    </receiver>
```

## 激活設備管理器

在執行任何策略之前，用戶需要手動將程序激活為具有設備管理權限，下面的程序片段顯示瞭如何觸發設置框以便讓用戶為你的程序激活權限。

通過指定[EXTRA\\_ADD\\_EXPLANATION](#)給出明確的說明信息，以告知用戶為應用程序激活設備管理權限的好處。

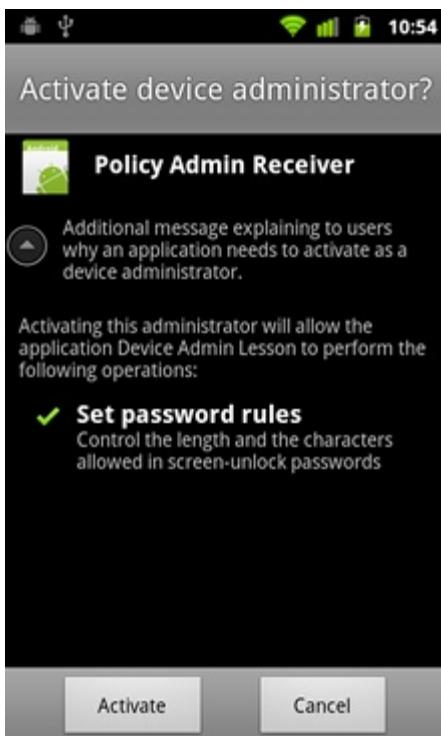
```
if (!mPolicy.isAdminActive()) {

    Intent activateDeviceAdminIntent =
        new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);

    activateDeviceAdminIntent.putExtra(
        DevicePolicyManager.EXTRA_DEVICE_ADMIN,
        mPolicy.getPolicyAdmin());

    // It is good practice to include the optional explanation text to
    // explain to user why the application is requesting to be a device
    // administrator. The system will display this message on the activation
    // screen.
    activateDeviceAdminIntent.putExtra(
        DevicePolicyManager.EXTRA_ADD_EXPLANATION,
        getResources().getString(R.string.device_admin_activation_message));

    startActivityForResult(activateDeviceAdminIntent,
        REQ_ACTIVATE_DEVICE_ADMIN);
}
```



如果用戶選擇"Activate"，程序就會獲取設備管理員權限並可以開始配置和執行策略。當然，程序也需要做好處理用戶選擇放棄激活的準備，比如用戶點擊了“取消”按鈕，返回鍵或者HOME鍵的情況。因此，如果有必要的話，策略設置中的`onResume()`方法需要加入重新評估的邏輯判斷代碼，以便將設備管理激活選項展示給用戶。

## 實施設備策略控制

在設備管理權限成功激活後，程序就會根據請求的策略來配置設備策略管理器。要牢記，新策略會被添加到每個版本的Android中。所以你需要在程序中做好平臺版本的檢測，以便新策略能被老版本平臺很好的支持。例如，“密碼中含有的最少大寫字符數”這個安全策略只有在高於API Level 11（Honeycomb）的平臺才被支持，以下代碼則演示瞭如何在運行時檢查版本：

```
DevicePolicyManager mDPM = (DevicePolicyManager)
    context.getSystemService(Context.DEVICE_POLICY_SERVICE);
ComponentName mPolicyAdmin = new ComponentName(context, PolicyAdmin.class);
...
mDPM.setPasswordQuality(mPolicyAdmin, PASSWORD_QUALITY_VALUES[mPasswordQuality]);
mDPM.setPasswordMinimumLength(mPolicyAdmin, mPasswordLength);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    mDPM.setPasswordMinimumUpperCase(mPolicyAdmin, mPasswordMinUpperCase);
}
```

這樣程序就可以執行策略了。當程序無法訪問正確的鎖屏密碼的時候，通過設備策略管理器（Device Policy Manager）API可以判斷當前密碼是否適用於請求的策略。如果當前鎖屏密碼滿足策略，設備管理API不會採取糾正措施。明確地啓動設置程序中的系統密碼更改界面是應用程序的責任。例如：

```
if (!mDPM.isActivePasswordSufficient()) {
    ...
    // Triggers password change screen in Settings.
    Intent intent =
        new Intent(DevicePolicyManager.ACTION_SET_NEW_PASSWORD);
    startActivity(intent);
}
```

一般來說，用戶可以從可用的鎖屏機制中任選一個，例如“無”、“圖案”、“PIN碼”（數字）或密碼（字母數字）。當一個密碼策略配置好後，那些比已定義密碼策略弱的密碼會被禁用。比如，如果配置了密碼級別為“Numeric”，那麼用戶只可以選擇PIN碼（數字）或者密碼（字母數字）。

一旦設備通過設置適當的鎖屏密碼處於被保護的狀態，應用程序便允許訪問受保護的內容。

```
if (!mDPM.isAdminActive(..)) {  
    // Activates device administrator.  
    ...  
} else if (!mDPM.isActivePasswordSufficient()) {  
    // Launches password set-up screen in Settings.  
    ...  
} else {  
    // Grants access to secure content.  
    ...  
    startActivity(new Intent(context, SecureActivity.class));  
}
```

# 測試程序

---

編寫:kesenhoo - 原文:<http://developer.android.com/training/testing.html>

These classes and articles provide information about how to test your Android application.

## Testing Your Activity

How to test Activities in your Android applications.

# 測試你的Activity

---

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/index.html>

我們應該把編寫和運行測試作為Android應用開發週期的一部分。完備的測試可以幫助我們在開發過程中儘早發現漏洞，並讓我們對自己的代碼更有信心。

測試用例定義了一系列對象和方法從而獨立進行多個測試。測試用例可以編寫成測試組並按計劃的運行，由測試框架組織成一個可以重複運行的測試Runner（運行器，譯者注）。

這節內容將會講解如何基於最流行的JUnit框架來自定義測試框架。我們可以編寫測試用例來測試我們應用程序的特定行為，並在不同的Android設備上檢測一致性。測試用例還可以用來描述應用組件的預期行為，並作為內部代碼文檔。

## 課程

---

- [建立測試環境](#)

學習如何創建測試項目

- [創建與執行測試用例](#)

學習如何寫測試用例來檢驗Activity中的特性，並使用Android框架提供的Instrumentation運行用例。

- [測試UI組件](#)

學習如何編寫UI測試用例

- [創建單元測試](#)

學習如何隔離開Activity執行單元測試

- [創建功能測試](#)

學習如何執行功能測試來檢驗各Activity之間的交互

# 建立測試環境

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/preparing-activity-testing.html>

在開始編寫並運行我們的測試之前，我們應該建立測試開發環境。本小節將會講解如何建立Eclipse IDE來構建和運行我們的測試，以及怎樣用Gradle構建工具在命令行下構建和運行我們的測試。

注意: 本小節基於的是Eclipse及ADT插件。然而，你在自己測試開發時可以自由選用IDE或命令行。

## 用Eclipse建立測試

安裝了Android Developer Tools (ADT) 插件的Eclipse將為我們創建，構建，以及運行Android程序提供一個基於圖形界面的集成開發環境。Eclipse可以自動為我們的Android應用項目創建一個對應的測試項目。

開始在Eclipse中創建測試環境:

1. 如果還沒安裝Eclipse ADT插件，請先下載安裝。
2. 導入或創建我們想要測試的Android應用項目。
3. 生成一個對應於應用程序項目測試的測試項目。為導入項目生成一個測試項目: a.在項目瀏覽器裏，右擊我們的應用項目，然後選擇Android Tools > New Test Project b.在新建Android測試項目面板，為我們的測試項目設置合適的參數，然後點擊Finish

現在應該可以在Eclipse環境中創建，構建和運行測試項目了。想要繼續學習如何在Eclipse中進行這些任務，可以閱讀[創建與執行測試用例](#)

## 用命令行建立測試

如果正在使用Gradle version 1.6或者更高的版本作為構建工具，可以用Gradle Wrapper創建。構建和運行Android應用測試。確保在 `gradle.build` 文件中， `defaultConfig` 部分中的`minSdkVersion`屬性是8或更高。可以參考包含在下載包中的示例文件`gradle.build`

## 用Gradle Wrapper運行測試:

1. 連接Android真機或開啟Android模擬器。
2. 在項目目錄運行如下命令:

```
./gradlew build connectedCheck
```

進一步學習Gradle關於Android測試的內容，參看[Gradle Plugin User Guide](#)。

進一步學習使用Gradle及其它命令行工具，參看[Testing from Other IDEs](#)。

本節示例代碼[AndroidTestingFun.zip](#)

# 創建與執行測試用例

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/activity-basic-testing.html>

為了驗證應用的佈局設計和功能是否符合預期，為應用的每個Activity建立測試非常重要。對於每一個測試，我們需要在測試用例中創建一個個獨立的部分，包括測試數據，前提條件和Activity的測試方法。之後我們就可以運行測試並得到測試報告。如果有任何測試沒有通過，這表明在我們代碼中可能有潛在的缺陷。

注意: 在測試驅動開發 (TDD) 方法中，不推薦先編寫大部分或整個應用，並在開發完成後再運行測試。而是應該先編寫測試，然後及時編寫正確的代碼，以通過測試。通過更新測試案例來反映新的功能需求，並以此反覆。

## 創建一個測試用例

Activity測試都是通過結構化的方式編寫的。請務必把測試代碼放在一個單獨的包內，從而與被測試的代碼分開。

按照慣例，測試包的名稱應該遵循與應用包名相同的命名方式，在應用包名後接“.tests”。在創建的測試包中，為我們的測試用例添加Java類。按照慣例，測試用例名稱也應遵循要測試的Java或Android的類相同的名稱，並增加後綴“Test”。

要在Eclipse中創建一個新的測試用例可遵循如下步驟：

- a. 在Package Explorer中，右鍵點擊待測試工程的src/文件夾，New > Package。
- b. 設置文件夾名稱 <你的包名稱>.tests (比如, com.example.android.testingfun.tests) 並點擊Finish。
- c. 右鍵點擊創建的測試包，並選擇New > Calss。
- d. 設置文件名稱 <你的Activity名稱>Test (比如, MyFirstTestActivityTest)，然後點擊Finish。

## 建立測試數據集(Fixture)

測試數據集包含運行測試前必鬚生成的一些對象。要建立測試數據集，可以在我們的測試中覆寫setUp()和tearDown()方法。測試會在運行任何其它測試方法之前自動執行setUp()方法。我們可以用這些方法使得被測試代碼與測試初始化和清理是分開的。

在你的Eclipse中建立測試數據集：

1. 在 Package Explorer中雙擊測試打開之前編寫的測試用例，然後修改測試用例使它繼承ActivityTestCase的子類。比如：

```
public class MyFirstTestActivityTest  
    extends ActivityInstrumentationTestCase2<MyFirstTestActivity> {
```

2. 下一步，給測試用例添加構造函數和setUp()方法，併為我們想測試的Activity添加變量聲明。比如：

```
public class MyFirstTestActivityTest  
    extends ActivityInstrumentationTestCase2<MyFirstTestActivity> {  
  
    private MyFirstTestActivity mFirstTestActivity;
```

```

private TextView mFirstTestText;

public MyFirstTestActivityTest() {
    super(MyFirstTestActivity.class);
}

@Override
protected void setUp() throws Exception {
    super.setUp();
    mFirstTestActivity = getActivity();
    mFirstTestText =
        (TextView) mFirstTestActivity
            .findViewById(R.id.my_first_test_text_view);
}
}

```

構造函數是由測試用的Runner調用，用於初始化測試類的，而setUp()方法是由測試Runner在其他測試方法開始前運行的。

通常在 setUp() 方法中，我們應該：

- 為 setUp() 調用父類構造函數，這是JUnit要求的。
- 初始化測試數據集的狀態，具體而言：
  - 定義保存測試數據及狀態的實例變量
  - 創建並保存正在測試的Activity的引用實例。
  - 獲得想要測試的Activity中任何UI組件的引用。

我們可以使用getActivity()方法得到正在測試的Activity的引用。

## 增加一個測試前提

我們最好在執行測試之前，檢查測試數據集的設置是否正確，以及我們想要測試的對象是否已經正確地初始化。這樣，測試就不會因為有測試數據集的設置錯誤而失敗。按照慣例，驗證測試數據集的方法被稱為 testPreconditions()。

例如，我們可能想添加一個像這樣的 testPreconditions() 方法：

```

public void testPreconditions() {
    assertNotNull("mFirstTestActivity is null", mFirstTestActivity);
    assertNotNull("mFirstTestText is null", mFirstTestText);
}

```

Assertion（斷言，譯者注）方法源自於JunitAssert類。通常，我們可以使用斷言來驗證某一特定的條件是否是真的。

- 如果條件為假，斷言方法拋出一個AssertionFailedError異常，通常會由測試Runner報告。我們可以在斷言失敗時給斷言方法添加一個字符串作為第一個參數從而給出一些上下文詳細信息。
- 如果條件為真，測試通過。

在這兩種情況下，Runner都會繼續運行其它測試用例的測試方法。

## 添加一個測試方法來驗證Activity

下一步，添加一個或多個測試方法來驗證Activity佈局和功能。

例如，如果我們的Activity含有一個TextView，可以添加如下方法來檢查它是否有正確的標籤文本：

```
public void testMyFirstTestTextView_labelText() {  
    final String expected =  
        mFirstTestActivity.getString(R.string.my_first_test);  
    final String actual = mFirstTestText.getText().toString();  
    assertEquals(expected, actual);  
}
```

該 `testMyFirstTestTextView_labelText()` 方法只是簡單的檢查Layout中TextView的默認文本是否和 `strings.xml` 資源中定義的文本一樣。

注意：當命名測試方法時，我們可以使用下劃線將被測試的內容與測試用例區分開。這種風格使得我們可以更容易分清哪些是測試用例。

做這種類型的字符串比較時，推薦從資源文件中讀取預期字符串，而不是在代碼中硬性編寫字符串做比較。這可以防止當資源文件中的字符串定義被修改時，會影響到測試的效果。

為了進行比較，預期的和實際的字符串都要做為 `assertEquals()` 方法的參數。如果值是不一樣的，斷言將拋出一個 `AssertionFailedError` 異常。

如果添加了一個 `testPreconditions()` 方法，我們可以把測試方法放在 `testPreconditions` 之後。

要參看一個完整的測試案例，可以參考本節示例中的 `MyFirstTestActivityTest.java`。

## 構建和運行測試

我們可以在Eclipse中的包瀏覽器（Package Explorer）中運行我們的測試。

利用如下步驟構建和運行測試：

1. 連接一個Android設備，在設備或模擬器中，打開設置菜單，選擇開發者選項並確保啓用USB調試。
2. 在包瀏覽器(Package Explorer)中，右鍵單擊測試類，並選擇Run As > Android Junit Test。
3. 在Android設備選擇對話框，選擇剛纔連接的設備，然後單擊“確定”。
4. 在JUnit視圖，驗證測試是否通過，有無錯誤或失敗。

本節示例代碼[AndroidTestingFun.zip](#)

# 測試UI組件

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/activity-ui-testing.html>

通常情況下，[Activity](#)，包括用戶界面組件（如按鈕，複選框，可編輯的文本域，和選框）允許用戶與Android應用程序交互。本節介紹如何對一個簡單的帶有按鈕的界面交互測試。我們可以使用相同的步驟來測試其他更複雜的UI組件。

注意：這一節的測試方法叫做白盒測試，因為我們擁有要測試應用程序的源碼。Android Instrumentation框架適用於創建應用程序中UI部件的白盒測試。用戶界面測試的另一種類型是黑盒測試，即無法得知應用程序源代碼的類型。這種類型的測試可以用來測試應用程序如何與其他應用程序，或與系統進行交互。黑盒測試不包括在本節中。瞭解更多關於如何在你的Android應用程序進行黑盒測試，請閱讀[UI Testing guide](#)。

要參看完整的測試案例，可以查看本節示例代碼中的 `ClickFunActivityTest.java` 文件。

## 使用 Instrumentation 建立UI測試

當測試擁有UI的[Activity](#)時，被測試的[Activity](#)在UI線程中運行。然而，測試程序會在程序自己的進程中，單獨的一個線程內運行。這意味着，我們的測試程序可以獲得UI線程的對象，但是如果它嘗試改變UI線程對象的值，會得到 `WrongThreadException` 錯誤。

為了安全地將 `Intent` 注入到 `Activity`，或是在UI線程中執行測試方法，我們可以讓測試類繼承於[ActivityInstrumentationTestCase2](#)。要學習如何在UI線程運行測試方法，請看[在UI線程測試](#)。

## 建立測試數據集（Fixture）

當為UI測試建立測試數據集時，我們應該在[setUp\(\)](#)方法中指定[touch mode](#)。把[touch mode](#)設置為真可以防止在執行編寫的測試方法時，人為的UI操作獲取到控件的焦點（比如，一個按鈕會觸發它的點擊監聽器）。確保在調用[getActivity\(\)](#)方法前調用了[setActivityInitialTouchMode\(\)](#)。

比如：

```
public class ClickFunActivityTest
    extends ActivityInstrumentationTestCase2 {
    ...
    @Override
    protected void setUp() throws Exception {
        super.setUp();

        setActivityInitialTouchMode(true);

        mClickFunActivity = getActivity();
        mClickMeButton = (Button)
            mClickFunActivity
                .findViewById(R.id.launch_next_activity_button);
        mInfoTextView = (TextView)
            mClickFunActivity.findViewById(R.id.info_text_view);
    }
}
```

## 添加測試方法確認UI響應表現

UI測試目標應包括：

. 檢驗Activity啓動時Button在正確佈局位置顯示。 . 檢驗TextView初始化時是隱藏的。 \*. 檢驗TextView在Button點擊時顯示預期的字符串

接下來的部分會演示怎樣實現上述驗證方法

## 驗證Button佈局參數

我們應該像如下添加的測試方法那樣。驗證Activity中的按鈕是否正確顯示:

```
@MediumTest
public void testClickMeButton_layout() {
    final View decorView = mClickFunActivity.getWindow().getDecorView();

    ViewAsserts.assertOnScreen(decorView, mClickMeButton);

    final ViewGroup.LayoutParams layoutParams =
        mClickMeButton.getLayoutParams();
    assertNotNull(layoutParams);
    assertEquals(layoutParams.width, WindowManager.LayoutParams.MATCH_PARENT);
    assertEquals(layoutParams.height, WindowManager.LayoutParams.WRAP_CONTENT);
}
```

在調用assertOnScreen()方法時，傳遞根視圖以及期望呈現在屏幕上的視圖作為參數。如果想呈現的視圖沒有在根視圖中，該方法會拋出一個AssertionFailedError異常，否則測試通過。

我們也可以通過獲取一個ViewGroup.LayoutParams對象的引用驗證Button佈局是否正確，然後調用 assert 方法驗證Button對象的寬高屬性值是否與預期值一致。

@MediumTest 註解指定測試是如何歸類的（和它的執行時間相關）。要瞭解更多有關測試的註解，見本節示例。

## 驗證TextView的佈局參數

可以像這樣添加一個測試方法來驗證TextView最初是隱藏在Activity中的:

```
@MediumTest
public void testInfoTextView_layout() {
    final View decorView = mClickFunActivity.getWindow().getDecorView();
    ViewAsserts.assertOnScreen(decorView, mInfoTextView);
    assertTrue(View.GONE == mInfoTextView.getVisibility());
}
```

我們可以調用 getDecorView() 方法得到一個Activity中修飾試圖（Decor View）的引用。要修飾的View在佈局層次視圖中是最上層的ViewGroup(FrameLayout)

## 驗證按鈕的行為

可以使用如下測試方法來驗證當按下按鈕時TextView變得可見:

```
@MediumTest
public void testClickMeButton_clickButtonAndExpectInfoText() {
    String expectedInfoText = mClickFunActivity.getString(R.string.info_text);
    TouchUtils.clickView(this, mClickMeButton);
    assertTrue(View.VISIBLE == mInfoTextView.getVisibility());
    assertEquals(expectedInfoText, mInfoTextView.getText());
}
```

在測試中調用[clickView\(\)](#)可以讓我們用編程方式點擊一個按鈕。我們必須傳遞正在運行的測試用例的一個引用和要操作按鈕的引用。

注意:[TouchUtils](#)輔助類提供與應用程序交互的方法可以方便進行模擬觸摸操作。我們可以使用這些方法來模擬點擊，輕敲，或應用程序屏幕拖動View。

警告[TouchUtils](#)方法的目的是將事件安全地從測試線程發送到UI線程。我們不可以直接在UI線程或任何標註@UiThread的測試方法中使用[TouchUtils](#)這樣做可能會增加錯誤線程異常。

## 應用測試註解

### @SmallTest

標誌該測試方法是小型測試的一部分。

### @MediumTest

標誌該測試方法是中等測試的一部分。

### @LargeTest

標誌該測試方法是大型測試的一部分。

通常情況下，如果測試方法只需要幾毫秒的時間，那麼它應該被標記為[@SmallTest](#)，長時間運行的測試（100毫秒或更多）通常被標記為[@MediumTest](#)或[@LargeTest](#)，這主要取決於測試訪問資源在網絡上或在本地系統。可以參看[Android Tools Protip](#)，它可以更好地指導我們使用測試註釋。

我們可以創建其它的測試註釋來控制測試的組織和運行。要瞭解更多關於其他註釋的信息，見[Annotation](#)類參考。

本節示例代碼[AndroidTestingFun.zip](#)

# 創建單元測試

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/activity-unit-testing.html>

**Activity**單元測試可以快速且獨立地（和系統其它部分分離）驗證一個**Activity**的狀態以及其與其它組件交互的正確性。一個單元測試通常用來測試代碼中最小單位的代碼塊（可以是一個方法，類，或者組件），而且也不依賴於系統或網絡資源。比如說，你可以寫一個單元測試去檢查**Activity**是否正確地佈局或者是否可以正確地觸發一個**Intent**對象。

單元測試一般不適合測試與系統有複雜交互的UI。我們應該使用如同**測試UI組件**所描述的 `ActivityInstrumentationTestCase2` 來對這類UI交互進行測試。

這節內容將會講解如何編寫一個單元測試來驗證一個**Intent**是否正確地觸發了另一個**Activity**。由於測試是與環境獨立的，所以**Intent**實際上並沒有發送給Android系統，但我們可以檢查**Intent**對象的載荷數據是否正確。讀者可以參考一下示例代碼中的 `LaunchActivityTest.java`，將它作為一個例子，瞭解完備的測試用例是怎麼樣的。

注意: 如果要針對系統或者外部依賴進行測試，我們可以使用Mocking Framework的Mock類，並把它集成到我們的你的單元測試中。要瞭解更多關於Android提供的Mocking Framework內容請參考[Mock Object Classes](#)。

## 編寫一個Android單元測試例子

`ActivityUnitTestCase`類提供對於單個**Activity**進行分離測試的支持。要創建單元測試，我們的測試類應該繼承自 `ActivityUnitTestCase`。繼承 `ActivityUnitTestCase` 的**Activity**不會被Android自動啓動。要單獨啓動**Activity**，我們需要顯式的調用**startActivity()**方法，並傳遞一個**Intent**來啓動我們的目標**Activity**。

例如：

```
public class LaunchActivityTest
    extends ActivityUnitTestCase<LaunchActivity> {
    ...
    @Override
    protected void setUp() throws Exception {
        super.setUp();
        mLaunchIntent = new Intent(getApplicationContext(),
            .getTargetContext(), LaunchActivity.class);
        startActivity(mLaunchIntent, null, null);
        final Button launchNextButton =
            (Button) getActivity()
            .findViewById(R.id.launch_next_activity_button);
    }
}
```

## 驗證另一個**Activity**的啓動

我們的單元測試目標可能包括:

- 驗證當Button被按下時，啓動的**LaunchActivity**是否正確。
- 驗證啓動的**Intent**是否包含有效的數據。

為了驗證一個觸發**Intent**的Button的事件，我們可以使用**getStartedActivityIntent()**方法。通過使用斷言方法，我們可以驗證返回的**Intent**是否為空，以及是否包含了預期的數據來啓動下一個**Activity**。如果兩個斷言值都是真，那麼我們就成功地驗證了**Activity**發送的**Intent**是正確的了。

我們可以這樣實現測試方法：

```
@MediumTest
public void testNextActivityWasLaunchedWithIntent() {
    startActivity(mLaunchIntent, null, null);
    final Button launchNextButton =
        (Button) getActivity()
            .findViewById(R.id.launch_next_activity_button);
    launchNextButton.performClick();

    final Intent launchIntent = getStartedActivityIntent();
    assertNotNull("Intent was null", launchIntent);
    assertTrue(isFinishCalled());

    final String payload =
        launchIntent.getStringExtra(NextActivity.EXTRA_PAYLOAD_KEY);
    assertEquals("Payload is empty", LaunchActivity.STRING_PAYLOAD, payload);
}
```

因為`LaunchActivity`是獨立運行的，所以不可以使用`TouchUtils`庫來操作UI。如果要直接進行`Button`點擊，我們可以調用`perfoemClick()`方法。

本節示例代碼[AndroidTestingFun.zip](#)

# 創建功能測試

編寫:[huanglizhuo](#) - 原文:<http://developer.android.com/training/activity-testing/activity-functional-testing.html>

功能測試包括驗證單個應用中的各個組件是否與使用者期望的那樣（與其它組件）協同工作。比如，我們可以創建一個功能測試驗證在用戶執行UI交互時Activity是否正確啓動目標Activity。

要為Activity創建功能測試，我們的測試類應該對ActivityInstrumentationTestCase2進行擴展。

與ActivityUnitTestCase不同，ActivityInstrumentationTestCase2中的測試可以與Android系統通信，發送鍵盤輸入及點擊事件到UI中。

要瞭解一個完整的測試例子可以參考示例應用中的 SenderActivityTest.java。

## 添加測試方法驗證函數的行為

我們的函數測試目標應該包括：

- 驗證UI控制是否正確啓動了目標Activity。
- 驗證目標Activity的表現是否按照發送Activity提供的數據呈現。

我們可以這樣實現測試方法：

```
@MediumTest
public void testSendMessageToReceiverActivity() {
    final Button sendToReceiverButton = (Button)
        mSenderActivity.findViewById(R.id.send_message_button);

    final EditText senderMessageEditText = (EditText)
        mSenderActivity.findViewById(R.id.message_input_edit_text);

    // Set up an ActivityMonitor
    ...

    // Send string input value
    ...

    // Validate that ReceiverActivity is started
    ...

    // Validate that ReceiverActivity has the correct data
    ...

    // Remove the ActivityMonitor
    ...
}
```

測試會等待匹配的Activity啓動，如果超時則會返回null。如果ReceiverActivity啓動了，那麼先前配置的ActivityMonitor就會收到一次碰撞（Hit）。我們可以使用斷言方法驗證ReceiverActivity是否的確啓動了，以及ActivityMonitor記錄的碰撞次數是否按照預想地那樣增加。

## 設立一個ActivityMonitor

為了在應用中監視單個Activity我們可以註冊一個ActivityMonitor。每當一個符合要求的Activity啓動時，系統會通

知ActivityMoniter，進而更新碰撞數目。

通常來說要使用ActivityMoniter，我們可以這樣：

1. 使用getInstrumentation()方法為測試用例實現Instrumentation。
2. 使用Instrumentation的一種addMonitor()方法為當前instrumentation添加一個Instrumentation.ActivityMonitor實例。匹配規則可以通過IntentFilter或者類名字符串。
3. 等待開啟一個Activity。
4. 驗證監視器撞擊次數的增加。
5. 移除監視器。

下面是一個例子：

```
// Set up an ActivityMonitor
ActivityMonitor receiverActivityMonitor =
    getInstrumentation().addMonitor(ReceiverActivity.class.getName(),
        null, false);

// Validate that ReceiverActivity is started
TouchUtils.clickView(this, sendToReceiverButton);
ReceiverActivity receiverActivity = (ReceiverActivity)
    receiverActivityMonitor.waitForActivityWithTimeout(TIMEOUT_IN_MS);
assertNotNull("ReceiverActivity is null", receiverActivity);
assertEquals("Monitor for ReceiverActivity has not been called",
    1, receiverActivityMonitor.getHits());
assertEquals("Activity is of wrong type",
    ReceiverActivity.class, receiverActivity.getClass());

// Remove the ActivityMonitor
getInstrumentation().removeMonitor(receiverActivityMonitor);
```

## 使用Instrumentation發送一個鍵盤輸入

如果Activity有一個EditText，我們可以測試用戶是否可以給EditText對象輸入數值。

通常在ActivityInstrumentationTestCase2中給EditText對象發送串字符，我們可以這樣做：

1. 使用runOnMainSync()方法在一個循環中同步地調用requestFocus()。這樣，我們的UI線程就會在獲得焦點前一直被阻塞。
2. 調用waitForIdleSync()方法等待主線程空閒（也就是說，沒有更多事件需要處理）。
3. 調用sendStringSync()方法給EditText對象發送一個我們輸入的字符串。

比如：

```
// Send string input value
getInstrumentation().runOnMainSync(new Runnable() {
    @Override
    public void run() {
        senderMessageEditText.requestFocus();
    }
});
getInstrumentation().waitForIdleSync();
getInstrumentation().sendStringSync("Hello Android!");
getInstrumentation().waitForIdleSync();
```

本節例子AndroidTestingFun.zip

# Glossary

---

## Activity

---

An activity represents a single screen with a user interface.

- 5.1.1. 使得網絡服務可發現    5.1.2. 使用WiFi建立P2P連接    5.2.1. 連接到網絡    5.2.2. 管理網絡
- 5.2.3. 解析XML數據    5.6.3. 創建Sync Adapter    5.7.2. 建立請求隊列    5.7.3. 創建標準的網絡請求
- 5.7.1. 發送簡單的網絡請求    0. 序言    4.3.3. 展示卡片翻轉動畫    4.3.1. View間漸變
- 4.3.5. 佈局變更動畫    4.3.2. 使用ViewPager實現屏幕側滑    13.2.2. 處理查詢的結果
- 13.2. 使用CursorLoader在後臺加載數據    13.2.1. 使用CursorLoader執行查詢任務
- 13.1.1. 創建IntentService    13.1.3. 報告後臺任務執行狀態    13.1.2. 發送工作任務到IntentService
- 13.3.1. 保持設備的喚醒    1.2.2. 添加Action按鈕    1.2. 添加ActionBar    1.2.4. ActionBar的覆蓋層疊
- 1.2.1. 建立ActionBar    1.2.3. 自定義ActionBar的風格    1.4. 管理Activity的生命週期
- 1.4.2. 暫停與恢復Activity    1.4.4. 重新創建Activity    1.4.1. 啓動與銷燬Activity
- 1.4.3. 停止與重啓Activity    1.6.1. 保存到Preference    1.1.3. 建立簡單的用戶界面
- 1.1.1. 創建Android項目    1.1.4. 啓動其他的Activity    1.5.3. Fragments之間的交互
- 1.5.1. 創建一個Fragment    1.5.2. 建立靈活動態的UI    1.5. 使用Fragment建立動態的UI
- 1. Android入門基礎：從這裏開始    1.7.3. Intent過濾    1.7. 與其他應用的交互
- 1.7.2. 接收Activity返回的結果    1.7.1. Intent的發送    1.3.3. 適配不同的系統版本    16. Android測試程序
- 6.1.3. 使用Intents修改聯繫人信息    6.1.2. 獲取聯繫人詳情    6.1.1. 獲取聯繫人列表
- 2.3.2. 接收其他設備的文件    2.3.1. 發送文件給其他設備    2.2.3. 請求分享一個文件    2.2.2. 分享文件
- 2.1.2. 接收從其他App返回的數據    2.1.1. 為其他App發送簡單的數據    4.1.3. 緩存Bitmap
- 4.1.5. 在UI上顯示Bitmap    4.2.1. 建立OpenGL ES的環境    12.3.1. 處理控制器輸入動作
- 12.1.1. 檢測常用的手勢    12.1.2. 跟蹤手勢移動    12.1.4. 處理多觸摸手勢
- 12.1.6. 管理ViewGroup中的觸摸事件    12.2.4. 處理按鍵動作    12.2.2. 處理輸入法可見性
- 6.2.3. 顯示位置地址    6.2.4. 創建和監視地理圍欄    6.2.1. 獲取最後可知位置    6.2.2. 獲取位置更新
- 11.6.7. 維護兼容性    11.6.6. 自定義動畫    11.6.1. 開始使用Material Design
- 11.6. 創建使用Material Design的應用    11.6.3. 創建Lists與Cards    11.6.4. 定義Shadows與Clipping視圖
- 11.6.2. 使用Material的主題    3.1.1. 控制音量與音頻播放    3.2.1. 簡單的拍照    3.2.2. 簡單的錄像
- 3.3.3. 打印自定義文檔    3.3.2. 打印HTML文檔    14.3.2. 使用include標籤重用layouts
- 14.1. 管理應用的內存    14.6. 避免出現程序無響應ANR    15.3. 為防止SSL漏洞而更新Security
- 15.1. Security Tips    16.1.2. 創建與執行測試用例    16.1.5. 創建功能測試    16.1.3. 測試UI組件
- 16.1.4. 創建單元測試    16.1. 測試你的Activity    16.1.1. 建立測試環境    8.3.3. 使用TV應用進行搜索
- 8.3.2. 使得TV App能夠被搜索    8.2.1. 創建目錄瀏覽器    8.2.3. 創建詳情頁    8.2.4. 顯示正在播放卡片
- 8.6. TV Apps Checklist    8.1.2. 處理TV硬件部分    8.1.3. 創建TV的佈局文件    8.1.1. 創建TV應用的第一步
- 11.4.2. 開發輔助服務    11.3.1. 抽象新的APIs    11.3.2. 代理至新的APIs
- 11.3.3. 使用舊的APIs實現新API的效果    11.3.4. 使用版本敏感的組件    11.2.4. 優化自定義View
- 11.1.3. 實現可適應的UI    11.5.2. 隱藏系統Bar    11.5.4. 全屏沉浸式應用    11.5.3. 隱藏導航Bar
- 10.5.1. 為App內容開啓深度鏈接    10.5. 使得你的App內容可被Google搜索
- 10.1.2. 為多種大小的屏幕進行規劃    10.2.3. 提供向上的導航    10.2.5. 實現向下的導航
- 10.2. 實現高效的導航    10.2.1. 使用Tabs創建Swipe視圖    10.2.2. 創建抽屜導航
- 10.2.4. 提供向後的導航    10.3.1. 建立Notification    10.3.4. 使用BigView風格    10.3. 通知提示用戶
- 10.3.2. 當啓動Activity時保留導航    10.3.5. 顯示Notification進度    10.4. 增加搜索功能
- 10.4.2. 保存並搜索數據    10.4.1. 建立搜索界面    7.2.1. 創建並執行可穿戴應用    7.2. 創建可穿戴的應用
- 7.2.2. 創建自定義的佈局    7.2.3. 添加語音能力    7.4.2. 同步數據單元    7.4.5. 處理數據層的事件
- 7.4. 發送並同步數據    7.4.4. 發送與接收消息    7.1.2. 在Notification中接收語音輸入
- 7.3.4. 創建2D-Picker    7.3.2. 創建Cards    7.3.5. 創建確認界面    7.3.6. 退出全屏的Activity

