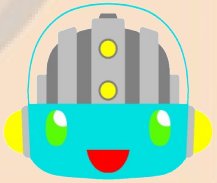
Android Kitkat 入門

robo



自己紹介

- 名前　　robo @cch_robo（兼高理恵）
- お仕事　　Java技術者
  要件定義に設計～実装まで
- 好きなもの　モバイル端末

Android 4.4 KitKat 発表

2013年10月13日、
Androidバージョン4.4がGoogleから発表されました。

Android 4.4 キットカットを大まかまとめれば、
512MB以上のローエンド端末でも滑らかに動作するように
全面改良されたシステムOSです。

キットカット ドロイドくん 画像元
http://www.android.com/versions/kit-kat-4-4/

Android 4.4 KitKat 発表

紹介文では、最も革新的で、最も美しく、最も便利な機能を
何処の国でも、どのAndroid端末にでももたらされるように、
AndroidシステムやAPIに開発ツールとハードウェアを含めた、
広範囲での改良が行われているそうです。

壁紙のフルスクリーンでの
プレビューや
HDR+ モードでの写真撮影、
タップへの応答速度や
タップの精度の向上などは、
まだ Nexus 5 のみ。

このため現時点では、
リファレンス機である Nexus5 でしか試せない機能もあります。

Google Nexus5 画像元 http://www.google.co.jp/nexus/images/nexus-5-new/learn_intro_1200.jpg

Android 4.4 KitKat の新機能紹介先

android サイトでは、
コンシューマ向けの新機能紹介が、

Android Developers では、
開発者向けの新機能紹介ページがあります。

新機能紹介(日本語)　Android - 4.4 (KitKat) | android
http://www.android.com/versions/kit-kat-4-4/

新機能紹介(英語)　Android KitKat | Android Developers
http://developer.android.com/intl/ja/about/versions/kitkat.html#44-hcem/versions/kit-kat-4-4/

Android 4.4 KitKat の新機能紹介先

KitKatが公開されて4ヶ月が経ち、英語資料も翻訳済みですので、詳細については、以下の公開資料を参照されると良いと思います。

Android Developers 新機能紹介(日本語翻訳)
Android(アンドロイド)情報-ブリリアントサービス
Android 4.4 Kitkat詳細（開発者向けハイライト･その1～4）
http://d.hatena.ne.jp/bs-android/20131101/1383296429
http://d.hatena.ne.jp/bs-android/20131101/1383304122
http://d.hatena.ne.jp/bs-android/20131108/1383879993
http://d.hatena.ne.jp/bs-android/20131118/1384743525

IT関係全般 - Firespeed | Android KitKatで何が新しくなったか
http://firespeed.org/diary.php?diary=kenz-1673-junl2

@IT | 低性能端末でも使えるか？ Android 4.4 KitKatの新機能39選 (1～4)
http://www.atmarkit.co.jp/ait/articles/1312/09/news015.html
http://www.atmarkit.co.jp/ait/articles/1312/09/news015_2.html
http://www.atmarkit.co.jp/ait/articles/1312/09/news015_3.html
http://www.atmarkit.co.jp/ait/articles/1312/09/news015_4.html

KitKat 新機能/改良機能 項目名一覧

新機能の詳細内容については、
翻訳済み資料を参照していただくとして、

ここからは、
どれだけ広範囲な改良が行われたのかを
項目名一覧で紹介します。

KitKat 新機能/改良機能 項目名一覧(1)

Making Android for everyone
全ての人にAndroidを

(最初の章だけ意訳)
アンドロイド4.4は、今まで以上に幅広い端末が適用できるよう 512MB以上のRAMで、滑らかで、高速に実行するように設計されています。　革新的で応答性が良くメモリ使用量の低いアプリが作れるよう、新しいAPIとツールが導入されました。

OEMメーカは、メモリ不足へのチューンナップができるよう、Dalvik JITやカーネルを最適化できます。　アンドロイド自体では、メモリ大量消費アプリからシステム･メモリを保護するよう、メモリ管理を改善しメモリフットプリントを削減しました。
開発者には、端末のメモリ構成に合わせてアプリの挙動を調整できるよう、新しいAPI ActivityManager.isLowRamDevice() が提供されました。　低メモリデバイス用にアプリケーションを最適化する方法については、こちらをご覧ください。

新規提供のprocstatsツールは、メモリ使用の詳細分析を行えるようにします。
meminfoツールは、メモリオーバヘッド分析がさらに強化されました。

(意訳ですので、正確な内容でないことに留意ください)

KitKat 新機能/改良機能 項目名一覧(2)

- New NFC capabilities through Host Card Emulation
  ホストカードエミュレーションによる新たなNFC機能

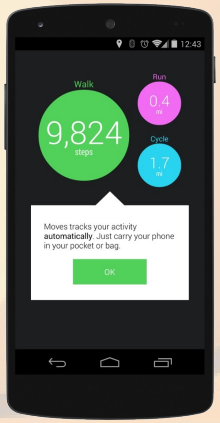
- Printing framework
  印刷フレームワーク

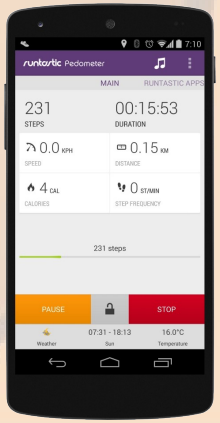
- Storage access framework
  ストレージアクセス･フレームワーク

- Low-power sensors
  低電力消費センサー

  Sensor batching
  センサーのバッチ処理

  Step Detector and Step Counter
  歩行検知と歩数計





KitKat 新機能/改良機能 項目名一覧(3)

- SMS provider
  SMSプロバイダ

- New ways to build beautiful apps
  美しいアプリケーションを構築するための新しい方法

  Full-screen Immersive mode
  フルスクリーン没入モード

  Transitions framework for animating scenes
  アニメーションシーンのための遷移フレームワーク

  Translucent system UI styling
  半透明のシステムUIスタイリング
  (ステータスバーやナビバー背景の透明化)

  Enhanced notification access
  強化されたアクセス通知

  Chromium WebView
  ChromiumベースのWebView

KitKat 新機能/改良機能 項目名一覧(4)

- New media capabilities
  新しいメディア機能

  Screen recording
  スクリーン録画

  Resolution switching through adaptive playback
  adaptive playbackによる解像度の切り替え

  Common Encryption for DASH
  DASH用の共通暗号化

  HTTP Live Streaming
  HTTPライブストリーミング

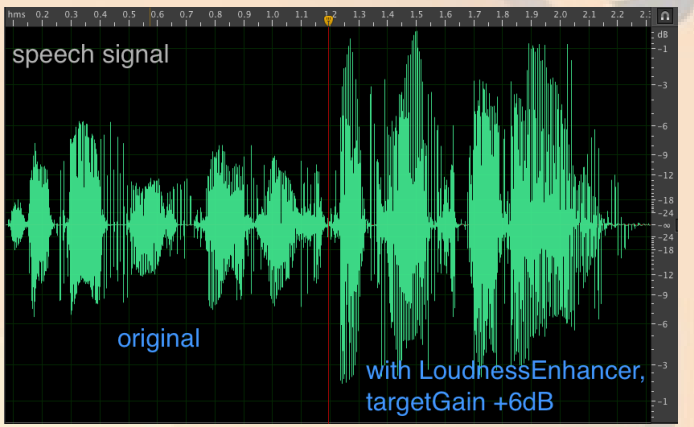
  Audio Tunneling to DSP
  DSPへのオーディオ・トンネリング

KitKat 新機能/改良機能 項目名一覧(5)

- 新しいメディア機能の続き

Audio monitoring
オーディオ・モニタリング

ラウドネス・エンハンサー
Loudness enhancer



Audio timestamps for improved AV sync
改善されたAV同期用のオーディオ・タイムスタンプ

Wi-Fi CERTIFIED Miracast ™
Wi-Fi による公式 Miracast ™ 対応

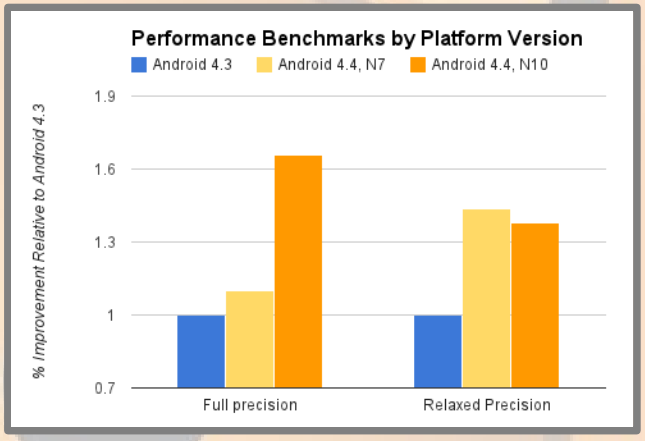
KitKat 新機能/改良機能 項目名一覧(6)

- RenderScript Compute
  レンダースクリプト計算

  Ongoing performance improvements
  継続的なパフォーマンスの向上

  GPU acceleration
  GPUアクセラレーション

  RenderScript in the Android NDK
  Android NDKでのレンダースクリプト



- Graphics
  グラフィック

  GLES 2.0 Surface Flinger
  Surface Flingerの GLES 2.0 アップグレード

  New Hardware Composer support for virtual displays
  仮想ディスプレイのための新しいHardware Composer

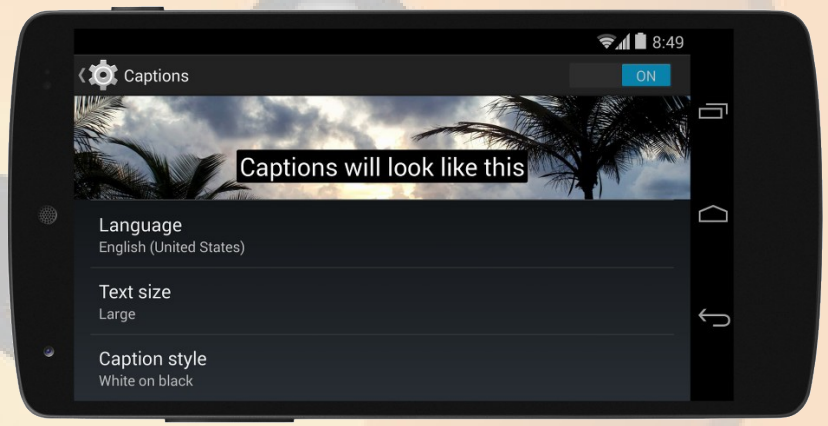
KitKat 新機能/改良機能 項目名一覧(7)

- New Types of Connectivity
  新しいタイプの接続機能

  New Bluetooth profiles
  新しいBluetooth Profile

  IR Blasters
  赤外線ブラスター

  Wi-Fi TDLS support
  Wi-Fi TDLSのサポート



- Accessibility
  アクセシビリティ

  System-wide settings for closed captioning
  クローズドキャプション(解説付字幕)のためのシステム設定

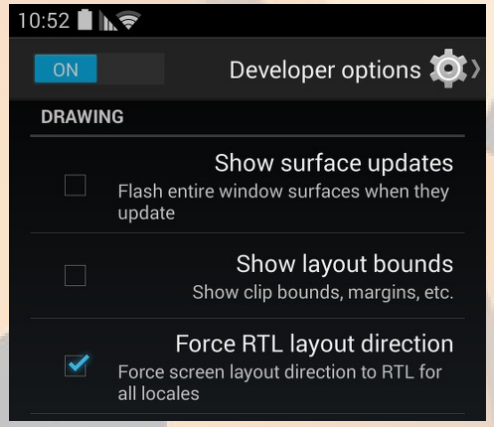
  Enhanced Accessibility APIs
  強化されたアクセシビリティAPI

KitKat 新機能/改良機能 項目名一覧(8)

- Support for international Users
  国際的なユーザーのためのサポート

  Drawable mirroring for RTL locales
  RTLロケール用Drawableのミラーリング

  Force RTL Layout
  強制RTLレイアウト



- Security enhancements
  セキュリティ強化

  SELinux (enforcing mode)
  SELinux(強制モード)

  Improved cryptographic algorithms
  改良された暗号アルゴリズム

  Other enhancements
  その他の機能強化

KitKat 新機能/改良機能 項目名一覧(9)

- Tools for analyzing memory use
  メモリ使用量を分析するためのツール

  Procstats
  Procstats

  (新しいメモリ使用状況解析ツールの名前)

```
            Home: 0.00%
        (Cached): 67% (54MB-56MB-67MB/44MB-47MB-62MB over 32)
* com.google.android.gms / u0a7:
           TOTAL: 28% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 16)
             Top: 27% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 16)
          Imp Fg: 1.2%
         Service: 0.14%
        Receiver: 0.01%
            Home: 0.00%
    (Last Act): 0.09%
      (Cached): 70% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 15)
* com.google.android.apps.books / u0a26:
           TOTAL: 22% (50MB-80MB-113MB/43MB-63MB-82MB over 12)
             Top: 22% (50MB-80MB-113MB/43MB-63MB-82MB over 12)
         Service: 0.00%
    (Last Act): 4.6% (85MB-85MB-85MB/77MB-77MB-77MB over 2)
      (Cached): 58% (78MB-81MB-82MB/70MB-73MB-74MB over 4)
* com.android.settings / 1000:
           TOTAL: 2.8% (14MB-15MB-16MB/11MB-12MB-13MB over 3)
             Top: 2.8% (14MB-15MB-16MB/11MB-12MB-13MB over 3)
```

- On-device memory status and profiling
  端末上のメモリの状態とプロファイリング

…お疲れさまでした。

Android 4.4 - KitKat の新機能や改良は、
項目名だけでも、30以上あります。

新しいAPIやハード対応だけでなく、
開発ツールの追加や改良まであることを
確認されたと思います。

KitKat アップグレードについて

KitKat公開時は、
最初から4.4で出荷された
Nexus5 のみが対応端末でしたが、

2014/03/08現在、他のNexusシリーズにも
Android 4.4 - KitKat への OTA による
システム･アップデートが配信済です。

KitKat アップグレード

ただし、システム・アップデートには、

4.4(Nexus5用)のホームラウンチャアプリ
GoogleHome.apk が含まれていません。

GoogleHome.apk は、最初から4.4で出荷された
Nexus 5 にしかインストールされていないのです。

このため、
他のNexusシリーズでは、システム・アップデートしても、
ステータスバーとナビゲーションバー背景の透明化および、
「Google Now」でマイクをタップせずに、
「OK Google」と発音することで、音声検索できる機能も
使えません…

KitKat アップグレード

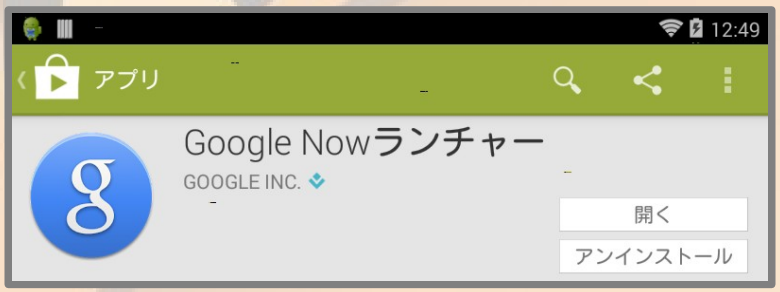
…でしたが、
つい先週ほど(2014/02/26)に、
Google から「GoogleNowランチャー」が、
Google Playで公開されました。

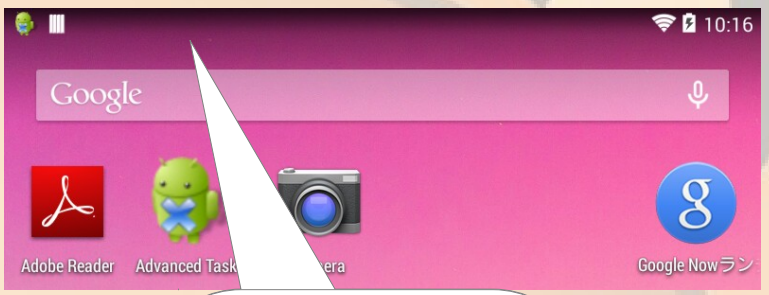
このアプリは、Nexus 5 の
標準ホームラウンチャーに相当するアプリです。

これをインストールして有効にすれば、
Nexus 5 以外の Nexus デバイスと
Google Play Edition デバイスで、
前記の機能が利用できるようになります。

AndroidLover.net | Android4.4の新型Nexus7(2013)で、Nexus5に提供されているホームアプリ「Google Nowランチャー」を使って「OK Google」の音声コマンド機能を使う方法。
http://androidlover.net/tablet/nexus7-2/android4-4-nexus7-2013-nexus5-google-home-ok-google.html

Google Now ランチャー





ただし、ホーム画面のアイコンが少し大きくなることや、
利用できるホーム画面が1面になること、
言語を英語にしないと「OK Google」と発音しても
音声検索が自動起動しない…など注意が必要です。

（補足）新しい仮想マシン ART について

Android 4.4には「ART」(Android RunTime)と呼ばれる
新しい仮想マシンが追加されました。

ARTは、Just-In-Time(JIT)コンパイル形式のDalvikと異なり、
実行前に全コードをネイティブコードにコンパイルしておくことで
実行時のオーバーヘッドを無くし、速度向上と省電力を図ります

ARTは、まだ実験段階だそうですので、ランタイムを Dalvik から ART に
切り替えるには、「開発者オプション」の「ランタイム変更」で、
「ARTを使用する」を選択する必要があるそうです。

(注意)Nexus 7(2012)は、**ART** 切り替えに対応していません。

西川和久の不定期コラム | Google、「Android 4.4」～新仮想マシンARTを搭載
http://pc.watch.impress.co.jp/docs/column/nishikawa/20131128_625412.html

開発者向けオプションについて

Android の設定画面には、
USBデバッグなどをONにすることができる
「開発者向けオプション」というデベロッパー用の
設定項目がありますが、

Jelly Bean (4.1, 4.2, 4.3)からは、
一般の人が利用しないよう、デフォルトでは非表示になり、
利用には、端末の開発者用への切り替えが必要になりました。

Kitkatでも、デフォルトでの非表示を踏襲していますので、
利用するには、端末の開発者用への切り替え操作が必要です。

開発者向けオプションの有効化

「開発者向けオプション」を表示するには、
「ビルド番号」を連続タップし続ける操作が必要です。

連続タップを7回続けると、
「これでデベロッパーになりました!」と表示され
設定画面に「開発者向けオプション」が表示されるようになります。

Nexus5の場合は、
設定 > 端末管理 > 端末情報 に「ビルド番号」があります。

すまほん!! | Androidの「開発者向けオプション」を表示する方法。(＋動作のサクサク化)
http://smhn.info/201311-android-developer-option

駆け足で、新機能の紹介を行いましたが、
Android 4.4 - KitKat が発表されてから
4ヶ月以上が経っていますので、

「そんなの、もう知ってるよ！　（＾３＾）」
と思われた方も多いと思います。

そこで、ちょっと脱線して、
最近の Android 噂話を入れますね。

最近の Android 噂話

Android の世界市場シェアは、約8割!

2013年世界スマートフォン市場、「Android」シェアは8割--IDC調査（2014/02/13）
http://japan.cnet.com/marketers/news/35043836/

Androidの2013年の世界のスマートフォンの市場シェアは79%
- 成長率は62%で過去最低（2014/01/30）
http://jp.techcrunch.com/2014/01/30/20140129android-79-ios-16-wp-4/

IDCから、スマートフォンOS別の市場シェアが発表されました。

Android	78.6%	69.0%
iOS	15.2%	18.7%
Windows Phone	3.3%	2.4%
Black Berry	1.9%	4.5%
Others	1.0%	5.4%
総出荷台数	10.096億	7.253億（左が2013年、右が2012年）

世界市場シェアは8割近く圧倒的ですが、成長率は過去最低となりました。
2014年の成長率はさらに低下すると予測され、市場飽和が懸念されているとあります。

注目？の2端末

続いて Android ではありませんが、2端末を紹介します。

ノキア流Android「Nokia X+」にハンズオン：
限りなく非Android的スマートフォン （2014/02/26）
http://www.gizmodo.jp/2014/02/androidnokia_xandroid.html

Nokia X と Nokia X+ は、Android Open Source Project(AOSP)から UIを ~~魔改造~~大幅カスタマイズした、Windows Phone のような外観の Android と呼べない Android 互換端末。

サムスンがTizen搭載スマートウォッチ Gear2 を4月発売、
カメラ非搭載の Gear 2 Neo も製品化 （2014/02/23）
http://japanese.engadget.com/2014/02/23/tizen-gear2-4-gear-2-neo/

Galaxy Gear は、Samsung が開発した、同社の Android スマートフォン Galaxy シリーズと連携するスマートウォッチです。注目点は、昨年の前機は Android OS 搭載でしたが、最新版 Gear2 では、Tizen OS に変更され、Samsung の戦略変化が見られることです。

Google 副社長さんの2端末への見解

この2端末の登場に対して、

グーグル副社長「サムスンはアンドロイドから離れられない」（2014/02/28）
http://itpro.nikkeibp.co.jp/article/NEWS/20140228/540242/?ST=android-dev&P=1

グーグルのAndroid責任者「次期Nexusスマホは2014年上半期には出しません。Galaxy S6はAndroidを載せます」（2014/03/03）
http://www.gizmodo.jp/2014/03/nexusgalaxy_s6androidandroid.html

Googleの Android／Chrome 担当上級副社長 スンダル・ピチャイ(Sundar Pichai)さんは、MWC2014でのインタビューで、

・Samsung については、Gear2 のOS切り替えは容易でないことや Galaxy S6 は、Android が搭載されること
・Androidでない Android互換端末の Nokia Xや Kindle Fireに対し(Google謹製の)標準 Android こそがベストなプラットフォームだとしたそうです。

Google I/O 2014 は、**6**月**25**日から**2**日間開催予定ですが、今年は、**Google I/O** での新型端末発表がないのかな**... ('･_･`)**

Google と Samsung との距離感

Google のライバルである Nokia や Amazon との関係はもとより、
Google と Samsung との距離感は、別の記事でも指摘されています。

Androidはクローズドソースモデルを目指しているのか？ （2014/02/05）
http://www.infoq.com/jp/news/2014/02/android-closed-source-model

- Googleによる非標準 Android（フラグメント/断片化）撲滅活動が、
  OEM メーカへの均質化強制（端末の横並び化/平凡化圧力）となり、
  OEM メーカの独自性や差別化を否定する閉塞感を与えている。

- Samsung であっても例外でなく、
  Android を乗り越えるため Tizen OS への関与を深めている。

と見られています。

Google と Samsung との距離感

Galaxy Gear だけでなく、
問題は根が深そう…

サムスン、「Android」独自仕様の縮小をグーグルと合意か（2014/01/30）
http://japan.cnet.com/news/service/35043207/

グーグル、モトローラ使ってサムスンに2度の打撃（2014/02/20）
http://www.nikkei.com/article/DGXNASFK19033_Z10C14A2000000/

記事では、Google は、モトローラ買収によって得た特許により、
Samsung に対し、独自のUIやアプリのプリインストールよりも、
Android 標準のUIやアプリを優先させるよう圧力をかけた
と見ています。

スマフォって高級品じゃぁなくなるの？

この背景には、
スマートフォンの急速なコモディティ(一般/均質)化により、
OEM メーカーには勝者なき戦場となっている感覚があるようです。

もはやスマホは急成長しない？
世界出荷台数の伸び、今後数年で大幅鈍化との予測　（2014/03/07）
http://itpro.nikkeibp.co.jp/article/COLUMN/20140304/540842/?k3

IDCによると、2014年は世界スマートフォン市場の成長が鈍化し、
150ドル未満の低価格端末が増え、昨年335ドルの平均販売価格は、
2018年には260ドルにまで低下すると推計されているそうです。

「ゴシップ話は、もういいよう…」と、思いますので、
お口直しに、KitKat 新機能を実際に使ってみましょう。

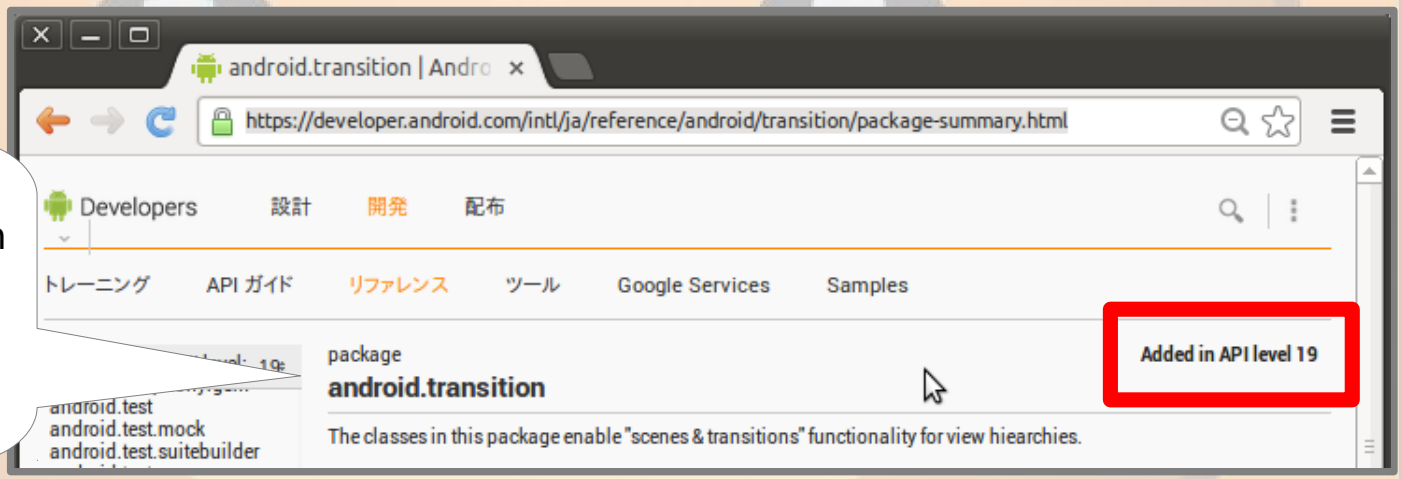
KitKat の新機能を使ってみましょう!

KitKat で追加された新機能は、いくつかありますが。
特殊なハードウェアや SMS などの利用を前提としない、
シーン遷移のアニメーション・フレームワーク
(android.transitionパッケージ)のサンプルを作ってみます。

android.transition パッケージについて

KitKat 4.4 より、
android.transition パッケージが追加され、
シーン遷移によるアニメーション表現(*1)が
可能になりました。

android.transition
パッケージは、
API 19 – 4.4 で
新規追加されました



(*1)以降より、トランジション・アニメーションと表記します。

(※)シーンとはキーとなる表示内容です、詳細は後述します。

トランジション･アニメーションのポイント

トランジション･アニメーションは、
あるレイアウトの表示内容(View内容)の変更前後を比較して、
移動しているViewや、サイズが変更されているViewがあれば、
指定されたDuration(変更期間)中の位置や大きさの
中割動画(中割表示)を自動算定(生成)して表示することで、
アニメーション表現を行っています。

このため、アニメーション対象となるレイアウトとその親、
キーとなる表示内容(キーフレーム)が、ポイントとなります。

トランジション･アニメーションでの役割別･概念名一覧

- SceneParentLayout　シーン対象レイアウトの親レイアウト
- SceneRootLayout　シーン対象レイアウト(シーンのルートビュー)
- SceneViews　シーン対象レイアウトの表示物と内容(子View達)
- Scene　(class)　シーン(場面)
  キーフレーム(変更前後)となるレイアウト設定先
- Transition(class)　アニメーション表現設定先
  Viewの移動やサイズ変更の扱い(ChangeBounds)や
  フェードIN/OUT(Fade)に変更期間(duration)など
  キーフレーム前後の中割算定方法を指定します。
- TransitionSet　(class)　アニメーション表現設定の一括格納先
- TransitionManager　(class)　アニメーション実行の本体

トランジション･アニメーションでは、
キーフレームをシーン(場面)と呼称して android.transition.Scene オブジェクトで取り扱います。
ここでの概念名は、私の説明用の表現です、**Google**さんの想定とは異なることに留意ください。

トランジション･アニメーションでの注意点

中割動画の自動算定(生成)を行うには、
キーフレーム前後で、どのViewが対応するのかの
明示が必要です。

このため、
ParentLayout、RootLayout、SceneViews
(全ての子View)には、View#setId(int 識別番号)で
固有のViewIDを割り当ててください。

トランジション･アニメーションの使用例

```
//次シーン(表示内容)レイアウト設定
ViewGroup sceneRootLayout   = <SceneViews表示内容のルートレイアウト>;
ViewGroup sceneParentLayout = <SceneRootLayoutの親レイアウト>;
Scene scene = new Scene(sceneParentLayout, sceneRootLayout)

//アニメーション表現設定
ChangeBounds changeBounds = new ChangeBounds();
changeBounds.setDuration(3000); //変更期間 3000ミリ秒

//アニメーション表現格納
TransitionSet transitionSet = new TransitionSet();
transitionSet.addTransition(changeBounds);

//トランジション･アニメーション実行
//(※)現在の表示内容から､シーン指定の表示内容に
//     中割りアニメーションさせながら変更します。
TransitionManager.go(scene, transitionSet);
```

サンプル･アニメ構想

トランジション･アニメーションのポイントを掴みましたので
安易ですけれど、KitKat でパワーアップを表現する

1.ドロイド君が合体して

2.キットカットになり

3.巨大化する

アニメーションを作ることにします。

アニメーションのためのキーフレーム作り

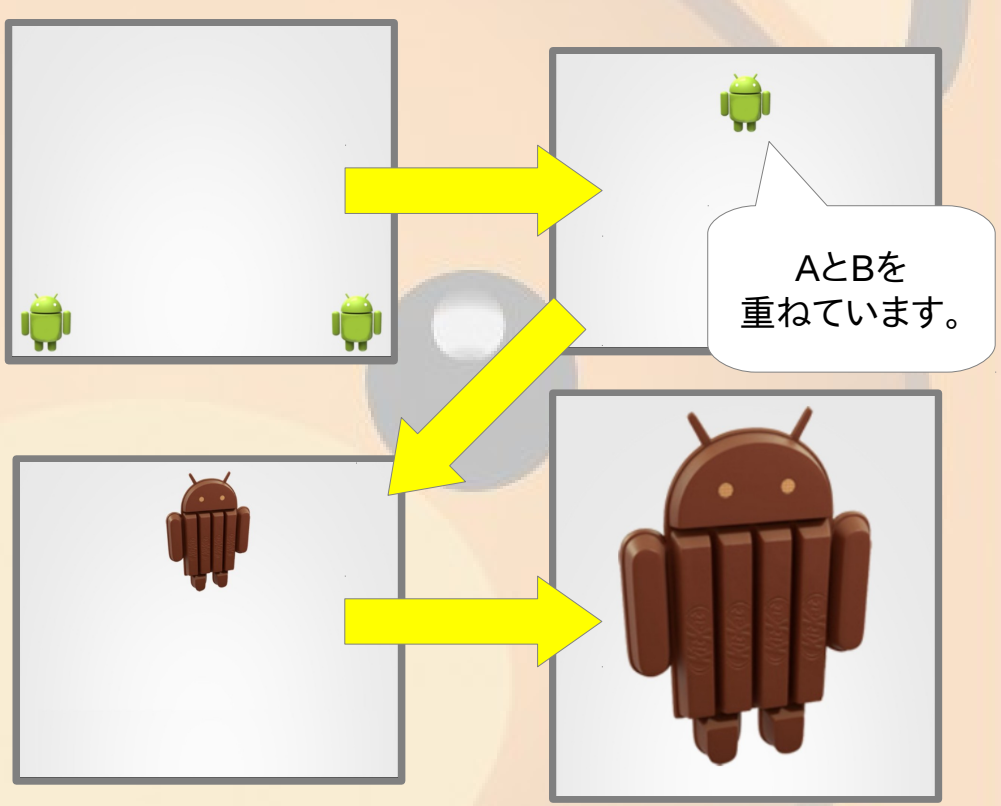
シーン(キーフレーム)となる表示内容は、以下のようになります。

1.ドロイド君(A, B)登場

2.ドロイド君(A, B)合体

3.キットカットに変身

4.キットカット巨大化



Duration(遷移/変更期間**)**は、
1～2:1000ミリ秒、**2～3:0**秒**(**表示置換**)**、**3～4:3000**ミリ秒 とします。

シーン(キーフレーム)のワンポイント

トランジション・アニメーションでは、
前後の表示内容の位置やサイズの変更から、中割りを自動生成します。
このため登場人物?は、変更前後で存在している必要があります。

ドロイド君からキットカットへの変身を表現するため、
画面途中から登場するキットカットは、あらかじめ非表示で登場させておき、
画面途中から居なくなるドロイド君は、途中から非表示にして退場させます。

サンプル実装のソースについて

サンプルのソース内容は、
資料末尾に追加させていただきました。

サンプル･アニメーションの実行結果については、
画面記録を使って録画しています。
（これも KitKat での新機能です）

KitKat の画面記録機能

KitKat では、画面録画機能が追加されました。
180秒までなら画面サイズやビットレートを指定して、
画面操作(やアニメーション)を mp4 形式で端末に記録できます。

画面録画機能は、screenrecord というシェルコマンドのため、
Android SDK 環境が構築されたPCと、KitKat 端末とをUSBで接続し、
adb shell を実行したコンソール上で実行することに注意してください。

コマンド例

```
screenrecord --help                    （コマンド・オプション一覧表示）
screenrecord --time-limit 15 <外部ストレージ>/record.mp4
（外部ストレージにrecord.mp4というファイル名で15秒間だけ記録）
```

塩田紳二のアンドロイドなう |「android 4.4 "Kit Kat"」で画面を録画する
http://news.mynavi.jp/column/androidnow/053/

サンプル･アニメの再生

それでは、mp4 対応の動画プレーヤで、
サンプル･アニメーションを再生してみましょう…

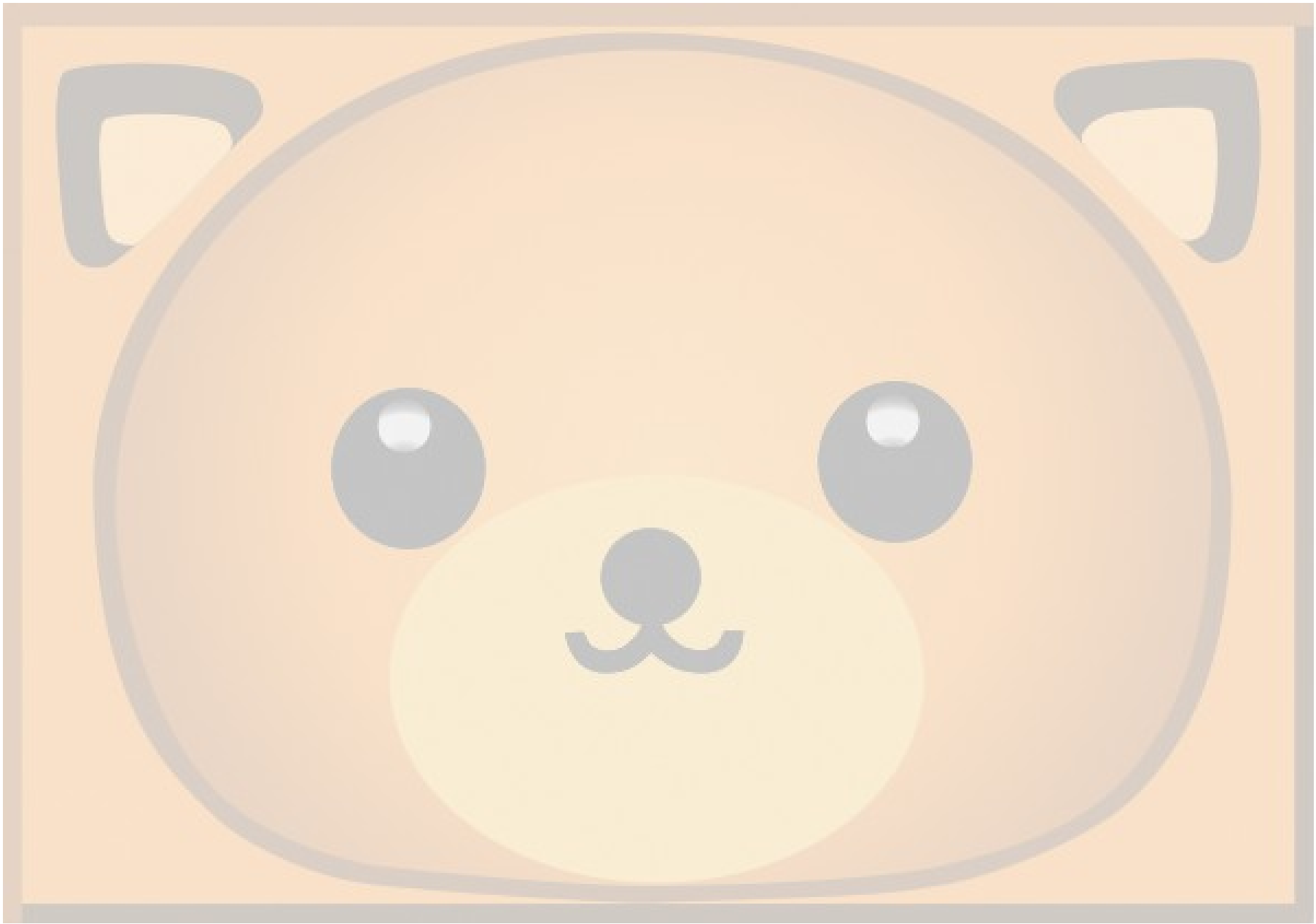
file://./Sample.mp4

画面表示の録画は、
今までルート化しないとできない機能でしたが、
KitKat になって手軽に利用できるようになりました。

KitKat 入門は、いかがでしたか？

Android 4.4 - KitKat は、
色々と手を入れられているのだな...と
思ってくださいましたなら、幸いです。

ご清聴、
ありがとうございました。



わんくま同盟

わんくま同盟 大阪勉強会 #58

サンプル実装･関連資料

以降より、サンプルのソース関連資料を記載いたします。

サンプル実装では、ソース･ファイルが散らばらないよう、
SceneView の表示内容や Transition の設定内容を
コードで直接指定していますので、メンテナンス性などが
低下しています。

実際にアプリなどで利用される場合は、
XML リソースで指定してください。

【補足】Android SDK でのサンプルついて

シーン遷移のアニメーション･サンプルは、Android SDK の APIDemos (API レベル19 - 4.4) にも含まれています。
<Android SDK>/samples/android-19/legacy/ApiDemos

ここでは、
Transition の設定内容を XML で定義していますので
勉強される方は、併せて御参照ください。

Android SDK API Demosでの トランジション･アニメーション サンプル構成

サンプル画面遷移順　：API Demos > Animation > Simple Transitions
サンプル画面タイトル："Animation/Simple Transitions"

ソース・ファイル
・パッケージ：com.example.android.apis.animation
・クラス　　：Transitions.java
・画面全体レイアウト：
  res/layout/transition.xml
  (トランジション対象部分(変更部)のルートビューのレイアウトは、プログラムで差し替えられます)

・遷移部分レイアウト：
  res/layout/transition_scene1.xml
  res/layout/transition_scene2.xml
  res/layout/transition_scene3.xml
  (scene4用のレイアウトは、プログラムで作成)

・遷移設定：
  res/transition/transitions_mgr.xml　　(トランジションの開始レイアウトと終了レイアウトと表現の参照先指定)
  res/transition/changebounds.xml　　(無条件時の表現指定)
  res/transition/changebounds_fadein_together.xml　　(フェードイン時の表現指定)
  res/transition/changebounds_fadeout_sequential.xml　(フェードアウト時の表現指定)

サンプル実装のソースとリソースについて

以降より、サンプル実装のソースを記載いたします。

必要なソースは、MainActivity.java と AndroidManifest.xml と、
doroid.png と android_com_kitkat.png 画像のみとなっています。

画像リソース定義

```
droid.png                (144 x 144 pixel ノーマル･ドロイド君)
android_com_kitkat.png   (179 x 253 pixel キットカット･ドロイド君)
wankuma_icon.png         (144 x 144 pixel わんくまアプリアイコン)
```

(※)画像リソースは、res/drawable に配置しています。



ノーマル･ドロイド君は、IDEのアイコンから、
キットカット･ドロイド君は、以下の画像を利用させていただきました。
http://www.android.com/versions/kit-kat-4-4/

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cch_lab.android.apply.sample.kitkat_transition_animation"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/wankuma_icon"
        android:label="KitKat TransitionAnimation サンプル"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.cch_lab.android.apply.sample.kitkat_transition_animation.MainActivity"
            android:label="KitKat TransitionAnimation サンプル"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

MainActivity.xml (1/14)

```
package com.cch_lab.android.apply.sample.kitkat_transition_animation;

import com.cch_lab.android.apply.sample.kitkat_transition_animation.R;

import android.os.Bundle;
import android.app.Activity;
import android.graphics.Color;
import android.transition.ChangeBounds;
import android.transition.Scene;
import android.transition.TransitionManager;
import android.transition.TransitionSet;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;
```

MainActivity.xml (2/14)

```
public class MainActivity extends Activity implements OnClickListener {

    private LinearLayout   screenLayout;
    private LinearLayout   sceneParentViewGroup;
    private RelativeLayout sceneRootViewGroup;
    private Button         button;

    private Scene          scene1;
    private Scene          scene2;
    private Scene          scene3;
    private Scene          scene4;

    private final int SCREEN_LAYOUT_ID           = 10000;
    private final int SCENE_PARENT_VIEW_GROUP_ID = 10001;
    private final int SCENE_ROOT_VIEW_GROUP_ID   = 10002;
    private final int DROID_A_VIEW_ID            = 10003;
    private final int DROID_B_VIEW_ID            = 10004;
    private final int KITKAT_VIEW_ID             = 10005;
    private final int MESSAGE_VIEW_ID            = 10006;
```

MainActivity.xml (3/14)

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

    //画面のルート・レイアウト
    screenLayout = new LinearLayout(this);
    screenLayout.setId(SCREEN_LAYOUT_ID);
    screenLayout.setOrientation(LinearLayout.VERTICAL);
    screenLayout.setLayoutParams(getScreenLayoutParams());

    //ボタン配置
    button = new Button(this);
    button.setText("バージョンアップ");
    screenLayout.addView(button,
                    new LinearLayout.LayoutParams(
                            ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT));
    button.setOnClickListener(this);

    //シーンルートの親ViewGroup配置
    sceneParentViewGroup =new LinearLayout(this);
    sceneParentViewGroup.setId(SCENE_PARENT_VIEW_GROUP_ID);
    sceneParentViewGroup.setOrientation(LinearLayout.VERTICAL);
    sceneParentViewGroup.setLayoutParams(getSceneParentViewGroupLayoutParams());
    screenLayout.addView(sceneParentViewGroup);
```

MainActivity.xml (4/14)

```
        //シーンのルートViewGroup配置
        //(※)初期表示内容を生成(画面下両脇にドロイド君を配置)
        sceneRootViewGroup = (RelativeLayout)createSceneRootView(1);
        sceneParentViewGroup.addView(sceneRootViewGroup);

        setContentView(screenLayout);
    }

    @Override
    public void onClick(View v) {
        //シーン表示内容1を生成(画面下両脇にドロイド君を配置)
        scene1 = crateScene(sceneParentViewGroup, createSceneRootView(1));

        //シーン表示内容2を生成(ドロイド君が画面中央に移動して合体)
        scene2 = crateScene(sceneParentViewGroup, createSceneRootView(2));

        //シーン表示内容3を生成(キットカットの巨大化)
        scene3 = crateScene(sceneParentViewGroup, createSceneRootView(3));

        //シーン表示内容4を生成 (バージョンアップ完了文字を上に向かってアニメーション)
        scene4 = crateScene(sceneParentViewGroup, createSceneRootView(4));

        //シーンとシーンの間をトランジション･アニメーション表現で表示
        doTransitionAnimation(scene1, 500, 500);
        doTransitionAnimation(scene2, 1000, 1000);
        doTransitionAnimation(scene3, 3000, 2000);
        doTransitionAnimation(scene4, 2000, 5000);
    }
```

MainActivity.xml (5/14)

```
private ViewGroup createSceneRootView(int index){
    final int WRAP_CONTENT = ViewGroup.LayoutParams.WRAP_CONTENT;
    final int MATCH_PARENT = ViewGroup.LayoutParams.MATCH_PARENT;
    RelativeLayout rootViewGroup = null;

～　ここから下の1処理は、記述が長いので一旦分割します ～
```

MainActivity.xml (6/14)

```
//シーン表示内容1(画面下両脇にドロイド君を配置)
if(index == 1){
    //シーンルートビューを新規生成
    rootViewGroup = new RelativeLayout(this);
    rootViewGroup.setId(SCENE_ROOT_VIEW_GROUP_ID);
    rootViewGroup.setLayoutParams(getSceneRootViewGroupLayoutParams());

    //シーン要素View:画面左下にドロイドAを配置
    ImageView droidA = createImageView(DROID_A_VIEW_ID, R.drawable.droid, true);
    rootViewGroup.addView(droidA,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.ALIGN_PARENT_LEFT, RelativeLayout.ALIGN_PARENT_BOTTOM));

    //シーン要素View:画面右下にドロイドBを配置
    ImageView droidB = createImageView(DROID_B_VIEW_ID, R.drawable.droid, true);
    rootViewGroup.addView(droidB,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.ALIGN_PARENT_RIGHT, RelativeLayout.ALIGN_PARENT_BOTTOM));

    //シーン要素View:画面中央にKitkatを配置(非表示)
    ImageView kitkat = createImageView(KITKAT_VIEW_ID, R.drawable.android_com_kitkat, false);
    rootViewGroup.addView(kitkat,
            getChildViewLayoutParams(179, 253,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中下にメッセージを配置(非表示)
    TextView message = createTextView(MESSAGE_VIEW_ID, "バージョンアップ完了", false, 50, Color.RED);
    rootViewGroup.addView(message,
            getChildViewLayoutParams(WRAP_CONTENT, WRAP_CONTENT,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.ALIGN_PARENT_BOTTOM));
}
```

MainActivity.xml (7/14)

```
//シーン表示内容2(ドロイド君が画面中央に移動して合体)
if(index == 2){
    //シーンルートビューを新規生成
    rootViewGroup =new RelativeLayout(this);
    rootViewGroup.setId(SCENE_ROOT_VIEW_GROUP_ID);
    rootViewGroup.setLayoutParams(getSceneRootViewGroupLayoutParams());

    //シーン要素View:画面中央にドロイドAを配置
    ImageView droidA = createImageView(DROID_A_VIEW_ID, R.drawable.droid, true);
    rootViewGroup.addView(droidA,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にドロイドBを配置
    ImageView droidB = createImageView(DROID_B_VIEW_ID, R.drawable.droid, true);
    rootViewGroup.addView(droidB,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にKitkatを配置(非表示)(変更なし)
    ImageView kitkat = createImageView(KITKAT_VIEW_ID, R.drawable.android_com_kitkat, false);
    rootViewGroup.addView(kitkat,
            getChildViewLayoutParams(179, 253,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中下にメッセージを配置(非表示)(変更なし)
    TextView message = createTextView(MESSAGE_VIEW_ID, "バージョンアップ完了", false, 50, Color.RED);
    rootViewGroup.addView(message,
            getChildViewLayoutParams(WRAP_CONTENT,WRAP_CONTENT,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.ALIGN_PARENT_BOTTOM));
}
```

MainActivity.xml (8/14)

```
//シーン表示内容3(キットカットの巨大化)
if(index == 3){
    //シーンルートビューを新規生成
    rootViewGroup =new RelativeLayout(this);
    rootViewGroup.setId(SCENE_ROOT_VIEW_GROUP_ID);
    rootViewGroup.setLayoutParams(getSceneRootViewGroupLayoutParams());

    //シーン要素View:画面中央にドロイドAを配置((非表示)(変更なし)
    ImageView droidA = createImageView(DROID_A_VIEW_ID, R.drawable.droid, false);
    rootViewGroup.addView(droidA,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にドロイドBを配置(非表示)(変更なし)
    ImageView droidB = createImageView(DROID_B_VIEW_ID, R.drawable.droid, false);
    rootViewGroup.addView(droidB,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にKitkatを配置
    ImageView kitkat = createImageView(KITKAT_VIEW_ID, R.drawable.android_com_kitkat, true);
    rootViewGroup.addView(kitkat,
            getChildViewLayoutParams(537, 759,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中下にメッセージを配置(非表示)(変更なし)
    TextView message = createTextView(MESSAGE_VIEW_ID, "バージョンアップ完了", false, 50, Color.RED);
    rootViewGroup.addView(message,
            getChildViewLayoutParams(WRAP_CONTENT, WRAP_CONTENT,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.ALIGN_PARENT_BOTTOM));
}
```

MainActivity.xml (9/14)

```
//シーン表示内容4（バージョンアップ完了文字を上に向かってアニメーション）
if(index == 4){
    //シーンルートビューを新規生成
    rootViewGroup =new RelativeLayout(this);
    rootViewGroup.setId(SCENE_ROOT_VIEW_GROUP_ID);
    rootViewGroup.setLayoutParams(getSceneRootViewGroupLayoutParams());

    //シーン要素View:画面中央にドロイドAを配置（非表示）（変更なし）
    ImageView droidA = createImageView(DROID_A_VIEW_ID, R.drawable.droid, false);
    rootViewGroup.addView(droidA,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にドロイドBを配置（非表示）（変更なし）
    ImageView droidB = createImageView(DROID_B_VIEW_ID, R.drawable.droid, false);
    rootViewGroup.addView(droidB,
            getChildViewLayoutParams(144, 144,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中央にKitkatを配置（変更なし）
    ImageView kitkat = createImageView(KITKAT_VIEW_ID, R.drawable.android_com_kitkat, true);
    rootViewGroup.addView(kitkat,
            getChildViewLayoutParams(537, 759,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.CENTER_VERTICAL));

    //シーン要素View:画面中下にメッセージを配置
    TextView message = createTextView(MESSAGE_VIEW_ID, ”バージョンアップ完了”, true, 50, Color.RED);
    rootViewGroup.addView(message,
            getChildViewLayoutParams(WRAP_CONTENT, WRAP_CONTENT,
                    RelativeLayout.CENTER_HORIZONTAL, RelativeLayout.ALIGN_PARENT_TOP));
}
```

MainActivity.xml (10/14)

```
    return (ViewGroup)rootViewGroup;
}

private Scene crateScene(ViewGroup sceneParentViewGroup, ViewGroup sceneRootViewGroup){
    return new Scene(sceneParentViewGroup, sceneRootViewGroup);
}
```

～　次のメソッドは、記述が長いので一旦分割します ～

MainActivity.xml (11/14)

```
    private void doTransitionAnimation(final Scene scene, long duration, long startDelay){
//      final TransitionManager transitionManager = new TransitionManager();
        final TransitionSet     transitionSet     = new TransitionSet();
        final ChangeBounds      changeBounds      = new ChangeBounds();

        //アニメーション表現設定
        transitionSet.setDuration(duration);
//      transitionSet.setStartDelay(startDelay);
        transitionSet.addTransition(changeBounds);

        //トランジション・アニメーション実行
        //(※)現在の表示内容から、シーン指定の表示内容に
        //    中割りアニメーションさせながら変更します。
        sceneParentViewGroup.postDelayed(
                new Runnable() {
                    @Override
                    public void run() {
                        TransitionManager.go(scene, transitionSet);
                    }
        }, startDelay);

        //(※)Transition#setStartDelay(遅延ミリ秒)は、
        //    TransitionManager.beginDelayedTransition(親ビュー)で、
        //    シーンルートの親レイアウトの現状と遅延ミリ秒後の表示変更を
        //    対象とするようなのでシーン一括生成の場合には、不向き?です。
    }
```

MainActivity.xml (12/14)

```
private LinearLayout.LayoutParams getScreenLayoutParams(){
    LinearLayout.LayoutParams params =
            new LinearLayout.LayoutParams(
                                LinearLayout.LayoutParams.MATCH_PARENT,
                                LinearLayout.LayoutParams.MATCH_PARENT);
    return params;
}

private LinearLayout.LayoutParams getSceneParentViewGroupLayoutParams(){
    LinearLayout.LayoutParams params =
            new LinearLayout.LayoutParams(
                                LinearLayout.LayoutParams.MATCH_PARENT,
                                LinearLayout.LayoutParams.MATCH_PARENT);
    return params;
}

private RelativeLayout.LayoutParams getSceneRootViewGroupLayoutParams(){
    RelativeLayout.LayoutParams params =
            new RelativeLayout.LayoutParams(
                                RelativeLayout.LayoutParams.MATCH_PARENT,
                                RelativeLayout.LayoutParams.MATCH_PARENT);
    return params;
}
```

MainActivity.xml (13/14)

```
private ImageView createImageView(int viewId, int drawableId, boolean isVisible){
    ImageView imageView = new ImageView(this);
    imageView.setId( viewId );
    imageView.setBackgroundResource( drawableId );
    imageView.setVisibility( isVisible ? View.VISIBLE : View.INVISIBLE );
    return imageView;
}

private TextView createTextView(int viewId, String message, boolean isVisible,
                                int textSize, int textColor){
    TextView textView = new TextView(this);
    textView.setId( viewId );
    textView.setText( message );
    textView.setTextSize( textSize );
    textView.setTextColor( textColor );
    textView.setVisibility( isVisible ? View.VISIBLE : View.INVISIBLE );
    return textView;
}

private RelativeLayout.LayoutParams getChildViewLayoutParams(
                    int width, int height,
                    int align_horizontal, int align_vertical){

    RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(width, height);
    params.addRule(align_horizontal);
    params.addRule(align_vertical);
    return params;
}
```

MainActivity.xml (14/14)

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```