



ANDROID MASTERS!

Android Masters!

TechBooster 著

2015-08-24 版 TechBooster 発行

はじめに

Android Masters!を手に取っていただき、ありがとうございます。本書は Design Support Library のすべてを収録しただけでなく Android M に対応した、はじめての技術書です。

執筆にあたって Android M の新機能をいち早く試し、ノウハウとしてまとめました。本文には、サンプルによる実例、考察をふんだんに盛り込んでいます。これらの知見がアプリ開発において参考になれば幸いです。

(発行代表 @mhidaka)

本書の内容について

- マテリアルデザインを実践する Design Support Library
- App Links や Auto Backup、MIDI など Android M の新機能
- Android Studio の NDK 対応とその利用方法

TechBooster とは

TechBooster は Android をはじめとしたモバイルのための技術サークル^{*1}です。オープンソースへの貢献や社会還元を目的にサイトでモバイル技術を解説しています。

お問い合わせ先

本書に関するお問い合わせ <https://plus.google.com/+TechboosterOrg/>

*1 TechBooster の Web サイト <http://techbooster.org/>

目次

| | |
|---|----------|
| はじめに | 1 |
| 本書の内容について | 1 |
| TechBooster とは | 1 |
| お問い合わせ先 | 1 |
| | |
| 第1章 Android Design Support Library | 8 |
| 1.1 Android Design Support Library について | 8 |
| 1.2 導入方法 | 9 |
| 1.3 TabLayout | 9 |
| 1.3.1 TabLayout の表示方法 | 10 |
| 1.3.2 タブが画面に収まりきらない場合の表示方法の設定 | 11 |
| 1.3.3 タブの表示位置の設定 | 13 |
| 1.3.4 ViewPager との連携 | 14 |
| 1.4 NavigationView | 17 |
| 1.4.1 NavigationView の表示方法 | 18 |
| 1.4.2 メニューリソースの作成方法 | 19 |
| 1.4.3 リスト部分の要素のタップの検知 | 24 |
| 1.4.4 リスト部分の要素に選択状態を持たせる | 25 |
| 1.5 TextInputLayout | 28 |
| 1.5.1 TextInputLayout の表示方法 | 29 |
| 1.5.2 エラーメッセージの表示方法 | 29 |
| 1.6 FloatingActionButton | 30 |
| 1.6.1 FloatingActionButton の表示方法 | 30 |

目次

| | | |
|--------------|--|-----------|
| 1.6.2 | FloatingActionButton の大きさ | 31 |
| 1.6.3 | 画像の設定 | 32 |
| 1.7 | Snackbar | 32 |
| 1.7.1 | Snackbar の表示方法 | 34 |
| 1.7.2 | Action の設定 | 34 |
| 1.7.3 | 表示時間の設定 | 36 |
| 1.7.4 | 色の変更 | 38 |
| 1.8 | CoordinatorLayout | 38 |
| 1.8.1 | Anchor | 39 |
| 1.8.2 | Behavior | 41 |
| 1.8.3 | Snackbar + FloatingActionButton | 42 |
| 1.8.4 | 独自 Behavior の設定方法 | 44 |
| 1.8.5 | スクロールに合わせて Floating Action Button を隠す独自 Behavior | 47 |
| 1.9 | AppBarLayout | 52 |
| 1.9.1 | AppBarLayout の表示方法 | 52 |
| 1.9.2 | スクロール時の子 View の挙動の変更 | 53 |
| 1.9.3 | 運動できるスクロール可能な View | 54 |
| 1.9.4 | スクロール可能な View と AppBarLayout の運動 | 54 |
| 1.9.5 | AppBarLayout + TabLayout | 54 |
| 1.10 | CollapsingToolbarLayout | 55 |
| 1.10.1 | CollapsingToolbarLayout を表示する | 56 |
| 1.10.2 | スクロール時の子 View の挙動の変更 | 58 |
| 1.10.3 | タイトルのセット | 59 |
| 1.10.4 | 縮小後の Toolbar の色の変える | 59 |
| 1.11 | 開発時のお役立ち情報 | 60 |
| 1.11.1 | リソースの参照 | 61 |
| 1.11.2 | 参考リンク | 61 |
| 第 2 章 | App Links を使ってアプリを優先的に開く仕組みを学ぶ | 62 |
| 2.1 | App Links がない時代 | 62 |
| 2.2 | App Links がある時代 | 63 |
| 2.3 | App Links の制限 | 64 |

目次

| | | |
|--------------|---|-----------|
| 2.4 | App Links に対応する | 64 |
| 2.4.1 | AndroidManifest.xml の変更 | 64 |
| 2.4.2 | statements.json の設置 | 65 |
| 2.5 | 動作確認 | 66 |
| 2.6 | App Links の解除 | 66 |
| 2.7 | 最後に | 67 |
| 第 3 章 | 最低限知らないと事故る Auto Backup for Apps | 68 |
| 3.1 | 実は以前からあったバックアップの仕組み | 68 |
| 3.2 | 注意しなければならないこと | 68 |
| 3.2.1 | 整合性を失ってしまうファイル | 69 |
| 3.2.2 | AccountManager との関連性 | 69 |
| 3.2.3 | ファイルサイズの制限 | 69 |
| 3.3 | バックアップされるファイル | 69 |
| 3.3.1 | バックアップの対象となるファイル | 69 |
| 3.3.2 | バックアップの対象とならないファイル | 70 |
| 3.4 | バックアップ対象のファイルの追加と除外 | 70 |
| 3.5 | バックアップの検証手順 | 72 |
| 3.5.1 | ログ出力の有効化 | 72 |
| 3.5.2 | バックアップの検証 | 73 |
| 3.5.3 | バックアップからの復元の検証 | 73 |
| 3.5.4 | バックアップの削除 | 73 |
| 3.6 | おわりに | 74 |
| 第 4 章 | M is for MIDI | 75 |
| 4.1 | 導入 | 75 |
| 4.1.1 | 2015 年における MIDI サポートの意味 | 75 |
| 4.1.2 | Android の MIDI サポートに至るまでの状況 | 78 |
| 4.1.3 | JETPlayer | 81 |
| 4.2 | MIDI デバイスを使用する | 83 |
| 4.2.1 | MIDI API の概要 | 83 |

| | | |
|--------------|--|------------|
| 4.2.2 | 利用可能なデバイスの情報を取得する | 84 |
| 4.2.3 | MIDI デバイスと MIDI ポートをオープンする | 85 |
| 4.2.4 | MIDI メッセージを送受信する | 87 |
| 4.2.5 | MIDI アプリケーションのサンプル | 89 |
| 4.3 | MIDI デバイスを作成して公開する | 89 |
| 4.3.1 | 誰がこの API を使うべきか | 90 |
| 4.3.2 | MIDI デバイスの種類 | 90 |
| 4.3.3 | MidiDeviceService クラス | 93 |
| 4.4 | 最後に | 98 |
| 第 5 章 | Android Wear 5.1.1 | 99 |
| 5.1 | Always-on アプリを実装する | 99 |
| 5.1.1 | プロジェクトの設定 | 101 |
| 5.1.2 | アンビエントモードに対応した Activity を作成する | 101 |
| 5.1.3 | アンビエントモードになったとき | 102 |
| 5.1.4 | アンビエントモードから抜けたとき | 103 |
| 5.1.5 | アンビエントモードかどうか確認する | 103 |
| 5.1.6 | 一分毎にアンビエントモードの画面を更新する | 104 |
| 5.2 | Wearable Support Library に追加された View | 104 |
| 5.2.1 | CircularButton | 104 |
| 5.2.2 | ActionLabel | 105 |
| 5.2.3 | ActionPage | 107 |
| 5.2.4 | DotsPageIndicator | 108 |
| 5.2.5 | WearableFrameLayout | 110 |
| 5.3 | 最後に | 112 |
| 第 6 章 | Android Studio で NDK ライフを送ろう | 114 |
| 6.1 | 開発環境の準備 | 114 |
| 6.1.1 | Canary Channel アップデートを行う | 114 |
| 6.1.2 | これは Java8 ですか？ いいえ、これは Java7 です | 116 |
| 6.1.3 | Android NDKr10e のインストール | 116 |
| 6.2 | build.gradle を大胆に、ときに纖細に書き換えよう | 116 |

目次

| | | |
|------------|-----------------------------------|------------|
| 6.3 | デバッグをしようじゃないか | 119 |
| 6.3.1 | 起動直後もデバッグをしようじゃないか | 121 |
| 6.4 | 最後に | 124 |
| 第7章 | Unity 5.1 で入門する VR アプリ開発 | 125 |
| 7.1 | はじめに | 125 |
| 7.1.1 | VR の R (Reality) と本章で扱う範囲について | 125 |
| 7.1.2 | 想定読者について | 125 |
| 7.2 | VR とスマートフォンの関係 | 126 |
| 7.2.1 | 手持ちのスマートフォンで始めよう | 126 |
| 7.2.2 | 具体的な機種とコンテンツの選び方について | 126 |
| 7.2.3 | よりハイエンドなデバイスと、その価格について | 127 |
| 7.3 | スマートフォン向け VR デバイスの紹介 | 128 |
| 7.3.1 | Google Cardboard | 129 |
| 7.3.2 | ハコスコ | 130 |
| 7.3.3 | Oculus Gear VR | 130 |
| 7.4 | VR と 3D とコンテンツ開発手法 | 131 |
| 7.4.1 | VR コンテンツは 3D で、3D なら Unity で | 131 |
| 7.4.2 | Cardboard SDK for Unity のメリット | 131 |
| | つくってみるメリット | 132 |
| 7.5 | VR アプリ開発環境の構築 | 133 |
| 7.5.1 | 手順のおおまかな流れ | 133 |
| 7.5.2 | Android SDK とインストール | 134 |
| 7.5.3 | Unity のインストールとバージョン | 134 |
| 7.5.4 | Cardboard SDK for Unity のダウンロード | 135 |
| 7.6 | Cardboard 向け、VR アプリの Hello, World | 135 |
| 7.6.1 | 新規 Unity プロジェクトの作成 | 135 |
| 7.6.2 | Cardboard SDK for Unity のインポート | 137 |
| 7.6.3 | プレビューの再生 | 139 |
| 7.6.4 | Android プラットフォーム向けの設定 | 140 |
| 7.6.5 | ビルドとインストールと実機での実行 | 143 |
| 7.7 | PONG アプリを Cardboard 向けに対応させる | 144 |

目次

| | | |
|-------|---|------------|
| 7.7.1 | 新規プロジェクトと PONG unitypackage のインポート | 145 |
| 7.7.2 | Android プラットフォーム向けの設定 | 146 |
| 7.7.3 | Cardboard SDK の適用 | 147 |
| 7.7.4 | VR らしい自機制御スクリプト | 148 |
| 7.8 | PONG アプリをハコスコ向けに対応させる | 151 |
| 7.9 | PONG アプリを Gear VR 向けに対応させる | 152 |
| 7.9.1 | 新規プロジェクトと Virtual Reality Supported チェックボックス . | 152 |
| 7.9.2 | Osig ファイル (Oculus Signature File) の準備 | 154 |
| 7.9.3 | Gear VR Service 開発者モードの有効化 | 156 |
| 7.10 | おわりに | 158 |
| | 著者紹介 | 160 |

第1章

Android Design Support Library

この章では Android Design Support Library の使い方について、各コンポーネントごとによく使うであろう機能を紹介します。基本的にレイアウトリソースを用いて各機能を紹介しますが、レイアウトリソースからは使うことのできない機能を紹介する際には Java のコードを用いて紹介します。

1.1 Android Design Support Library について

Android Design Support Library はこれまで提供されていなかった、Floating Action Button などの Material Design のコンポーネントを提供してくれるライブラリです。これまで Material Design に対応したアプリを作成しようと用意されていないコンポーネントが多く、独自に実装を行うか、サードパーティ製のライブラリを使う必要がありました。どちらの方法を採用しても、アプリごとに細かい見た目や挙動が異なってしまう問題や、今後の Support Library の更新により使えなくなってしまうリスクが有ります。また、独自実装やサードパーティ製ライブラリの選定にも時間が掛かります。これらの理由から中々 Material Design の導入に踏み切れない方もいたのではないでしょうか。このライブラリの登場により、前述の問題を気にせず、より気軽に Material Design 対応のアプリを作成することが可能になりました。

Android Design Support Library は次で示すコンポーネントで構成されています。

- TabLayout
- NavigationView
- FloatingActionButton
- Snackbar
- CoordinatorLayout
- AppBarLayout

- CollapsingToolbarLayout

それでは導入方法と各コンポーネントの使い方を見ていきましょう。

1.2 導入方法

build.gradle をリスト 1.1 のように変更します。

リスト 1.1: build.gradle を変更する

```
...
dependencies {
    ...
    compile 'com.android.support:design:22.2.1'
    ...
}
```

Android Design Support Library を読み込ませるだけですが、すでに Support Library を使用している既存のプロジェクトに Android Design Support Library を追加する場合は Support Library のバージョンを Android Design Support Library のバージョンに合わせる必要があります。

1.3 TabLayout

API Level 21 で、ActionBar の Navigation Mode が非推奨 (deprecated) になったため、従来の方法でタブを実装することができなくなりました。TabLayout はその代わりとなるコンポーネントです。Android Developers 内ではサンプルアプリ^{*1}を通してタブを実現するための代替手段が提案されていましたが、今後は TabLayout を使うとよいでしょう。TabLayout は ViewPager 連携やタブの横スクロールなどを簡単に実装できます。

この節では Tab の表示設定の変更方法と ViewPager との連携方法について説明します。

*1 <http://developer.android.com/samples/SlidingTabsBasic/index.html>

1.3.1 TabLayout の表示方法

TabLayout のようなコンポーネントは android.support.design.widget パッケージに含まれます。通常のコンポーネントと同様、XML と Java のソースコードの両方から利用できます。レイアウトリソースから参照する場合は、次のとおりです（リスト 1.2）。

リスト 1.2: レイアウトリソースに TabLayout を追加

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

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"/>
    <android.support.design.widget.TabLayout
        android:id="@+id/tab_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

タブは Toolbar の下に置くことが多いと思いますので、今回の例でも Toolbar の下に TabLayout を配置しています。

次に Java のソースコード上で TabLayout にタブを追加します（リスト 1.3）。

リスト 1.3: タブを TabLayout に追加

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("tab 2")
    .setIcon(R.drawable.tab));
tabLayout.addTab(tabLayout.newTab().setCustomView(R.layout.tab));
```

タブにはテキストとアイコンの他に、レイアウトファイルをセットすることができます。極

端な例として Button のみのレイアウトファイルをセットすると図 1.1 のようになります。

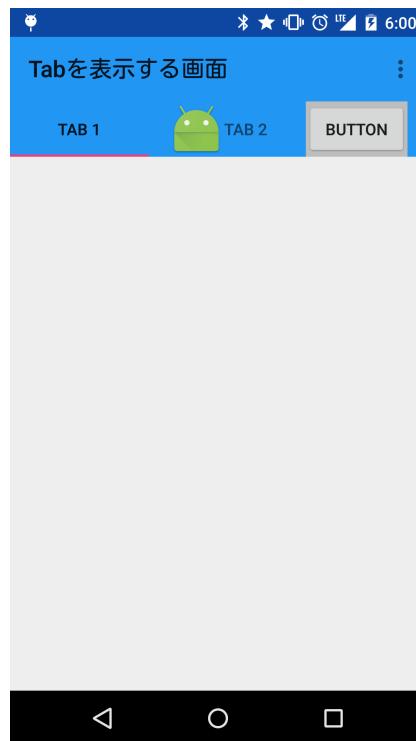


図 1.1 リスト 1.3 の表示結果

1.3.2 タブが画面に収まりきらない場合の表示方法の設定

タブの数が多くなり画面にすべてのタブが収まりきらない場合の表示方法を TabMode として指定することができます。用意されている TabMode は表 1.1 の 2 種です（図 1.2）。

表 1.1 TabMode の種類

| 値 | 説明 |
|---------------------------|------------|
| TabLayout.MODE_SCROLLABLE | スクロール可能な表示 |
| TabLayout.MODE_FIXED | 1 画面に収まる表示 |

TabMode で TabLayout.MODE_SCROLLABLE を指定するとタブの数が多く画面に収まりきらない場合に横スクロールできるようになり、タブのサイズを維持したまま表示します。タブの表示が省略されないためタブの識別に優れています。TabMode で TabLayout.MODE_FIXED を指定するとタブの数が多く画面に収まりきらない場合でもテキストの

第1章 Android Design Support Library

省略、アイコンの縮小などの方法ですべてのタブを画面内に表示します。文字だけのタブを使用する場合は文字が省略されタブの識別が難しいです。TabLayout.MODE_FIXEDを使う場合はアイコンを付けるなど省略されてもタブの識別を容易にする工夫が必要となります。

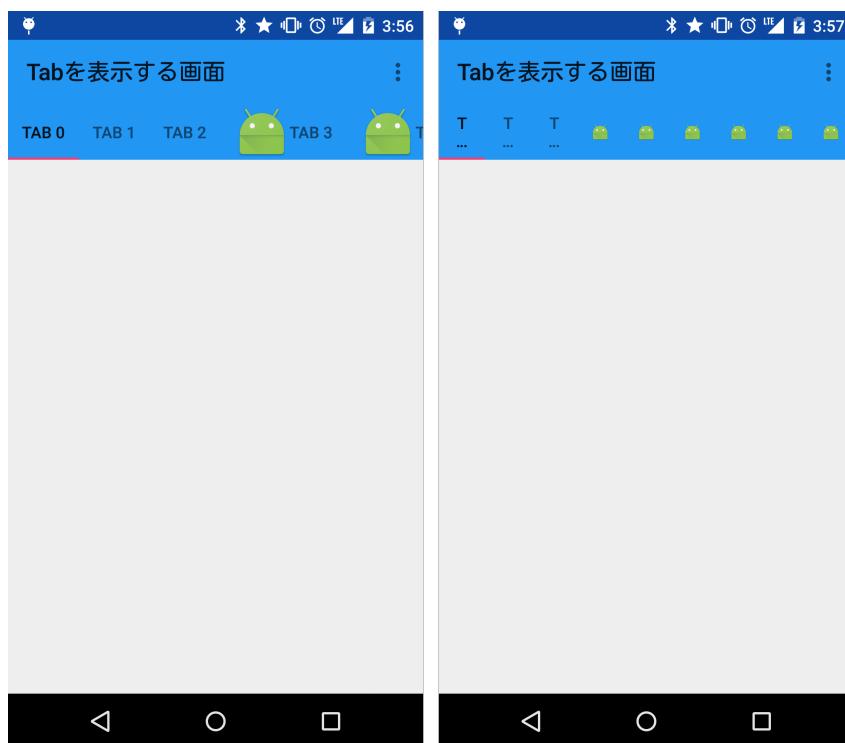


図 1.2 MODE_SCROLLABLE (左) と MODE_FIXED (右)

TabMode を指定しない場合、Tablayout.MODE_FIXED が初期値として扱われます。
TabMode の指定には setTabMode メソッドを使用します（リスト 1.4）。

リスト 1.4: setTabMode メソッドで TabMode を指定

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);  
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE);
```

findViewById で TabLayout のインスタンスを取得し、setTabMode メソッドで TabMode を指定してください。setTabMode メソッドを使用するタイミングは TabLayout にタブを追加する前後のどちらでも構いません。

1.3.3 タブの表示位置の設定

TabMode はタブの数が多くなり画面にすべてのタブが収まりきらない場合の表示方法を指定するためのパラメータでしたが、TabGravity では画面の大きさに対してタブの数が少なく表示に余裕がある場合の表示方法を指定できます。用意されている Gravity は表 1.2 の 2 種です（図 1.3）。

表 1.2 TabLayout の TabGravity

| 値 | 説明 |
|--------------------------|---------|
| TabLayout.GRAVITY_FILL | タブの均等配置 |
| TabLayout.GRAVITY_CENTER | タブの中央寄せ |

TabLayout.GRAVITY_FILL を指定するとタブの大きさがすべて均等になり、TabLayout の横幅すべてを使ってタブを均等に配置します。TabLayout.GRAVITY_CENTER を指定するとタブの大きさはタブのコンテンツ（文字や画像）に合わせたものとなり、TabLayout の中央に配置されます。TabGravity を指定しない場合は TabLayout.GRAVITY_FILL を初期値として扱います。

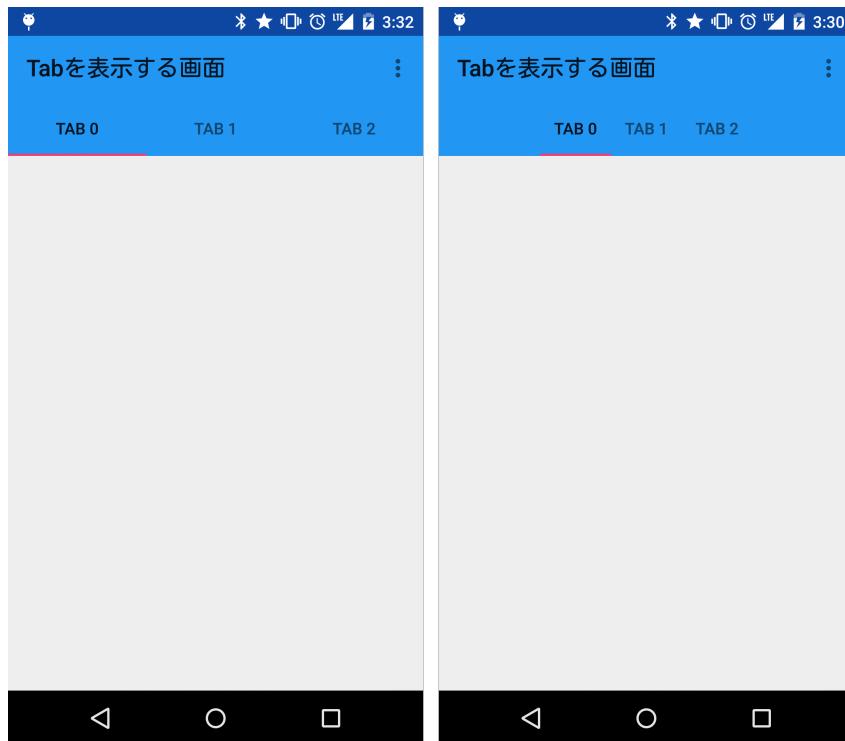


図 1.3 GRAVITY_FILL (左) と GRAVITY_CENTER (右)

TabGravity の指定には setTabGravity メソッドを使用します（リスト 1.5）。

リスト 1.5: setTabGravity メソッドで TabGravity を指定

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);  
tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER);
```

findViewById で TabLayout のインスタンスを取得し、setTabGravity メソッドで TabGravity を指定してください。setTabGravity メソッドを使用するタイミングは TabLayout にタブを追加する前後のどちらでも構いません。また、TabMode で TabLayout.MODE_SCROLLABLE を指定している場合は Gravity で何を指定しても左寄せになってしまいますため注意が必要です。

1.3.4 ViewPager との連携

TabLayout は、もちろん ViewPager と連携することもできます。ViewPager と連携するときは addTab メソッドでタブをセットする必要はありません。なぜなら ViewPager と連携すると addTab メソッドで追加したタブはすべて無視されてしまうためです。

第1章 Android Design Support Library

まず ViewPager を使えるように build.gradle を変更します（リスト 1.6）。

リスト 1.6: build.gradle の変更

```
...
dependencies {
    ...
    compile 'com.android.support:support-v4:22.2.1'
    ...
}
```

次にレイアウトリソースに ViewPager を加えましょう（リスト 1.7）。

リスト 1.7: ViewPager をレイアウトリソースに追加

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

    <android.support.design.widget.TabLayout
        android:id="@+id/tab_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <android.support.v4.view.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

ViewPager を表示したい場所に配置してください。Android ではタブの下にコンテンツを表示することが多いと思いますので、リスト 1.7 では TabLayout の下に ViewPager を配置しています。

次に ViewPager で使用する Adapter を実装します（リスト 1.8）。

リスト 1.8: ViewPagerAdapter の実装

```
public class ViewPagerAdapter extends FragmentPagerAdapter {
```

```
public ViewPagerAdapter(FragmentManager fm) {
    super(fm);
}

@Override public Fragment getItem(int position) {
    return TabFragment.newInstance(position);
}

@Override public int getCount() {
    return 3;
}

@Override public CharSequence getPageTitle(int position) {
    return "Tab " + position;
}
}
```

ViewPagerAdapter は PagerAdapter を継承している必要があります。今回は PagerAdapter のサブクラスである FragmentPagerAdapter を継承しています。getCount でタブの数、getPageTitle で Tab のタイトルを返します。getItem では表示する Fragment を返してください。

最後に Activity での実装です（リスト 1.9）。

リスト 1.9: ViewPager と TabLayout の連携

```
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tab);
    TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
    ViewPager viewPager = (ViewPager) findViewById(R.id.view_pager);
    ViewPagerAdapter adapter =
        new ViewPagerAdapter(getSupportFragmentManager());
    viewPager.setAdapter(adapter);
    tabLayout.setupWithViewPager(mViewPager);
}
```

ViewPager に Adapter をセットしたあと、setupWithViewPager メソッドで TabLayout に ViewPager をセットします。これでタブと ViewPager の連携は完了です。

1.4 NavigationView

NavigationView は Navigation Drawer のドロワー部分を簡単に作成するためのコンポーネントです。図 1.4 のようにヘッダ部分とリスト部分を持っている Navigation Drawer を簡単に作成できます。

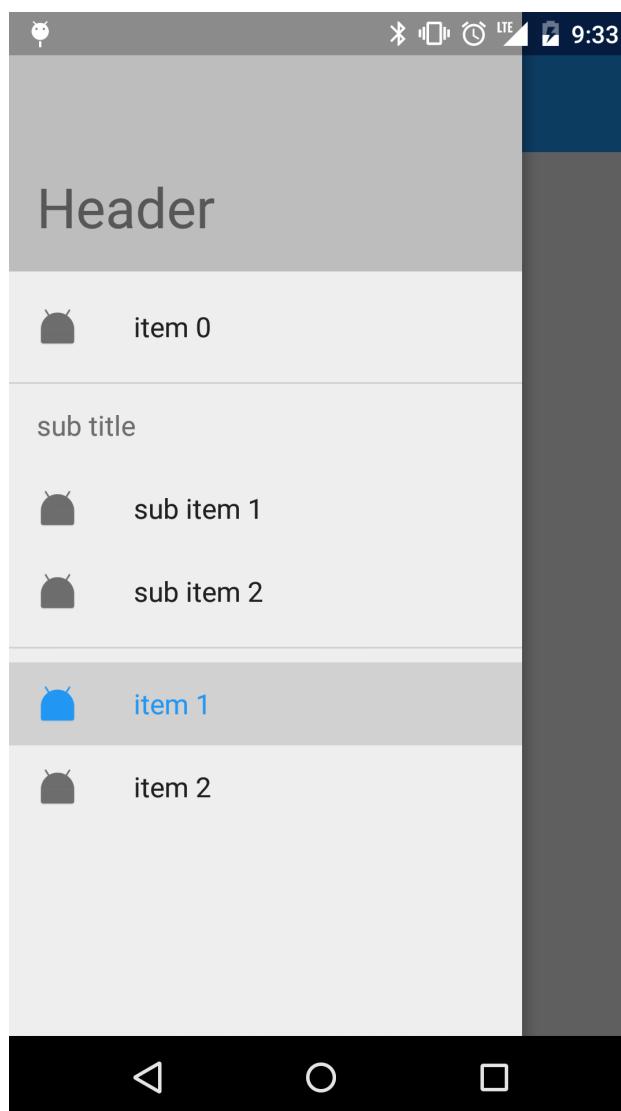


図 1.4 NavigationView を使った Navigation Drawer

大きく 3 つの特徴があります。

- ・ヘッダ部分をレイアウトリソースで指定
- ・リスト部分をメニューリソースから作成できる

- Status bar にドロワー部分を重ねる処理を行う (API Level 21 から)

この節では NavigationView の表示方法と、メニューリソースの作成方法について説明します。

1.4.1 NavigationView の表示方法

NavigationView は DrawerLayout のドロワー部分に使用することを前提としています。そのため、Navigation Drawer を実装するためには今までどおり DrawerLayout が必要です。DrawerLayout に NavigationView を子 View として追加するとレイアウトリソースはリスト 1.10 のようになります。

リスト 1.10: DrawerLayout に NavigationView を追加

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <!-- your content layout -->

    <android.support.design.widget.NavigationView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/drawer_header"
        app:menu="@menu/drawer"/>
</android.support.v4.widget.DrawerLayout>
```

DrawerLayout ではひとつめの子 View (リスト 1.10 では<!-- your content layout -->としています) がコンテンツ部分に、ふたつめの子 View がドロワー部分として扱われますので、NavigationView はふたつめの子 View としてレイアウトリソースに記入してください。NavigationView のヘッダ部分とメニュー部分はそれぞれ別の方で作成します。ヘッダ部分は app:headerLayout 属性でレイアウトリソースを指定し、リスト部分は app:menu 属性でメニューリソースを指定します。

1.4.2 メニューリソースの作成方法

まずふたつの要素だけを表示する、シンプルなリスト部分をメニューリソースで作成してみましょう（図1.5）。

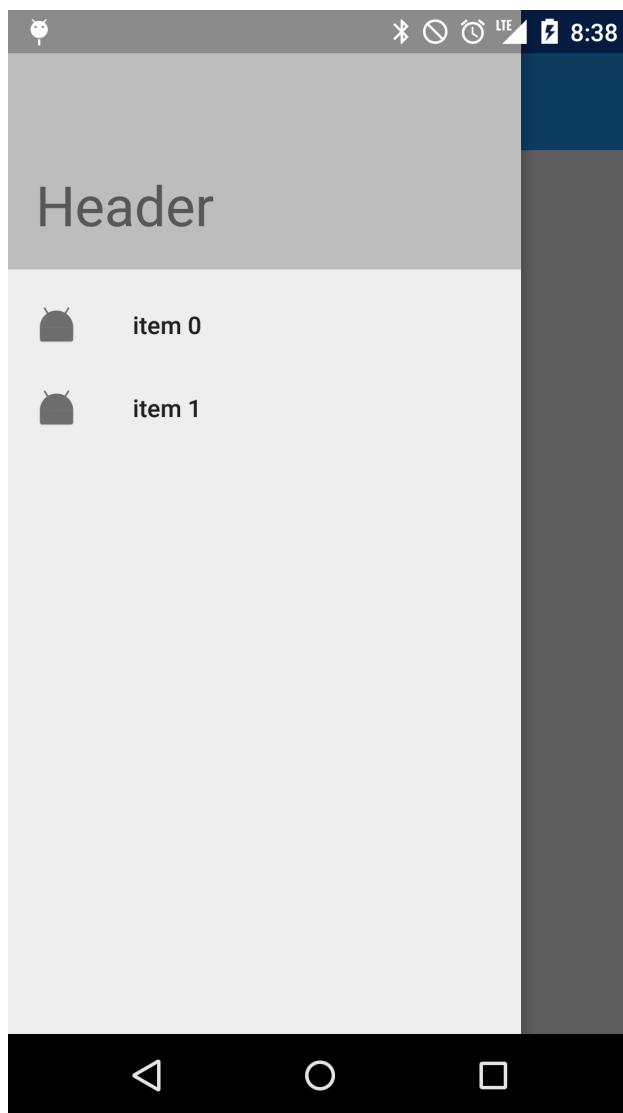


図1.5 ふたつの要素だけ表示するリスト部分の例

menuタグにitemタグを入れ子にするとitemタグの情報をリスト部分の要素として表示できます（リスト1.11）。

リスト1.11: シンプルなメニューリソースの作成

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_item_0"
        android:icon="@drawable/navigation_item_0"
        android:title="item 0"/>
    <item
        android:id="@+id/navigation_item_1"
        android:icon="@drawable/navigation_item_1"
        android:title="item 1"/>
</menu>
```

item タグには、他の要素や Java のソースコードから参照するための id を android:id 属性に、表示するアイコンを android:icon 属性に、表示する文字列を android:title 属性にそれぞれ指定します。

また NavigationView は、サブヘッダを表示することもできます（図 1.6）。

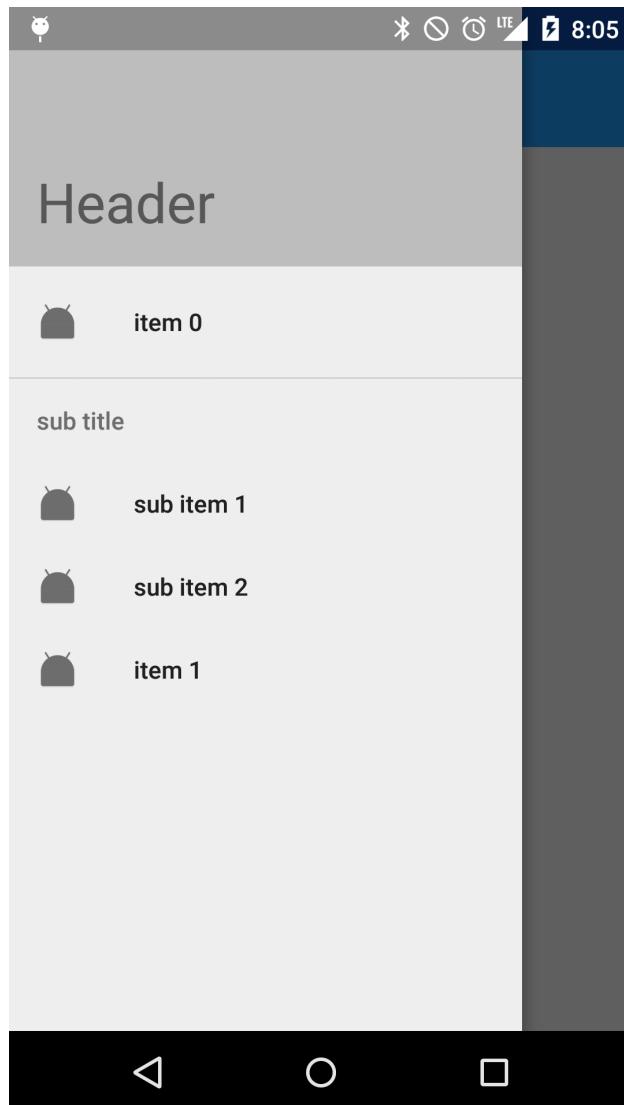


図 1.6 サブヘッダを表示するリスト部分の例

サブヘッダにしたい item タグで menu タグを入れ子にします（リスト 1.12）。

リスト 1.12: サブヘッダの表示

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_item_0"
        android:icon="@drawable/navigation_item_0"
        android:title="item 0"/>
    <item
        android:title="sub title">
        <menu>
            <item
                android:id="@+id/navigation_sub_item_0"
```

```
    android:icon="@drawable/navigation_item_1"
    android:title="sub item 0"/>
<item
    android:id="@+id/navigation_sub_item_1"
    android:icon="@drawable/navigation_item_2"
    android:title="sub item 1"/>
</menu>
</item>
<item
    android:id="@+id/navigation_item_1"
    android:icon="@drawable/navigation_item_1"
    android:title="item 1"/>
</menu>
```

item タグの中に menu タグを入れ子にすると親になった item タグがサブヘッダとして表示されます。リスト 1.12 のサンプルでは android:title 属性に sub title を指定している item タグがサブヘッダです。サブヘッダはタップすることができません。また、今回のサンプルではサブヘッダとその他の要素の境界がわかりやすくなるように、サブヘッダとなる item タグの上下に通常の item タグを置いています。図 1.6 を見るとサブヘッダの上部にはセパレータとなる横線が表示されていますが、下部には表示されていないことがわかります。

サブヘッダを表示せずに横線を表示することも可能ですが（図 1.7）。

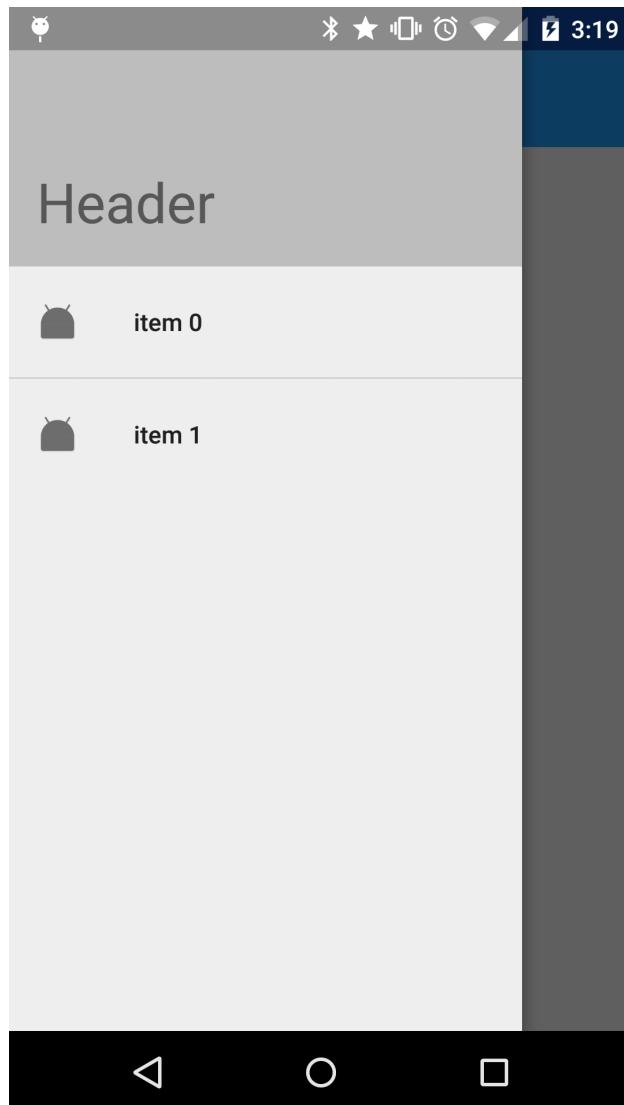


図 1.7 サブヘッダなしのセパレータ表示

サブヘッダを表示せずに横線を表示するには、group タグを使用します（リスト 1.13）。

リスト 1.13: group タグを使い、セパレータを表示する

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_item_0"
        android:icon="@drawable/navigation_item_0"
        android:title="item 0"/>
    <group
        android:id="@+id/group_1">
        <item
            android:id="@+id/navigation_item_1"
            android:icon="@drawable/navigation_item_1"
```

```
    android:title="item 1"/>
  </group>
</menu>
```

group タグで item タグを入れ子にし、group タグの android:id 属性で ID を指定すると、グループの上部に横線が表示されます。group タグを使用しても、android:id 属性で ID を指定しない場合は横線は表示されません。

1.4.3 リスト部分の要素のタップの検知

前項では NavigationView のリスト部分をメニューリソースで作成する方法を説明しました。この項では前項で説明したリスト部分の要素を、ユーザがタップしたときのイベントを取得する方法を紹介します。まず前項で紹介した方法でメニューリソースでリスト部分を作成します。次に NavigationView クラスの setNavigationItemSelectedListener メソッドで Listener をセットします（リスト 1.14）。

リスト 1.14: リストアイテムに Listener をセット

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

  NavigationView navigationView =
    (NavigationView) findViewById(R.id.navigation_view);
  navigationView.setNavigationItemSelectedListener(
    new NavigationView.OnNavigationItemSelectedListener() {
      @Override public boolean onNavigationItemSelected(MenuItem menuItem) {
        switch (menuItem.getItemId()) {
          case R.id.navigation_item_1:
            // TODO: タップ時のアクション
            return true;
          case R.id.navigation_item_2:
            // TODO: タップ時のアクション
            return true;
        }
        return false;
      }
    });
}
```

ユーザがリストの要素をタップすると `setNavigationItemSelectedListener` メソッドでセットした `OnNavigationItemSelectedListener` クラスの `onNavigationItemSelected` メソッドが呼び出され、引数として `MenuItem` クラスのインスタンスを取得できます。`MenuItem` クラスの `getItemId` メソッドでメニューリソースで設定した ID を取得できるため、どの要素がタップされたのか判断することができます。リスト 1.14 では `getItemId` メソッドで取得した ID をもとに `switch-case` 文でタップ時のアクションを要素ごとに分岐させています。

1.4.4 リスト部分の要素に選択状態を持たせる

リスト部分の要素に選択状態を持たせることも可能です（図 1.8）。

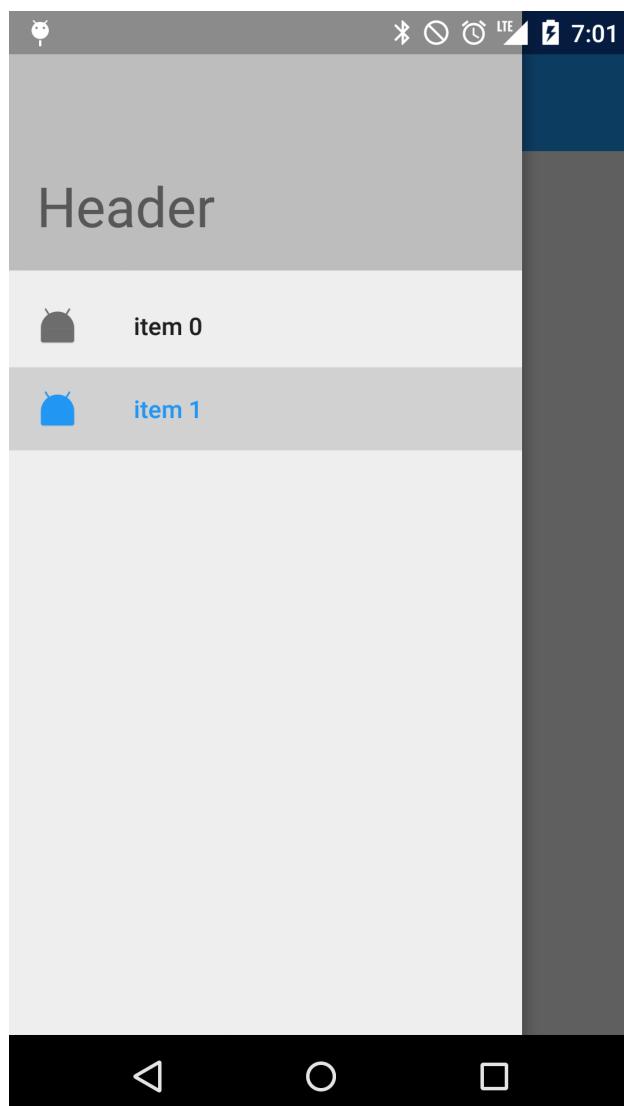


図 1.8 選択状態をもつリスト部分の要素の例

メニュー リソースを `menu` タグ、`group` タグ、`item` タグの順で入れ子にします（リスト 1.15）。

リスト 1.15: 選択状態を管理するメニュー リソースの作成

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/navigation_item_0"
            android:icon="@drawable/navigation_item_0"
            android:title="item 0"/>
        <item
            android:id="@+id/navigation_item_1"
            android:icon="@drawable/navigation_item_1"
            android:title="item 1"/>
    </group>
</menu>
```

```
    android:checked="true"
    android:title="item 1"/>
  </group>
</menu>
```

group タグで item タグを入れ子にし、group タグの android:checkableBehavior 属性で single を指定すると同 group タグ内でひとつの要素のみに選択状態を持たせることができます。single だけでなく複数同時選択できる all、選択状態を持たせない none もありますが、NavigationView の用途で複数アイテムを選択した状態にしたいことはまず無いと思いますので、実質 single か none(android:checkableBehavior を指定しない) のどちらかを使用することになるでしょう。初期状態で要素を選択状態にするには item タグの android:checked 属性を true にします。

また、Java のコードからも選択状態にすることができます。ユーザがリスト部分の要素をタップしても、自動でリスト部分の要素に選択状態が付与されるわけではありません。タップされたときに選択状態にするには、Java のコードからタップされたイベントを取得して選択状態にする必要があります（リスト 1.16）。

リスト 1.16: タップ時にリスト部分の要素を選択状態にする

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

  NavigationView navigationView =
    (NavigationView) findViewById(R.id.navigation_view);
  navigationView.setNavigationItemSelectedListener(
    new NavigationView.OnNavigationItemSelectedListener() {
      @Override public boolean onNavigationItemSelected(MenuItem menuItem) {
        switch (menuItem.getItemId()) {
          case R.id.navigation_item_1:
            menuItem.setChecked(true);
            return true;
          case R.id.navigation_item_2:
            menuItem.setChecked(true);
            return true;
        }
        return false;
      }
    });
}
```

```
});  
}
```

前項で紹介した方法でリスト部分の要素のタップを検知し、選択状態を指定します。MenuItem クラスの setChecked メソッドに true を指定すると選択状態に、false を指定すると未選択状態になります。メニューリソースの group タグの android:checkableBehavior 属性で single を指定している場合はひとつの要素を選択状態にすると他の要素は未選択状態になるため、他の要素を未選択状態に指定するコードは不要です。

1.5 TextInputLayout

TextInputLayout は Material Design のガイドラインの Floating labels^{*2} を簡単に実装できるコンポーネントです。EditText にフォーカスが当たると EditText の hint がアニメーション付きで EditText の上部へ移動しラベルとして表示されます。また、EditText の下部にエラーメッセージを表示することも可能です（図 1.9）。

この節では TextInputLayout の基本的な使い方と、エラーメッセージの表示方法について説明します。



図 1.9 フォーカス時の TextInputLayout（上）と通常時の TextInputLayout（下）

*2 <https://www.google.com/design/spec/components/text-fields.html#text-fields-single-line-text-field>

1.5.1 TextInputLayout の表示方法

TextInputLayout は EditText の親 View として使うことを前提としています。よってレイアウトリソース上では、TextInputLayout と EditText を入れ子にして使用します（リスト 1.17）。

リスト 1.17: レイアウトリソース上での基本的な TextInputLayout の構成

```
<android.support.design.widget.TextInputLayout  
    android:id="@+id/text_input_layout"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
  
    <EditText  
        android:id="@+id/edit_text"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:hint="@string/edit_text_label" />  
  
</android.support.design.widget.TextInputLayout>
```

ラベルとなるテキストは TextInputLayout に設定するのではなく、EditText に hint として与える必要がある点に注意が必要です。

1.5.2 エラーメッセージの表示方法

また、TextInputLayout にはエラーメッセージを表示することもできます（図 1.10）。

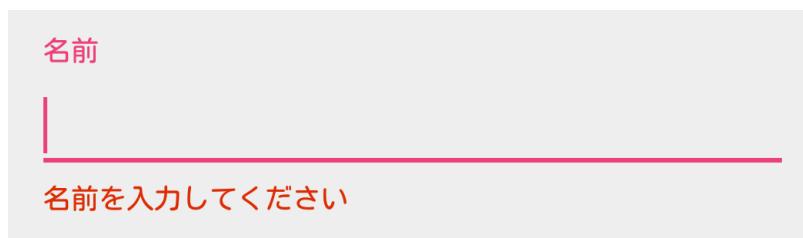


図 1.10 エラーメッセージの表示の例

エラーメッセージは TextInputLayout 上の最下部に表示されます。

エラーメッセージの表示方法は非常に簡単です（リスト 1.18）。

リスト 1.18: エラーメッセージの表示

```
TextInputLayout textInputLayout =  
    (TextInputLayout) findViewById(R.id.text_input_layout);  
textInputLayout.setError(getString(R.string.error_message));  
textInputLayout.setErrorEnabled(true);
```

setError メソッドで文字列をセットし、setErrorEnabled で表示非表示を切り替えます。

1.6 FloatingActionButton

Material Design といえば Floating Action Button を真っ先に思い浮かべる人もいるのではないかでしょうか。Material Design の中でも印象に残りやすい Floating Action Button ですが、なんと今まで用意されておらず独自実装する必要がありました。Android Design Support Library の登場で簡単にアプリに組み込むことができるようになりました。ただし、タップ時にサブメニューが出るような機能 (Speed dial^{*3}) は用意されていません。また、今回追加された FloatingActionButton は ImageView のサブクラスであるため、ImageView と同じように使うことができます。

この節では FloatingActionButton の表示方法、タップの検知、用意されているサイズについて説明します。

1.6.1 FloatingActionButton の表示方法

レイアウトリソースから画面右下に表示する FloatingActionButton を参照する方法は次のとおりです（リスト 1.19）。

リスト 1.19: レイアウトリソースに FloatingActionButton を追加

```
<FrameLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <android.support.design.widget.FloatingActionButton
```

^{*3} <https://www.google.com/design/spec/components/buttons-floating-action-button.html#buttons-floating-action-button-transitions>

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginRight="16dp"
    android:src="@drawable/ic_add"
    android:layout_gravity="bottom|right"
/>
</FrameLayout>
```

画面下に配置する Floating Action Button は画面の絶対的な位置に表示するため、FrameLayout や CoordinatorLayout の節で説明する CoordinatorLayout を使って配置するとよいでしょう。今回のサンプルでは android:layout_gravity 属性に bottom|right を指定することで右下に配置し、Material Design のガイドラインに合わせて android:layout_marginBottom 属性と android:layout_marginRight 属性に 16dp を指定しています。android:src 属性で Floating Action Button 中央に表示する画像を、app:fabSize 属性でボタン自体のサイズを指定することができます。これらについての詳細は後述します。

1.6.2 FloatingActionButton の大きさ

Material Design のガイドラインに合わせて 2 つのサイズが用意されており（表 1.3）、app:fabSize 属性で指定します。明示的にサイズを指定しない場合、normal として扱われます。

表 1.3 fabSize と dp

| fabSize | dp |
|---------|------|
| normal | 56dp |
| mini | 40dp |

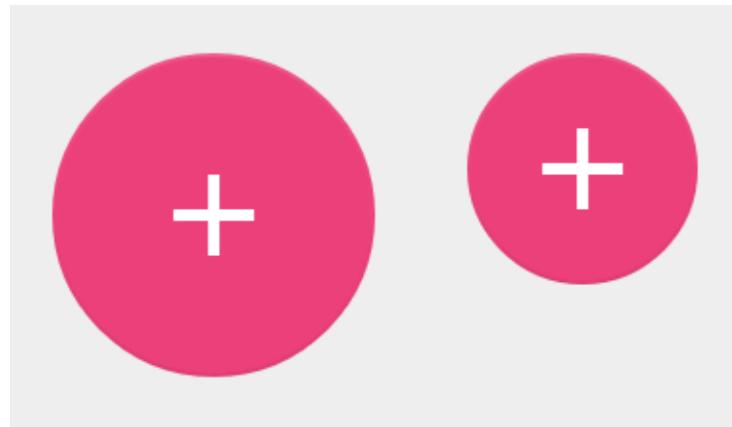


図 1.11 2つのサイズの Floating Action Button

1.6.3 画像の設定

FloatingActionButton は通常の ImageView と同じく android:src 属性で表示する画像を指定できます。表示される画像サイズは normal、mini 共に 24dp 固定です。図 1.11 を見るとボタンのサイズは異なるものの、中心の画像のサイズは同じことがわかります。内部的には 24dp になるように app:fabSize 属性から算出したパディングが設定されています。そのため android:layout_width 属性や android:layout_height 属性で直接大きさを指定すると画像のサイズが合わなくなってしまうので注意が必要です。

1.7 Snackbar

Snackbar は簡単なメッセージを表示するためのコンポーネントで、画面の下からスワイプインして表示されます（図 1.12）。

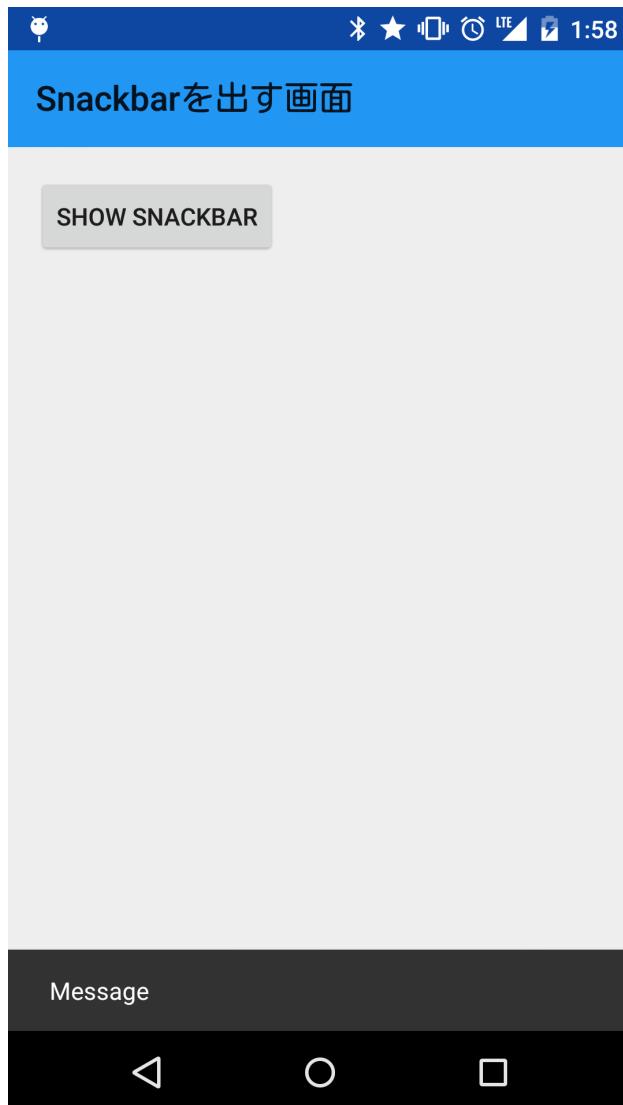


図 1.12 Snackbar (画面下)

Material Design 以前では簡単なメッセージを表示する際に、Toast というコンポーネントが使用されていましたが。Material Design のガイドラインの Snackbars & toasts^{*4}では Toast には殆ど触れておらず、Snackbar について細かく語られているため、今後は Snackbar を利用するとよいでしょう。Toast は画面から浮くような表示でメッセージを表示しており、一定時間で消えるようになっていました。Snackbar は Toast とよく似ていますが、Action を設定でき、スワイプで消すことができるなど、ユーザからの操作が可能という点で異なります。

この節では Snackbar の基本的な使い方として表示方法、Action の設定、表示時間の設定、そして色の変更について説明します。

^{*4} <https://www.google.com/design/spec/components/snackbars-toasts.html>

1.7.1 Snackbar の表示方法

Snackbar を表示する際のコードは、Toast を表示する際のコードとよく似ており、簡単に表示することができます（リスト 1.20）。

リスト 1.20: Snackbar を表示

```
Snackbar.make(parentView, "text", Snackbar.LENGTH_SHORT).show();
```

make メソッドで Snackbar のインスタンスを取得し、show メソッドで表示を行います。show メソッドを忘れるかと表示できないところまで Toast と同じですが、第 1 引数が Context ではなく View であることに気をつけてください。Snackbar 表示時に親となる View をここで指定する必要があります。表示した Snackbar はユーザがスワイプ操作を行うか、後述する表示時間の設定に合わせて画面から消えます。

1.7.2 Action の設定

Snackbar には Action と呼ばれるタップ可能なテキストをメッセージの右側に表示することで、Snackbar に対するユーザの反応を受け取ることができます（図 1.13）。Snackbar 自体が画面の表示やユーザの操作の邪魔になりにくいため、無視されてもよいがあると便利な機能を実装するのに向いています。たとえば完了メッセージに取り消しの Action、失敗メッセージに再実行の Action を設定するなどです。

第1章 Android Design Support Library

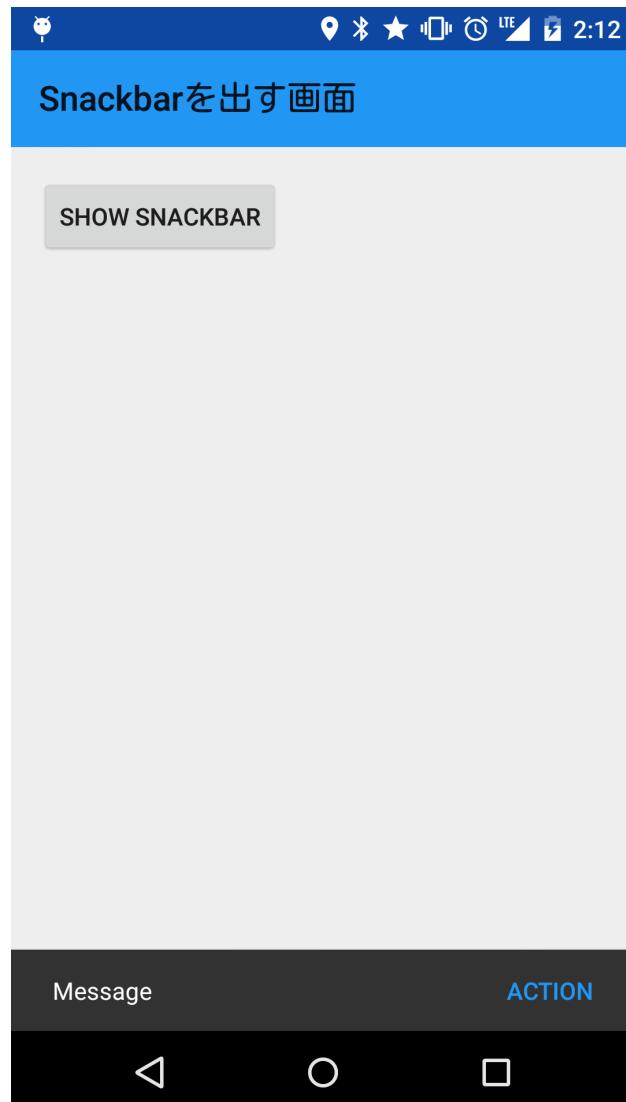


図 1.13 Action を設定した Snackbar

Snackbar クラスの `setAction` メソッドを用いて表示する文字列と `OnClickListener` をセットすることで Action を設定できます（リスト 1.21）。

リスト 1.21: Action の設定

```
Snackbar snackbar = Snackbar.make(parentView, R.string.snackbar_text,
    Snackbar.LENGTH_SHORT);
snackbar.setAction(R.string.snackbar_action, new View.OnClickListener() {
    @Override public void onClick(View v) {
        // TODO:action
    }
});
snackbar.show();
```

まず Snackbar クラスの make メソッドで Snackbar のインスタンスを取得し、setAction メソッドで表示する文字列と OnClickListener をセットします。OnClickListener には、ユーザが Action をタップした際に何を行うのかを実装してください。

1.7.3 表示時間の設定

make メソッドの第 3 引数 (duration) の値によって Snackbar の表示時間を変更することができます（リスト 1.22）。

リスト 1.22: make メソッドによる表示時間の設定

```
Snackbar snackbar = Snackbar.make(parentView, R.string.snackbar_text,  
        Snackbar.LENGTH_SHORT).show();
```

また、make メソッドでインスタンスを作成したあとでも、setDuration メソッドを使えば make メソッドで指定した時間を上書きすることができます。

リスト 1.23: setDuration メソッドによる表示時間の上書き

```
Snackbar snackbar = Snackbar.make(parentView, R.string.snackbar_text,  
        Snackbar.LENGTH_SHORT);  
snackbar.setDuration(Snackbar.LENGTH_LONG);  
snackbar.show();
```

make メソッドは Snackbar のインスタンスを取得する際に必ず使用するため、setDuration メソッドを使う機会はあまり多くなさそうです。表示時間には表 1.4 の 3 種の定数と 0 より大きい整数を設定できます。

表示時間として定数を指定すると決まった時間 Snackbar を表示でき、0 より大きい整数を指定すると指定した値をミリ秒として Snackbar を表示できます。短いメッセージの表示には Snackbar.LENGTH_SHORT、長いメッセージや Action を含む Snackbar の表示には Snackbar.LENGTH_LONG を使用するとよいでしょう。しかし、メッセージや Action の内容によっては 2.75 秒は短すぎるかもしれません。そのような場合には Snackbar を永続的に表示できる Snackbar.LENGTH_INDEFINITE を指定するか、直接ミリ秒を指定しましょう。

表 1.4 設定可能な表示時間

| 値 | 表示時間 |
|----------------------------|--------|
| Snackbar.LENGTH_SHORT | 1.5 秒 |
| Snackbar.LENGTH_LONG | 2.75 秒 |
| Snackbar.LENGTH_INDEFINITE | 無制限 |
| 0 より大きい整数 | ミリ秒 |

表示された Snackbar は時間経過以外にも、ユーザのスワイプ操作や dismiss メソッドを使うことで非表示にすることができます。dismiss メソッドを使えば、他のイベントに合わせて Snackbar を非表示にすることができるため、Snackbar を非表示にするボタンを設置することも可能です（リスト 1.24）。

リスト 1.24: Snackbar を表示するボタンと非表示にするボタンをもつ Activity

```
private Snackbar mSnackbar;

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

    findViewById(R.id.show_snackbar_button).setOnClickListener(this);
    findViewById(R.id.dismiss_snackbar_button).setOnClickListener(this);
}

@Override public void onClick(View v) {
    switch (v.getId()) {
        case R.id.show_snackbar_button:
            View parentView = findViewById(R.id.coordinator_layout);
            mSnackbar
                = Snackbar.make(parentView, "Message", Snackbar.LENGTH_INDEFINITE);
            mSnackbar.show();
            break;
        case R.id.dismiss_snackbar_button:
            if (mSnackbar == null) return;
            mSnackbar.dismiss();
            break;
    }
}
```

dismiss メソッドを使うには Snackbar のインスタンスが必要となるため、show する前にインスタンスを保持しておく必要があります。

1.7.4 色の変更

初期状態では背景色には#323232（暗めの灰色）が、Action の文字色には colorPrimary が設定されています。これらを変更するには setActionTextColor メソッドと getView メソッドを使用します（リスト 1.25）。

リスト 1.25: Snackbar の色の変更

```
Snackbar snackbar = Snackbar.make(parentView, R.string.snackbar_text,
    Snackbar.LENGTH_SHORT);
snackbar.setAction(R.string.snackbar_action, new View.OnClickListener() {
    @Override public void onClick(View v) {
        // TODO:action
    }
});
snackbar.setActionTextColor(getResources()
    .getColor(R.color.snackbar_action_color));
snackbar.getView().setBackgroundColor(getResources()
    .getColor(R.color.snackbar_color));
snackbar.show();
```

make メソッドで Snackbar のインスタンスを取得し、setActionTextColor で Action の文字色を設定します。Snackbar の背景色を設定するには getView メソッドで Snackbar の View を取得し、取得した View クラスの setBackgroundColor メソッドで背景色を設定します。

1.8 CoordinatorLayout

CoordinatorLayout は FrameLayout と同じように使うことができ、更に CoordinatorLayout の子（あるいは子孫）View 同士に関係を持たせることが出来るコンポーネントです。ある子 View の動きに合わせてもう 1 つの子 View を動かすことができます。何を出来るのか理解し難いですが、Android Design Support Library には CoordinatorLayout を使用して View

を連動させる機能が多く用意されています。これらの動きを学ぶことで CoordinatorLayout の理解を深めることができます。この節で解説する Snackbar + FloatingActionButton や、この後の節で解説する AppBarLayout、CollapsingToolbarLayout も CoordinatorLayout の利用を前提としています。

この節では CoordinatorLayout の重要な概念である Anchor、Behavior の説明、そしてそれらの使用例を紹介します。

1.8.1 Anchor

Anchor を使うと CoordinatorLayout の子孫 View 同士に関係を持たせることができます。大きくふたつの特徴があります。

- 参照先の View との相対的な View 配置
- 参照先の View が移動した場合の追従

Toolbar の境界線に Floating Action Button を配置する例を紹介します（図 1.14）。

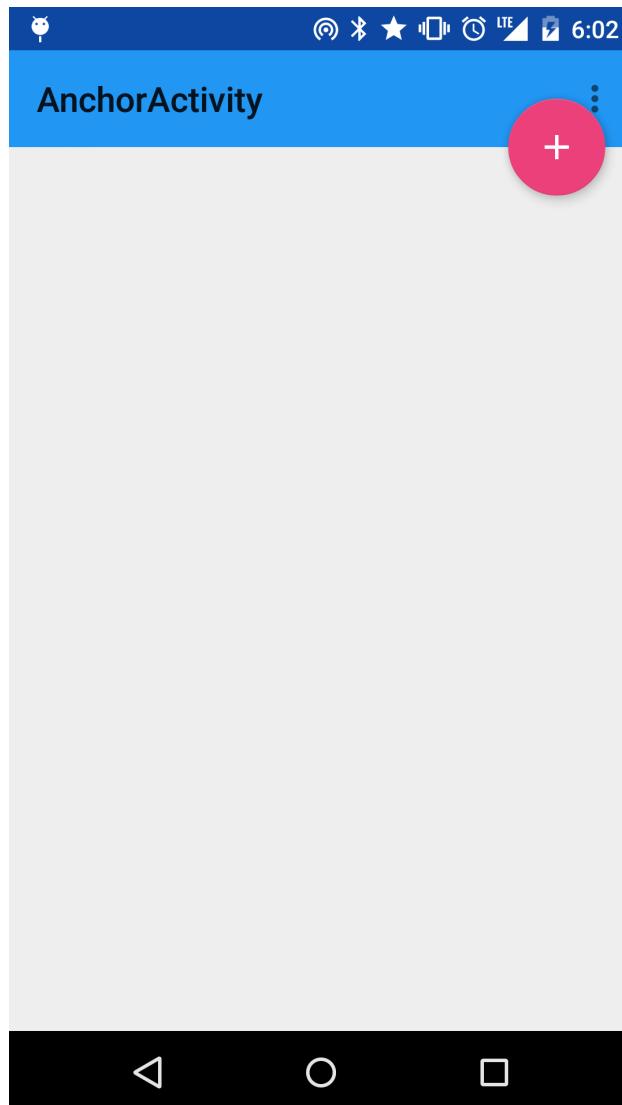


図 1.14 境界線に FloatingActionButton を置く

このサンプルでは FloatingActionButton の Anchor に Toolbar を指定しています。Anchor を指定したことでの、Toolbar の境界線に簡単に FloatingActionButton を配置することができます（リスト 1.26）。

リスト 1.26: Anchor を使って境界線に FloatingActionButton を置く

```
<android.support.design.widget.CoordinatorLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    >  
    <android.support.v7.widget.Toolbar
```

```
    android:id="@+id/tool_bar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"
/>
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:src="@drawable/ic_add"
    app:layout_anchor="@+id/tool_bar"
    app:layout_anchorGravity="right|bottom"
/>
</android.support.design.widget.CoordinatorLayout>
```

CoordinatorLayout は FrameLayout と同じ特徴を持っているため、`layout_gravity` 属性などを使わない限り各子 View が重なってしまいます。今回は Anchor を使って表示位置を調整しているため、Floating Action Button を Toolbar の境界線に配置できています。

`app:layout_anchor` 属性で Anchor とする View の ID を指定します。さらに `app:layout_anchorGravity` 属性で Anchor に対する表示位置を調整します。`app:layout_anchorGravity` 属性に指定できる値は通常の `android:layout_gravity` 属性で指定できる値と同じです。

また、Anchor として指定した View が移動すると、その Anchor を参照している View も同様に移動します。この挙動は後述する Behavior が決めています。

1.8.2 Behavior

Behavior では Anchor に対する振る舞いを決めることができます。CoordinatorLayout.Behavior が基本的な Behavior として適応されており、Anchor として指定した View が移動すると、その Anchor を参照している View も同様に移動します。ただし、追従する形では画面の外にでることはできません。また、独自の Behavior を作成、指定することで他の View の動きに合わせてさまざまなアニメーションを行うことが可能です。独自の Behavior を作成、指定する方法について詳しくは後述します。

1.8.3 Snackbar + FloatingActionButton

FrameLayoutなどを用いて、画面下に配置する FloatingActionButton と Snackbar を同時に利用すると両者が重なっていまいます。CoordinatorLayout を用いればこの問題を簡単に解決できます（図 1.15）。

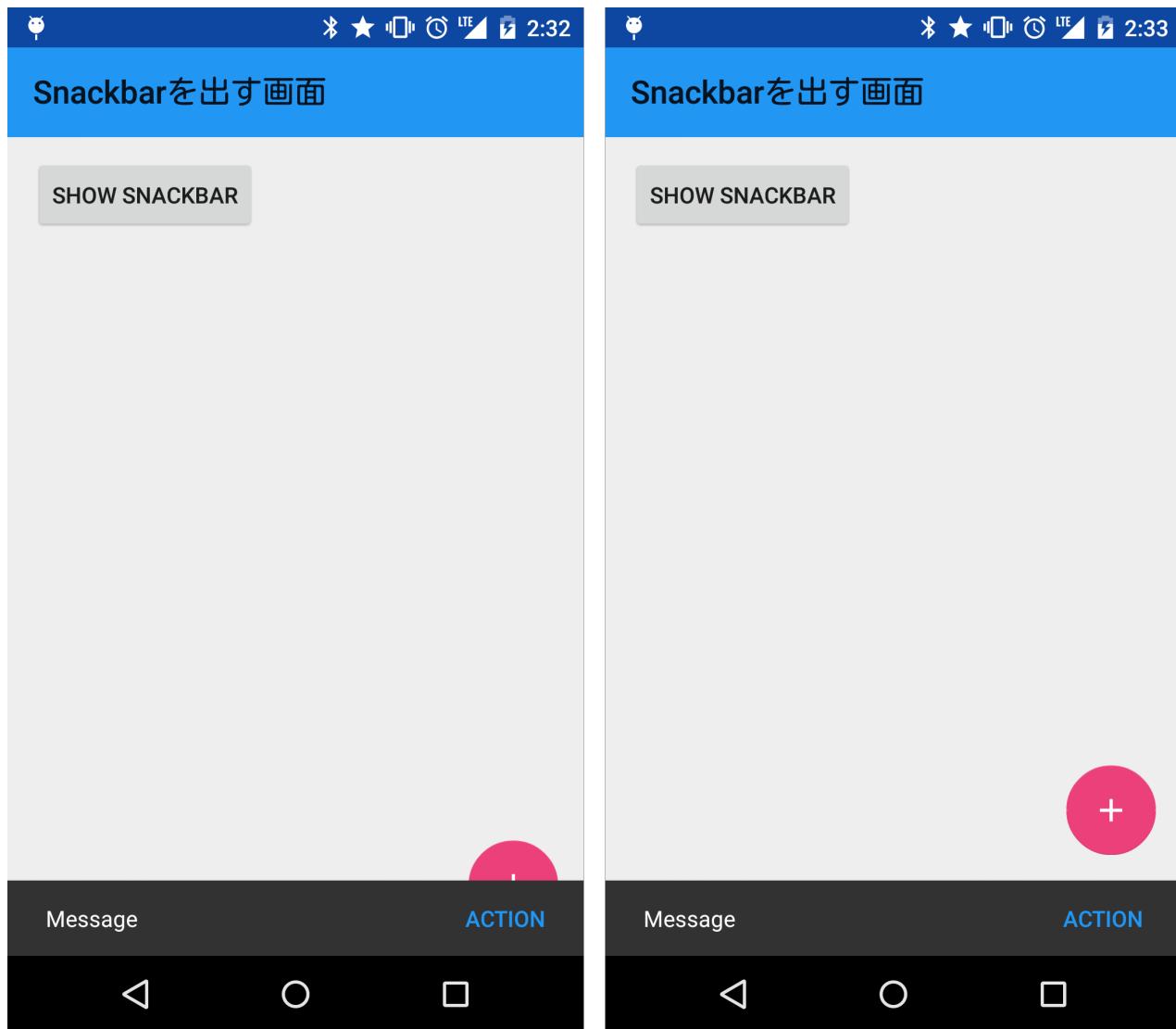


図 1.15 CoordinatorLayout 適応前（左）、CoordinatorLayout 適応後（右）

図 1.15 の左の画面では FloatingActionButton と Snackbar が重なってしまいました。 CoordinatorLayout を適応した右側の画面では、Snackbar の表示時に FloatingActionButton が Snackbar を避けているため、両者が重なっていないことがわかります。

Snackbar と FloatingActionButton に CoordinatorLayout を適応するには、Coordinator-

第1章 Android Design Support Library

Layout の子 View として FloatingActionButton を配置し、Snackbar の親 View として同じ CoordinatorLayout を指定するだけです。CoordinatorLayout の使用方法レイアウトリソース上では次のようにになります（リスト 1.27）。

リスト 1.27: Snackbar と FloatingActionButton の親要素として CoordinatorLayout を指定

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/coordinator_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="?attr/colorPrimary"
            android:minHeight="?attr/actionBarSize"/>

        <Button
            android:id="@+id/show_snackbar_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="show snackbar"
            android:layout_margin="16dp"/>
    </LinearLayout>

    <android.support.design.widget.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:src="@drawable/ic_add"
        android:layout_gravity="bottom|right"
        />
</android.support.design.widget.CoordinatorLayout>
```

Snackbar から CoordinatorLayout を参照するために CoordinatorLayout タグに android:id 属性を追加するのを忘れないようにしてください。Snackbar 表示時に CoordinatorLayout

を第1引数に指定します（リスト1.28）。

リスト1.28: CoordinatorLayoutを親ViewとするSnackbarの表示

```
View parentView = findViewById(R.id.coordinator_layout);
Snackbar.make(parentView, "Message", Snackbar.LENGTH_INDEFINITE).show();
```

FloatingActionButtonには独自のBehaviorが設定されており、親ViewであるCoordinatorLayoutの子ViewにSnackbarが表示されると、FloatingActionButtonはアニメーション付きでSnackbarを避けてくれます。CoordinatorLayoutの子ViewとしてSnackbarとFloating Action Buttonがあるケースを考えましょう。Snackbarは、画面下にメッセージを表示します。CoordinatorLayoutを使わなければ、Floating Action Buttonと位置が重複し、どちらかが見切れます。このような見切れ、重複はUIとして美しくありません。CoordinatorLayoutはFloatingActionButtonに初期設定されているBehaviorを通じて、このような見切れがないようにSnackbarが表示されるタイミングで連動してFloating Action Buttonの位置を動かします。連動時のアニメーションもサポートしており、スムーズな表現が可能です。見た目も綺麗ですし、連動する、という難しい動きをプログラミングしないで済む使い勝手のよいレイアウトです。

1.8.4 独自Behaviorの設定方法

自分でCoordinatorLayout.Behaviorクラスやそのサブクラスを継承するBehaviorを作成した場合など、最初から設定されているBehaviorとは異なるBehaviorを使用するにはBehaviorを設定する必要があります。Behaviorを設定するには次の3つの方法があります。

- レイアウトリソースのlayout_behavior属性
- CoordinatorLayout.LayoutParamsクラスのsetBehaviorメソッド
- @DefaultBehaviorアノテーション

それぞれの実装方法を具体的に説明していきます。

レイアウトリソースのlayout_behavior属性

レイアウトリソース上からBehaviorを指定するには、layout_behavior属性を使用します（リスト1.29）。

第1章 Android Design Support Library

リスト 1.29: FloatingActionButton に Behavior を指定するレイアウトリソース

```
<<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:src="@drawable/ic_add"
        android:layout_gravity="bottom|right"
        app:layout_behavior="com.example.CustomBehavior">
    />
</android.support.design.widget.CoordinatorLayout>
```

このサンプルでは FloatingActionButton に Behavior を指定しています。Behavior を指定する際には、パッケージ名も含める必要がある点に気をつけてください。また、レイアウトリソースから指定する Behavior には第 1 引数が Context、第 2 引数が AttributeSet のコンストラクタが必要になります（リスト 1.30）。

リスト 1.30: レイアウトリソースから指定する Behavior に必要なコンストラクタ

```
public class CustomBehavior extends CoordinatorLayout.Behavior {
    public CustomBehavior(Context context, AttributeSet attrs) {
        super();
    }
}
```

CoordinatorLayout.LayoutParams クラスの setBehavior メソッド

レイアウトリソース上からだけでなく、Java のコードからでも Behavior を指定できます（リスト 1.31）。

リスト 1.31: setBehavior メソッドによる Behavior の指定

```
@Override protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_behavior);  
  
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);  
    CoordinatorLayout.LayoutParams layoutParams = fab.getLayoutParams();  
    layoutParams.setBehavior(new CustomBehavior());  
}
```

Behavior を指定したい View から getLayoutParams メソッドで LayoutParams クラスのインスタンスを取得、CoordinatorLayout.LayoutParams クラスにキャストし、CoordinatorLayout.LayoutParams クラスの setBehavior メソッドで Behavior を指定します。

@DefaultBehavior アノテーション

@DefaultBehavior アノテーションを使うと、setBehavior メソッドや setBehavior メソッドや layout_behavior 属性から Behavior を指定しなかった場合に使う Behavior を指定できます。カスタム View に Behavior を指定したい場合に使用するとよいでしょう。カスタム View で @DefaultBehavior アノテーションを使用すると次のようになります（リスト 1.32）。

リスト 1.32: @DefaultBehavior アノテーションを使用するカスタム View

```
@DefaultBehavior(CustomView.CustomBehavior.class)  
public class CustomView extends View{  
    ...  
    public class CustomBehavior extends CoordinatorLayout.Behavior {  
        ...  
    }  
    ...  
}
```

カスタム View のクラスに @DefaultBehavior アノテーションで Behavior のクラスを指定します。今回のサンプルではカスタム View の中に実装した Behavior を指定しています。

1.8.5 スクロールに合わせて Floating Action Button を隠す独自 Behavior

この項では独自 Behavior の実装例としてスクロールに合わせて画面から Floating Action Button を隠すことのできる独自 Behavior を作成します。

まずレイアウトリソースを作成します（リスト 1.33）。

リスト 1.33: スクロールと FloatingActionButton を含むレイアウトリソース

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <!-- your content --!>

    </android.support.v7.widget.NestedScrollView>

    <android.support.design.widget.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:src="@drawable/ic_add"
        android:layout_gravity="bottom|right"
        />
</android.support.design.widget.CoordinatorLayout>
```

全体の親 View を CoordinatorLayout とし、NestedScrollView、FloatingActionButton をその子 View とします。FestedScrollView にはスクロールするために縦方向に長さのある Viewを入れてください。NestedScrollView の代わりに RecyclerView を使うこともできます（使うことのできるスクロール可能な View について、詳しくは AppBarLayout の節で後述します）。

第1章 Android Design Support Library

独自 Behavior を実装するには、CoordinatorLayout.Behavior クラスを継承した独自 Behavior クラスを作成します。対象となるコンポーネントが既に独自 Behavior を持っている場合、その Behavior を継承したほうがよいでしょう。今回は対象となる FloatingActionButton が独自 Behavior を持っているため、FloatingActionButton 用の Behavior である FloatingActionButton.Behavior クラスを継承します。

FloatingActionButton.Behavior クラスを継承したクラスを作成します（リスト 1.34）。

リスト 1.34: FloatingActionButton.Behavior クラスを継承したクラス

```
public class MyFabBehavior extends FloatingActionButton.Behavior {  
    public MyFabBehavior(Context context, AttributeSet attrs) {  
        super();  
    }  
}
```

前項で説明したとおり、レイアウトリソースから Behavior を指定するには Context と AttributeSet を引数とするコンストラクタが必要となるため、実装しています。作成した Behavior をレイアウトリソースから指定します。

リスト 1.35: app:layout_behavior 属性による Behavior の指定

```
<android.support.design.widget.CoordinatorLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/coordinator_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
<android.support.v4.widget.NestedScrollView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >  
  
<!-- your content --!>  
  
</android.support.v7.widget.NestedScrollView>  
  
<android.support.design.widget.FloatingActionButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginBottom="16dp"
```

```
    android:layout_marginRight="16dp"
    android:src="@drawable/ic_add"
    android:layout_gravity="bottom|right"
    app:layout_behavior="com.example.MyFabBehavior"
/>
</android.support.design.widget.CoordinatorLayout>
```

次に onStartNestedScroll メソッドをオーバーライドします（リスト 1.36）。

リスト 1.36: オーバーライドした onStartNestedScroll メソッド

```
public class MyFabBehavior extends FloatingActionButton.Behavior {
    ...
    @Override public boolean onStartNestedScroll(
        CoordinatorLayout coordinatorLayout, FloatingActionButton child,
        View directTargetChild, View target, int nestedScrollAxes) {
        return nestedScrollAxes == ViewCompat.SCROLL_AXIS_VERTICAL;
    }
}
```

onStartNestedScroll メソッドはスクロール開始時に呼ばれるメソッドで、true を返すと onNestedScroll メソッドなどを通じてこのスクロールの一連のイベントを受け取ることが出来ますが、false を返すとそのスクロールの情報を取得できなくなります。引数は coordinatorLayout が親（あるいは先祖）View となる CoordinatorLayout、child が Behavior をセットする FloatingActionButton 自身、directTargetChild はスクロールする View の子 View、target はスクロールする View、nestedScrollAxes はスクロール方向の情報です。onStartNestedScroll メソッド内でこのスクロールがハンドリングしたいスクロールなのかを判別します。今回は縦スクロールをハンドリングしたいので、nestedScrollAxes を ViewCompat.SCROLL_AXIS_VERTICAL 定数と比較し縦スクロール判定を行っています。また今回は行っていませんが、スクロールする View が複数ある場合は、View の ID を比較し対象の View なのか判別する必要があります。

続いて onNestedScroll メソッドをオーバーライドします。

リスト 1.37: オーバーライドした onNestedScroll メソッド

```
public class MyFabBehavior extends FloatingActionButton.Behavior {  
    ...  
    @Override  
    public void onNestedScroll(CoordinatorLayout coordinatorLayout,  
        FloatingActionButton child, View target, int dxConsumed, int dyConsumed,  
        int dxUnconsumed, int dyUnconsumed) {  
        if (dyConsumed > 0) {  
            animateHide(child);  
        } else if (dyConsumed < 0){  
            animateShow(child);  
        }  
    }  
}
```

onNestedScroll メソッドはスクロール時に隨時呼ばれるメソッドですが、onStartNestedScroll メソッドで false を返していた場合、そのスクロールでは onNestedScroll メソッドは呼ばれません。引数は onStartNestedScroll メソッドと似ていますが、スクロールの変化量に関するものが含まれています。dxConsumed、dyConsumed はそれぞれ X 軸（垂直方向）、Y 軸（水平方向）に対するスクロールの変化量（ピクセル）で、dxUnconsumed、dyUnconsumed はそれぞれ X 軸（垂直方向）、Y 軸（水平方向）に対するスクロール出来なかった変化量（ピクセル）です。画面端でスクロールする場合など、ユーザがスクロールのアクションを行ったが画面が動いていない場合には dxUnconsumed、dyUnconsumed にその分の値が入ります。また、今回はどちらの方向にスクロールしているのかを判別する必要があるため、dxConsumed の正負でスクロール方向を判別しています。

FloatingActionButton のアニメーションを実装します（リスト 1.38）。

リスト 1.38:

```
public class MyFabBehavior extends FloatingActionButton.Behavior {  
    ...  
    private boolean mIsAnimating;  
    ...  
    @Override  
    public void onNestedScroll(CoordinatorLayout coordinatorLayout,  
        FloatingActionButton child, View target, int dxConsumed, int dyConsumed,  
        int dxUnconsumed, int dyUnconsumed) {  
        if (dyConsumed > 0 && !mIsAnimating) {  
            animateHide(child);  
        }  
    }  
}
```

```
        } else if (dyConsumed < 0 && !mIsAnimating) {
            animateShow(child);
        }
    }

private void animateHide(FloatingActionButton button) {
    ViewCompat.animate(button).scaleX(0.0F).scaleY(0.0F).alpha(0.0F)
        .setInterpolator(new FastOutSlowInInterpolator())
        .setListener(new ViewPropertyAnimatorListener() {
            @Override public void onAnimationStart(View view) {
                mIsAnimating = true;
            }

            @Override public void onAnimationEnd(View view) {
                mIsAnimating = false;
            }

            @Override public void onAnimationCancel(View view) {
                mIsAnimating = false;
            }
        });
}

private void animateShow(FloatingActionButton button) {
    ViewCompat.animate(button).scaleX(1.0F).scaleY(1.0F).alpha(1.0F)
        .setInterpolator(new FastOutSlowInInterpolator());
}
```

今回追加した `animateHide` メソッドで `FloatingActionButton` を隠すアニメーションを、`animateShow` メソッドで `FloatingActionButton` を表示するアニメーションを行っています。`ViewCompat` クラスの `animate` メソッドを使い、`FloatingActionButton` が自身の中央に向かって小さくなると共に透過していくアニメーションを実装しています。また、アニメーション中にアニメーションを行わないように、`setListener` メソッドでアニメーション開始時と終了時のイベントを取得し、`mIsAnimating` フィールドに状態をもたせ、`mIsAnimating` フィールドの値によって、アニメーションを行うか行わないかの判別を `onNestedScroll` メソッド内で行っています。

また、次節で後述する `AppBarLayout` と組み合わせる場合は `onNestedScroll` メソッドの代わりに `onNestedPreScroll` メソッドをオーバーライドして使用しましょう。`onNestedPreScroll` メソッドは `onNestedScroll` メソッドの前に呼ばれるメソッドで、スクロールできる

できないの計算を行う前に呼ばれます。AppBarLayout の表示非表示のスクロールは、スクロール可能な View がスクロールしているわけではなく、AppBarLayout とスクロール可能な View をずらすことで実現しています。このため、AppBarLayout のスクロール中は onNestedPreScroll メソッドは呼ばれます but onNestedScroll メソッドが呼ばれず、AppBarLayout のスクロールが完了した後に onNestedScroll メソッドが呼ばれ始めます。AppBarLayout のスクロール完了後にアニメーションを行いたい場合は onNestedScroll メソッドのままで問題ありませんが、AppBarLayout のスクロールに合わせてアニメーションを行いたい場合は onNestedPreScroll メソッドを使用しましょう。

1.9 AppBarLayout

AppBarLayout はスクロールに合わせて Toolbar を隠すことができるコンポーネントです。

この節では AppBarLayout の基本的な使い方と、AppBarLayout に設定可能な値、AppBarLayout と連動可能な View、TabLayout と組み合わせた場合の使用例について説明します。

1.9.1 AppBarLayout の表示方法

CoordinatorLayout の直下で使用することを想定されています（リスト 1.39）。

リスト 1.39: レイアウトリソースに AppBarLayout を追加

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/tool_bar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="?attr/colorPrimary"
```

```
    android:minHeight="?attr/actionBarSize"
    app:layout_scrollFlags="scroll|enterAlways"/>

</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

</android.support.v7.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

リスト 1.39 にあるとおり、AppBarLayout と AppBarLayout で管理したい View（サンプルでは Toolbar）を入れ子にし、CoordinatorLayout の子 View として配置します。運動させたいスクロールする View（サンプルでは NestedScrollView）も同じ CoordinatorLayout の子 View にする必要があります。

1.9.2 スクロール時の子 View の挙動の変更

スクロール時に、運動して挙動を変更したいコンポーネントがある場合、該当の View (AppBarLayout の子 View です) に layout_scrollFlags 属性を追加します（表 1.5）。

表 1.5 app:layout_scrollFlag 属性に指定できる値

| 値 | 説明 |
|----------------------|----------------------------|
| scroll | スクロール可能な状態にします |
| enterAlways | 下スクロールで隠れた View をすぐ表示します |
| enterAlwaysCollapsed | 限界まで下スクロールで隠れた View を表示します |
| exitUntilCollapsed | enterAlwaysCollapsed の変形です |

app:layout_scrollFlags 属性には|（パイプ）区切りで複数の値を指定することができます。scroll 以外の値は scroll と組み合わせて使用することを前提としているため、リスト 1.39 のように scroll と他に何か 1 つを組み合わせて指定するとよいでしょう。scroll を指定すると、運動する View を上方向にスクロールする際に、スクロールに合わせて指定した View が画面の外に追い出されます。スクロール可能な状態にし、上方向スクロールで View が画面外に移

動するように隠れます。enterAlways を指定すると、下方向スクロール時に View が隠れている場合、スクロールに合わせて View を表示します。enterAlwaysCollapsed を指定すると、下方向スクロール時に View が隠れている場合に、限界まで下スクロールを行うと、スクロールに合わせて View を表示します。exitUntilCollapsed は enterAlwaysCollapsed と基本的に同じですが、`android:minHeight` 属性で指定した高さだけは常に表示されます。

1.9.3 連動できるスクロール可能な View

AppBarLayout とスクロールを連動することが可能な View は NestedScrollingChild を実装している View に限られます。代表的なものだと RecyclerView と NestedScrollView が該当しますが、Support Library のバージョンによっては NestedScrollingChild を実装していませんので、本章の冒頭でも説明したとおり Android Design Suppot Library のバージョンに合わせた Support Library を使うようにしましょう。

1.9.4 スクロール可能な View と AppBarLayout の連動

スクロール可能な View の Behavior に `AppBarLayout.ScrollingViewBehavior` をセットすると AppBarLayout と連動できます。レイアウトリソースから指定する場合はリスト 1.39 のように `app:layout_behavior` 属性に `@string/appbar_scrolling_view_behavior` をセットします。

1.9.5 AppBarLayout + TabLayout

Toolbar の下に TabLayout を配置し、スクロール時に Toolbar だけを隠し、TabLayout は常に表示されているような画面は Google Play ストアアプリなどでよく見かけます。この場合のレイアウトリソースはどうなるのか見てみましょう（リスト 1.40）。

リスト 1.40: AppBarLayout に TabLayout を追加

```
<android.support.design.widget.CoordinatorLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/coordinator_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <android.support.v7.widget.Toolbar
        android:id="@+id/tool_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        app:layout_scrollFlags="scroll|enterAlways"/>

    <android.support.design.widget.TabLayout
        android:id="@+id/tab_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

    </android.support.v7.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

AppBarLayout は LinearLayout のサブクラスであるため、縦方向の LinearLayout と同じように使うことができます。Toolbar のあとに TabLayout を置くことで Toolbar の下に TabLayout を配置できます。さらに TabLayout に app:layout_scrollFlags 属性を設定しないことで TabLayout のみを残すことができます。

1.10 CollapsingToolbarLayout

CollapsingToolbarLayout を使うとスクロールに合わせて縦幅の大きな AppBar^{*5}を縮小し、通常のサイズの AppBar に収めることができます（図 1.16）。

*5 ここでは複数の View などを組み合わせて Toolbar を拡張したものを AppBar と読んでいます

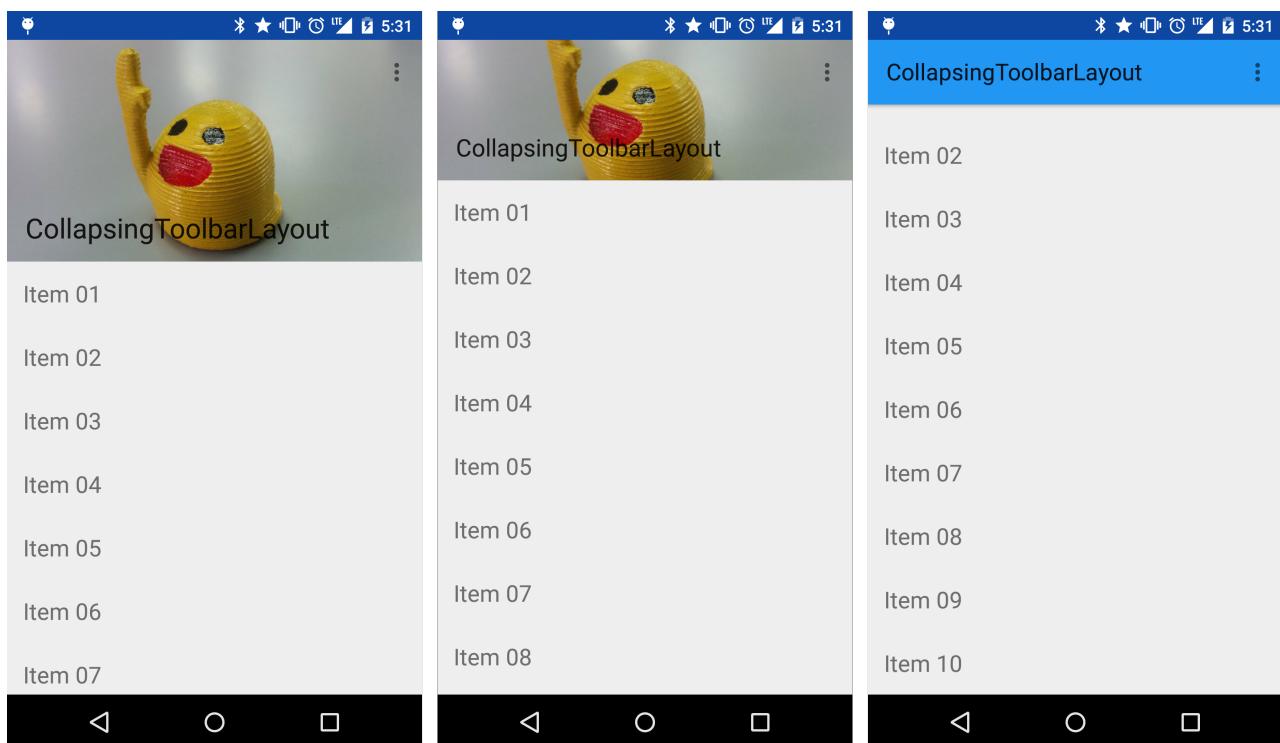


図 1.16 スクロール前（左）、スクロール中（中央）、スクロール後（右）

スクロールに合わせて図 1.16 の左の画面から中央の画面、そして右の画面のように AppBar を変化させることができます。詳細画面などで画面のヘッダに画像を表示させたいが、ヘッダ画像で画面に表示される情報量を減らしたくないときに向いているでしょう。

この節では CollapsingToolbarLayout の基本的な使い方と、スクロール時に設定可能な挙動、タイトルの設定方法、縮小時の背景画像の設定方法について説明します。

1.10.1 CollapsingToolbarLayout を表示する

CollapsingToolbarLayout は AppBarLayout の子 View として使うことを前提としています。また、Toolbar を子 View としている必要があります。レイアウトリソース上では次のようにになります（リスト 1.41）。

リスト 1.41: CollapsingToolbarLayout を使用する際の基本的なレイアウト

```
<android.support.design.widget.CoordinatorLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/coordinator_layout"  
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="192dp">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_scrollFlags="scroll|exitUntilCollapsed">

            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:src="@drawable/header"
                app:layout_collapseMode="parallax"/>

            <android.support.v7.widget.Toolbar
                android:id="@+id/tool_bar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"/>

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <!--content-->

    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

AppBarLayout は CoordinatorLayout の子 View として使うことになっているため、CoordinatorLayout、AppBarLayout、CollapsingToolbarLayout の順で入れ子になります。CollapsingToolbarLayout は FrameLayout のサブクラスですので、子 View の重なりに気をつける必要があります。android:layout_gravity="bottom"で Toolbar を最下部に配置するとスクロール後に Toolbar にセットしたメニューが見えなくなってしまうので注

意してください。CollapsingToolbarLayout にセットする app:layout_scrollFlags 属性は scroll|exitUntilCollapsed のままがよいでしょう。AppBarLayout で説明した値がそのまま使えますが、exitUntilCollapsed を指定しない場合はスクロールでそのまま画面外に消えてしまいしますので、CollapsingToolbarLayout を使う意味が薄くなってしまいます。また、AppBarLayout と Toolbar の android:layout_height 属性は wrap_content ではなく固定サイズにする必要があります。

1.10.2 スクロール時の子 View の挙動の変更

CollapsingToolbarLayout の子 View には app:layout_collapseMode 属性を指定することでスクロール時の挙動を変更できます（表 1.6）。

表 1.6 app:layout_collapseMode 属性の値

| collapseMode | 説明 |
|--------------|---------------|
| pin | スクロールに追従しない |
| parallax | スクロールの変化量が少ない |

app:layout_collapseMode 属性に pin を指定すると、CollapsingToolbarLayout が変化してもスクロールされずに CollapsingToolbarLayout 内に残ります。ただしスクロール後には CollapsingToolbarLayout が Toolbar の高さまで小さくなるので、android:layout_gravity などで View を移動させている場合は CollapsingToolbarLayout の縮小に合わせて画面から消えてしまうことがあります。また、スクロール後は Toolbar の高さになりますので、Toolbar より大きいサイズの View はスクロールとともに切れて表示されます。

app:layout_collapseMode 属性に parallax を指定すると、上下均等に CollapsingToolbarLayout から消えるようにスクロールの変化量が調整されます。

基本的には Toolbar には pin を、それ以外の View には parallax をセットすると良さそうです。図 1.16 では Toolbar に pin を、ImageView に parallax をセットしています。中央のスクロール中のスクリーンショットでは、通常のスクロールと異なり画像が上下切れていることがわかります。

1.10.3 タイトルのセット

`setSupportActionBar` メソッドで `Toolbar` をセットしても `Toolbar` にタイトルが表示されません。リスト 1.42 のように `CollapsingToolbarLayout` 用のタイトルをセットしましょう。このタイトルは `CollapsingToolbarLayout` の変化に合わせてサイズや位置がアニメーションします。

リスト 1.42: `collapsingToolbarLayout` にタイトルを追加する

```
CollapsingToolbarLayout collapsingToolbarLayout
    = (CollapsingToolbarLayout) findViewById(R.id.collapsing_toolbar);
collapsingToolbarLayout.setTitle("title");
```

`findViewById` メソッドで `CollapsingToolbarLayout` のインスタンスを取得し、`setTitle` で `CollapsingToolbarLayout` にタイトルをセットします。なお、タイトルの表示は XML では行うことができないためリスト 1.42 のようにコードから行う必要があります。

1.10.4 縮小後の `Toolbar` の色の変える

`CollapsingToolbarLayout` に `app:contentScrim` 属性を指定することで縮小後の `Toolbar` の色を変更することができます（リスト 1.43）。

リスト 1.43: 縮小後の `Toolbar` の色を変更

```
<android.support.design.widget.CollapsingToolbarLayout
    android:id="@+id/collapsing_toolbar"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:contentScrim="#666666"
    app:layout_scrollFlags="scroll|exitUntilCollapsed">
```

`app:contentScrim` 属性で色を指定する場合は `CollapsingToolbarLayout` の子 View の順番に気をつけてください。`Toolbar` に他の View が重なっていると `Toolbar` が隠れてしまいので、`app:contentScrim` 属性で色を指定する意味がありません。図 1.16 では `app:contentScrim` で色を指定しています。その為スクロール後（右）の画像では `ImageView`

でセットした画像が表示されていません。app:contentScrim 属性を指定しない場合は図 1.17 のようになります。

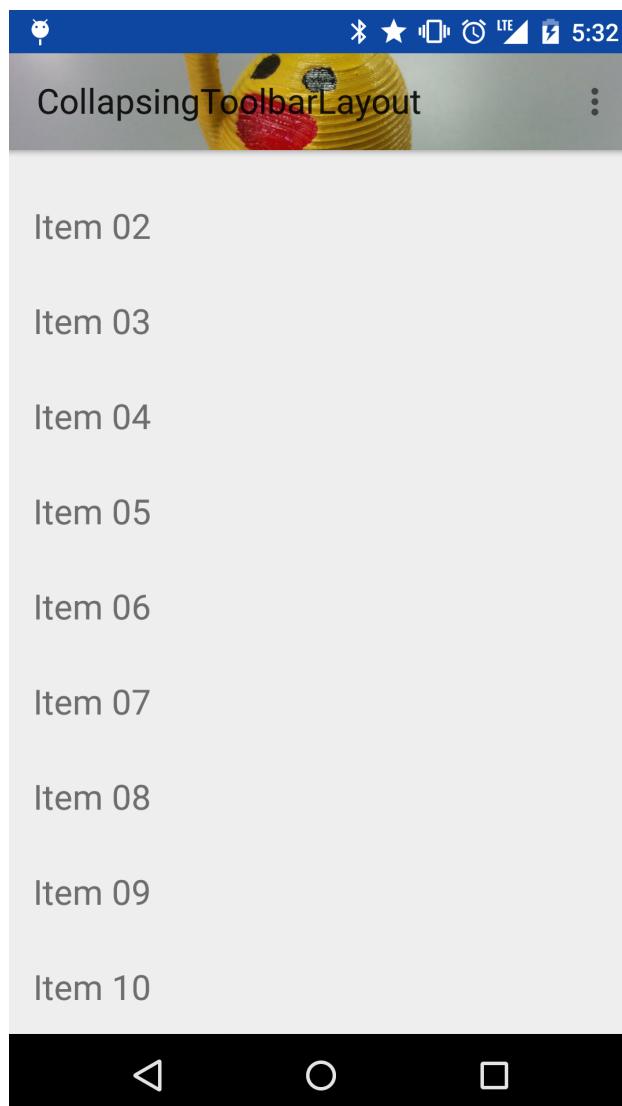


図 1.17 app:contentScrim 属性を指定しない場合の画面

app:contentScrim 属性を指定しない場合、Toolbar が塗りつぶされないので Toolbar の下の View も表示されたままになります。

1.11 開発時のお役立ち情報

開発時に役立つ公式の情報やサンプルプログラムなどを紹介します。

1.11.1 リソースの参照

SDK Manager を通してライブラリをダウンロードしている場合は、`<sdk>/extras/android/support/design` からライブラリのリソースを確認できます。`<sdk>` は Android SDK のパスを表しています。スタイルリソースから色などを変更したい場合はこのリソースを参考にスタイルやカラーを上書きするとよいでしょう。

1.11.2 参考リンク

- Material Design のガイドライン
 - <https://www.google.com/design/spec/material-design/introduction.html>
- Android Developers Blog の Android Design Supoprt Library を紹介する記事
 - <http://android-developers.blogspot.jp/2015/05/android-design-support-library.html>
- Android Design Support Library の作者のひとりが公開しているサンプルプログラム
 - <https://github.com/chrisbanes/cheesesquare>

第2章

App Links を使ってアプリを優先的に開く仕組みを学ぶ

Android M から App Links という仕組みが使えるようになります。

2.1 App Links がない時代

Android では URL を開こうとしたときに、その URL をフックしてアプリを起動することができます（図 2.1）。たとえば Twitter アプリがインストールされている状態で、Twitter の URL を開こうとすると、ブラウザで開くか Twitter アプリで開くかダイアログ表示がありますよね。

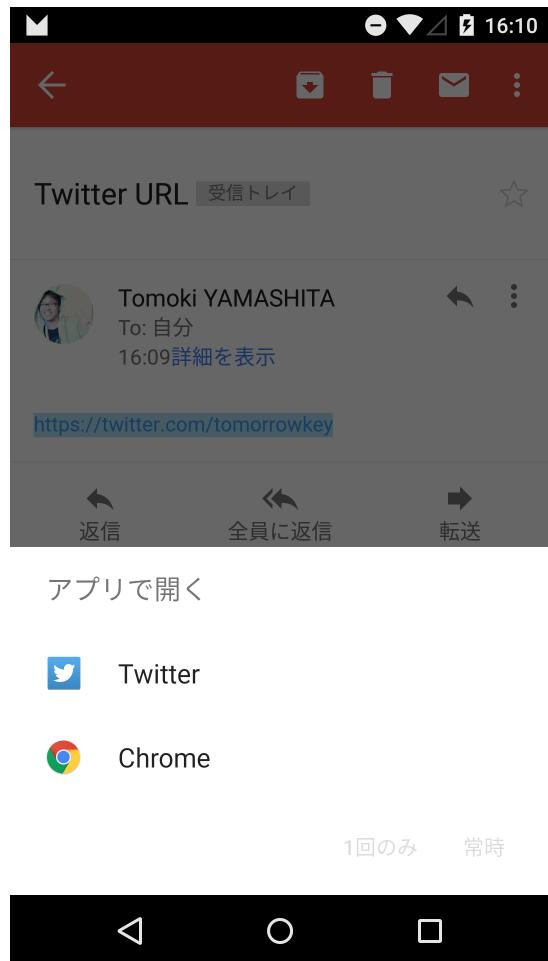


図 2.1 Twitter URL を開いたときに表示されるダイアログ

これは、開こうとしている URL のドメインが `twitter.com` だった場合に、Twitter アプリが URL をフックしてアプリを起動しようとしているのです。しかしこれは URL なので、ブラウザでも開くことができ、OS 側ではブラウザと Twitter アプリのどちらで開くべきか判断がつかず、ユーザーに選択を委ねているのです。ユーザーに操作を委ねるというのは「好きなアプリで開くことができる自由」であると同時に「その辺自動的にしてくれないイケてない仕組み」でもあります。ダイアログでアプリを選択するときに"常時"というボタンを押せば、このダイアログは二度と表示されなくなりますがこのボタンを活用しているユーザーは一体どれほど存在するでしょうか。

2.2 App Links がある時代

Android M から導入される App Links を使えばこのような問題が解決されます。App Links を使えば優先的に開くアプリをデベロッパーが宣言することができます。先ほどの例をとっ

て説明すると、Twitter の URL を開こうとした場合に、アプリを選択する画面が表示されず、Twitter アプリが起動するように設定することが可能です。

2.3 App Links の制限

アプリを優先的に起動することができると聞くと、なんと素晴らしい仕組みだろうと思うかもしれません。例に挙げた Twitter では API が公開されており、さまざまなアプリが存在します。優先的に起動できるようになるのであれば、どのアプリでもその宣言をしたいと思うかもしれませんのが App Links はサービス提供者にしか宣言できないような仕組みになっています。

App Links に対応するためにはアプリへの宣言の追加と、フックしたい URL と同じドメインに設定ファイルを置きます。後者に関してはサービス提供者でないと対応することはできないため、誰しも使える機能ではないのです。

2.4 App Links に対応する

App Links の概要はだいたいわかったと思うので、対応するための手順を見てみましょう。ここでは <http://tomorrowkey.jp> という URL をフックして起動するアプリを App Links に対応させてみます。

2.4.1 AndroidManifest.xml の変更

App Links に対応するには、まずアプリに宣言が必要です。宣言といつてもとても簡単です。App Links に対応したい intent filter に autoVerify という属性を追加するだけです（リスト 2.1）。

リスト 2.1: AndroidManifest の変更

```
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>

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

    <data
        android:host="tomorrowkey.jp"
        android:scheme="http"/>
```

```
</intent-filter>
```

2.4.2 statements.json の設置

App Links に対応する intent filter は tomorrowkey.jp というドメインの URL をフックするものなので、そのドメイン配下の/.well-known/statements.json というパスに設定ファイルを置きます。さきほど AndroidManifest の宣言に対応するためには、リスト 2.2 で示す URL で設定ファイルが読めるようにしなければなりません^{*1}。

リスト 2.2: statements.json の URL

```
http://tomorrowkey.jp/.well-known/statements.json
```

JSON ファイルには次のように記述します（リスト 2.3）。

リスト 2.3: statements.json

```
[  
  {  
    "relation": ["delegate_permission/common.handle_all_urls"],  
    "target": {  
      "namespace": "android_app",  
      "package_name": "jp.tomorrowkey.android.applinks",  
      "sha256_cert_fingerprints": ["A2:C1:...:16:E6"]  
    }  
  }  
]
```

指定しなければならないのは package_name と sha256_cert_fingerprints です。package_name には標準で起動するアプリを指定します。sha256_cert_fingerprints には key-store の fingerprint を指定します。keytool コマンドを使用して取得します（リスト 2.4）。

リスト 2.4: fingerprint の取得

^{*1} ちなみに Preview 版では https には対応しておらず、検証するには http を使用しなければなりません。正式版までには https をサポートするそうです

```
keytool -list -v -keystore ~/.android/debug.keystore
```

keystore のパスワードが求められるので入力すると fingerprint が表示されます。ちなみに開発用の keystore でも App Links は動作しました。

2.5 動作確認

アプリをインストールするとサーバへのアクセスが発生します（リスト 2.5）。

リスト 2.5: アクセスログ

```
unixtime:1435391110 datetime:27/Jun/2015:16:45:10 +0900
x-forwarded-for:- host:xxx.xxx.xxx.xxx size:324 response_time:597 status:200
server:xxx.xxx.xxx.xxx Host:tomorrowkey.jp method:GET
path:/.well-known/statements.json protocol:HTTP/1.1
UA:Dalvik/2.1.0 (Linux; U; Android M Build/MPZ440) referer:-
```

アプリがインストールされた段階で `AndroidManifest` の内容がパースされ、`autoVerify` が宣言されているので、サーバの設定ファイルをリクエストしています。この `verify` した結果はキャッシュされるらしく、同じ `IntentFilter` の内容であれば何度もリクエストされることはないそうです。

このアプリを使えば URL を開こうとした場合に、アプリを選択するダイアログは表示されずに、アプリが直接表示されるようになります。

2.6 App Links の解除

アプリの設定画面（図 2.2）を見ると、デフォルトで表示されるという設定があらかじめ有効になっています。これはデフォルトで有効になるだけで、ユーザーが明示的にオフにすることもできます。



図 2.2 アプリの設定画面

2.7 最後に

この章では App Links の紹介をしてきました。モバイルファーストと呼ばれて久しいですが、モバイルファーストを飛び越えてアプリファーストという言葉さえ最近耳に挟みますね。それは、アプリを使えばよりよい価値をユーザーに届けやすいということなんだと思います。App Links を使えばユーザーに煩わしい選択を迫ることなく、優先的にアプリを起動することができますので、積極的に活用していきましょう。

第3章

最低限知らないと事故る Auto Backup for Apps

Android M からアプリのデータディレクトリが自動的に Google Drive にバックアップされるようになります。これにより Android が壊れてしまいファクトリーリセットを余儀なくされた場合や、機種変更をして端末が新しくなった場合に、アプリの移行をスムーズにすることができます。

3.1 実は以前からあったバックアップの仕組み

実はこのバックアップの仕組み自体は目新しいものではなく、Android 2.2 から存在しました。後述しますが、Android M と Android M 以前のバックアップではバックアップしたいファイルの指定方法が異なります。えっ、じゃあ今まで使っていた設定や実装はどうすればいいの？ 新しい実装に移行しないといけないの？ と不安になると思いますが、それは今までどおりで大丈夫です。

Android 2.2 で導入されたバックアップでは、バックアップやリストアのインターフェイスだけ用意されており、サーバとの接続部分やバックアップ対象の選定などの実装はアプリ開発者に任されていました。新しく導入された Auto Backup for Apps ではその辺の実装が用意されるようになり、簡単な設定だけで済むようになったのです。つまりどちらも同じ枠組みの上に動作するので、そのまま問題ないです。

3.2 注意しなければならないこと

冒頭にも書きましたが、Android M からはアプリのデータディレクトリが自動的にバックアップされます。これだけであれば、アプリ開発者はバックアップのための複雑な処理を書かなくていいし、ユーザーは無駄なデータ入力のやり直しがなくなるしサヨーと思うかもしれません

んが、意外と考えないといけないことはあります。

3.2.1 整合性を失ってしまうファイル

端末環境に依存して生成されるデータがある場合、データがリストアされたときに整合性を失い、不具合を引き起こす原因となります。たとえば Google Cloud Messaging で使用する registration id は端末ごとに異なる値を使用しますが、これをバックアップの仕組みを使って新しい端末で再利用してしまうと、プッシュ通知が送られないという不具合になります。このように端末に依存したファイルがある場合、バックアップの対象から外すか、そもそもファイルに書き込まない設計にしましょう。

3.2.2 AccountManager との関連性

Auto Backup for Apps でバックアップされるのはあくまでデータディレクトリ配下だけなので、AccountManager の情報はバックアップされません。データディレクトリ中に AccountManager にログイン情報が存在することが前提のデータがあった場合、データがリストアされたときに整合性を失いアプリがクラッシュするということになりかねません。AccountManager を使っているアプリはこの辺にも注意しましょう。

3.2.3 ファイルサイズの制限

バックアップできるファイルサイズはアプリごとに 25MB の制限があります。25MB を越えた場合はバックアップ自体が行われませんでした。Android M Preview 1 はとても挙動が不安定でバックアップの検証が非常に難しいです。この挙動が正しくない可能性もあるので、正式版で検証をお勧めします。バックアップの恩恵にあやかるためにバックアップする必要がないファイルを除外してバックアップ対象が 25MB 以下になるようにしましょう。

3.3 バックアップされるファイル

3.3.1 バックアップの対象となるファイル

ここまでで、バックアップしない方がよいファイルはどんなものがあるか理解できたと思います。次にバックアップの対象となるファイルを確認しましょう。基本的にはアプリの次の

のようなファイルがバックアップ対象となります。

- 内部ストレージのアプリケーションディレクトリ
 - Context.getApplicationInfo().dataDir で取得できるディレクトリ
- 外部ストレージのアプリケーションディレクトリ
 - Context.getExternalFilesDir() で取得できるディレクトリ

3.3.2 バックアップの対象とならないファイル

バックアップ対象となるファイルの中でも例外的にバックアップ対象とならないディレクトリが存在します。一時ファイルやキャッシュなどはバックアップされるべきではありません。次のディレクトリ配下のファイルは自動的にバックアップ対象から外れます。

- Context.getCacheDir() や Context.getExternalCacheDir() で取得できるディレクトリ
- Context.getCodeCacheDir() で取得できるディレクトリ
- Context.getExternalFilesDir() で取得できるディレクトリ以外に存在する外部ストレージのファイル
- Context.getNoBackupFilesDir() で取得できるディレクトリ

もしこれらのファイルをバックアップの対象にしたい場合、後述するバックアップ対象への追加の設定をしましょう。

3.4 バックアップ対象のファイルの追加と除外

任意のファイルをバックアップ対象にしたり、バックアップ対象から除外したい場合、設定ファイルを書きます。

設定ファイルは xml ファイルに書きます。res/xml 配下に適当な名前で設定ファイルを追加します（リスト 3.1）。

リスト 3.1: mybackupscheme.xml

```
<full-backup-content>
<include domain=["file" | "database" | "sharedpref" | "external" | "root"]
    path="string" />
```

```
<exclude domain=["file" | "database" | "sharedpref" | "external" | "root"]  
    path="string" />  
</full-backup-content>
```

バックアップ対象のファイルを直接指定したい場合は include タグを使用します。include タグを使用した場合、それ以外のファイルはバックアップされなくなります。バックアップ対象からファイルを除外したい場合は exclude タグを使用します。include タグと exclude タグには domain と path という属性があり、それらを使って対象のファイルを示します。

domain に root を指定した場合、アプリのデータディレクトリのルートを指します。具体的には Context.getApplicationInfo().dataDir で取得できるディレクトリです。ディレクトリ配下のパスは path に指定します。

domain に files を指定した場合、アプリのデータディレクトリの files ディレクトリを指します。具体的には Context.getFilesDir() で取得できるディレクトリです。ディレクトリ配下のパスは path に指定します。

domain に database を指定した場合、アプリのデータディレクトリの databases ディレクトリを指します。具体的には Context.getDatabasePath(String) で取得できるパスです。引数は path に指定します。

domain に sharedpref を指定した場合、アプリのデータディレクトリの shared_prefs ディレクトリを指します。具体的には Context.getSharedPreferences(String) が使用され、引数は path に指定します。

domain に external を指定した場合、外部ストレージのパスを指します。具体的には Context.getExternalFilesDir() で取得できるディレクトリを指します。ディレクトリ以下のパスは path に指定します。

設定ファイルができたら、AndroidManifest に設定ファイルを指定します。fullBackupContent に xml ファイルを指定します。allowBackup が true になっていることも確認しましょう（リスト 3.2）。

リスト 3.2: AndroidManifest.xml

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

```
<uses-sdk android:minSdkVersion="MNC"/>
<uses-sdk android:targetSdkVersion="MNC"/>
<application ...
    android:fullBackupContent="@xml/mybackupscheme"
    android:allowBackup="true">
</application>
...
</manifest>
```

3.5 バックアップの検証手順

3.5.1 ログ出力の有効化

設定ファイルのパースをログ出力するためのフラグを立てます。

```
$ adb shell setprop log.tag.BackupXmlParserLogging VERBOSE
```

ログ出力を有効にするとバックアップ時に設定ファイルのパース結果を logcat で見ることができます。

```
=====
Found valid fullBackupContent; parsing xml resource.
=====
...parsed /data/data/jp.tomorrowkey.android.autobackupforapp for domain "root"

Xml resource parsing complete.
Final tally.
Includes:
 domain=r
 /data/data/jp.tomorrowkey.android.autobackupforapp
Excludes:
 ...nothing to exclude.

=====
```

3.5.2 バックアップの検証

手動でバックアップを検証するために、次のコマンドを使って Backup Manager を初期化します。

```
$ adb shell bmgr run
```

初期化が完了したら、バックアップを実行します。<PACKAGE>にはバックアップしたいアプリのパッケージ名を指定します。

```
$ adb shell bmgr fullbackup <PACKAGE>
```

3.5.3 バックアップからの復元の検証

バックアップされたデータを復元するには次のコマンドを実行します。<PACKAGE>には復元したいアプリのパッケージ名を指定します。

```
$ adb shell bmgr restore <PACKAGE>
```

この検証はアプリのデータを削除してから行って下さい。また、アプリをアンインストールしたうえでインストールしなおしても自動的にデータが復元されます。こちらの方がコマンドを使うよりより実践的な検証になるでしょう。

3.5.4 バックアップの削除

バックアップされたデータを削除するにはファクトリーリセットをするか、次のコマンドを実行します。

```
$ adb shell bmgr wipe <TRANSPORT> <PACKAGE>
```

<TRANSPORT>には Transport を指定します。Transport は実際にバックアップ処理を行ったり、サーバとのデータのやり取りをする役割があります。AOSP には LocalTransport という実装がありますので、詳しくはその実装を読んでみると実際にどのような処理をしているかが分かります。Android M Preview2 時点では AOSP 版の LocalTransport と Google の実装である com.google.android.gms.backup.BackupTransportService の2種類の transport が搭載されています。

前者の LocalTrasnport は AOSP に入っている実装なので、Google を含む特定のベンダに依存しない形で実装された transport です。特定のベンダに依存できないので、当然サーバに送信するような処理はなく、名前とおりバックアップデータをローカルに保存しておく transport です。後者の transport は Google Drive と連携するための transport です。デフォルトではこちらの transport が選択されています。端末にインストールされている transport 一覧の取得には次のコマンドを使用します。

```
$ adb shell bmgr list transports
```

使用する transport の切り替えには次のコマンドを使用します。

```
$ adb shell bmgr transport <TRANSPORT>
```

3.6 おわりに

繰り返しになりますが、AutoBackup for Apps は文字どおり自動的にバックアップされてしまうため、知らないままアプリをリリースすると、気づいたら不具合になってしまった！なんてことが起こりそうです。アプリのデータディレクトリの中にバックアップされてはいけないファイルがないかも一度確認をして Android M の正式リリースに備えましょう。

第4章

M is for MIDI

Android M では、新しく MIDI デバイスを操作するための API が追加されました。本章では、この MIDI サポートについて解説します。この API で何ができるのか？ どんな使い方が期待されているのか？ そもそも、なぜ、2015 年にもなって MIDI が必要とされているのか？ その現代的意義は何か？ といった疑問に、一定の回答を示すことができればと思います。

4.1 導入

4.1.1 2015 年における MIDI サポートの意味

19xx 年代 - MIDI の起源から全盛期まで

MIDI とは正式名称を Musical Instrument Digital Interface という規格で、楽器とコンピューターを繋ぐインターフェースを定めるものです。任意のデジタル楽器を任意のコンピューターで制御するために必要となる、楽器のセットを定義し、楽器を操作するための命令を規定しています。その歴史は古く、1983 年には技術標準として成立していました。

1980 年代当時のコンピューティング環境は、当然ながら 2015 年現在とは大きく異なっていました。インターネットや LAN はまだ一般には普及しておらず、Bluetooth も USB 規格も存在していません。

1990 年代にコンピューターが家庭に普及した頃には、それらと接続できる MIDI 楽器が登場しました。MIDI 楽器は、専用のシリアルケーブルを使用して、PC や他の MIDI 楽器と接続していました。鍵盤楽器のキーボードの一部は、人間による演奏を入力として、PC や他の MIDI 楽器に送信できる、MIDI インターフェースを備えるようになりました。また、もっぱらコンピューターで接続して音楽を演奏できる、Roland、YAMAHA、KORG などの MIDI 音源（音源モジュール）が登場しました（図 4.1 参照）。この頃は、MIDI は主にホビイストの道具であり、一般家庭にまで広く普及してはいませんでした。ソフトウェアでサンプリング音源の波形を合成して演奏しながらコンピューターを使用するのは、まだあまり実用的ではありませんでした。

んでした。

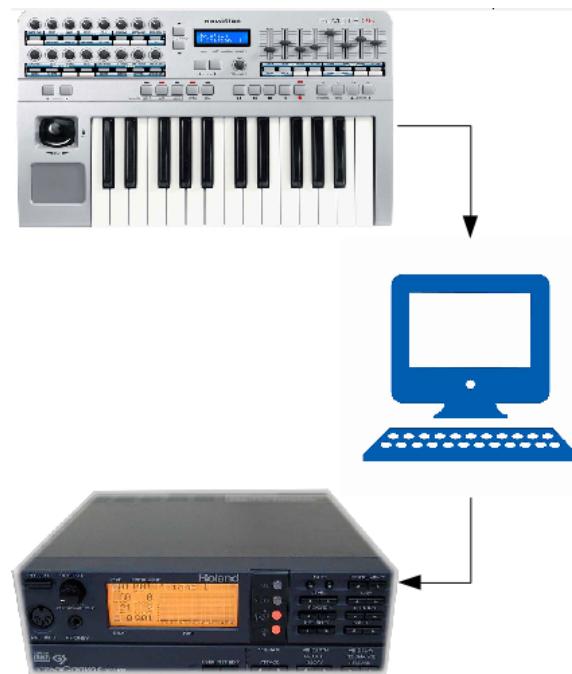


図 4.1 MIDI デバイスの接続イメージ

MIDI 規格のメリットは、128 種類もの楽器（図 4.2 参照。これは基本セットにすぎません）を、MIDI メッセージという単純な（データ量が小さい）命令によって、演奏させることができます。という点にあります。同じ楽曲でも、管弦楽団が異なれば異なって聞こえるように、MIDI 楽曲も、再生する MIDI 楽器によって聞こえ方は変わりますが、概ね同じ音楽が聴けることになります。生楽器を練習して演奏できるようになるのは大変ですし、MIDI 楽器は、使い方を少し勉強すれば、「それなり」に鳴ってくれる音を、どんな楽器でも、出せるようになります。

また、録音された演奏の記録には、膨大な情報量が必要となりますが、MIDI 楽曲の演奏データは、標準 MIDI ファイル (SMF) などの形式で、ネットワーク上で配布されていました。

| Pianos | | Chromatic Perc. | | Organs | | Guitars | |
|--------|----------------------|-----------------|--------------|--------|------------------|---------|-------------------------|
| 1 | Acoustic Piano | 9 | Celesta | 17 | Drawbar Organ | 25 | Acoustic Guitar (nylon) |
| 2 | Bright Piano | 10 | Glockenspiel | 18 | Percussive Organ | 26 | Acoustic Guitar (steel) |
| 3 | Electric Grand Piano | 11 | Musical Box | 19 | Rock Organ | 27 | Electric Guitar (jazz) |
| 4 | Honky-tonk Piano | 12 | Vibraphone | 20 | Church Organ | 28 | Electric Guitar (clean) |
| 5 | Electric Piano | 13 | Marimba | 21 | Reed Organ | 29 | Electric Guitar (muted) |
| 6 | Electric Piano 2 | 14 | Xylophone | 22 | Accordion | 30 | Overdriven Guitar |
| 7 | Harpsichord | 15 | Tubular Bell | 23 | Harmonica | 31 | Distortion Guitar |
| 8 | Clavi | 16 | Dulcimer | 24 | Tango Accordion | 32 | Guitar Harmonics |

図 4.2 MIDI 音色のリスト（の一部）

命令（図 4.3 参照）にしたがって再生する音は、単調なものであり、いつも同じ楽器を使っていたら、いつも同じような音が出ることになって退屈なものです。しかし、多様なサンプリング音源を自在に演奏できるようになるまでは、この MIDI 音源は、当時実現可能であった技術に基づいて、コストパフォーマンスのよい、かつ自由度の高い作曲環境をもたらしました。

| Status code | 命令 |
|-------------|-------------------|
| 8n | ノートオン |
| 9n | ノートオフ |
| An | ボリュームニックキー・ブレッシャー |
| Bn | コントロール・チェンジ |
| Cn | プログラム・チェンジ |
| Dn | チャネル・キー・ブレッシャー |
| En | ピッチ・ベンド |
| F0 | システム・エクスクルーシブ(開始) |
| F7 | システム・エクスクルーシブ(終了) |

図 4.3 MIDI メッセージのリスト（の一部）

20xx 年代 - MIDI の現代的意義

2015 年現在、MIDI を取り巻く環境は大きく変わりました。現在における MIDI の存在意義は、1980 年代に仕様策定された当時のものではありません。MIDI はもはや作曲における最先端技術ではなく、MIDI 楽器を使用して作った楽曲を完成品として公開するというの、何かしらの技術的な制約があるか、そうでなければほぼノスタルジーによるものでしょう。

PC では、DAW (Digital Audio Workstation、統合作曲環境) 上で、MIDI 標準に縛られない、多様なサンプリング音源を選択し合成することができます。演奏データの入力には、今でも MIDI の手法が用いられていますが、演奏に表情を加えられる多様なエフェクトは、もう MIDI 標準だけではカバーしきれません。

楽曲データは、完成品を MP3 や AAC などの圧縮 PCM データとして転送できます。ある楽器で細かくパラメーターを調整した演奏データが、他の MIDI 楽器でまったく意図したとお

りに鳴らないというのは、もう我慢する必要のない制約なのです。

では、MIDIとは、その役目を終えた、もう必要のない技術なのでしょうか? いや、実のところ、今でもMIDI技術は用いられています。ひとつには、音楽のパフォーマンスの手法が多様化した、ということが挙げられます。MIDI楽器は、伝統的には鍵盤楽器が多かったのですが、楽器とは入力あるいは演奏のためのインターフェースであり、ハードウェアとしてはあらゆる形態をとる可能性があります。

ネットワークなどでデジタルに繋がった楽器に、演奏メッセージを送受信させることで、音楽を演奏させるアーティストが、少なからずいます。そもそも、送受信されるメッセージが、「音楽の演奏」以外のオペレーションかもしれません^{*1}。ネットワーク上で演奏命令を処理する場面では、演奏に特化したソフトウェア資産の多いMIDIは、まだ有用なのです。

さまざまなガジェットでMP3などが再生できる現在、簡単に音楽を最終的な楽曲データとして再生するための仕様としては、MIDIはほぼその役目を終えています。しかし、作曲のためのツールにおいては、特にモバイルなど高度な編集には到底向いていないような環境上では、まだ利用する意味があります。作曲ツールにおける、音楽情報の大まかなフォーマット、たとえば「どのキーを、どの音量で、どれくらいの時間鳴らすか」といった基本構造は、昔も今も変わりません。たとえば、YAMAHAのVocaloidで使用されていたVSQファイルのフォーマットは、標準MIDI形式に則ったものでした^{*2}。演奏そのものではなく、演奏のタイミングに合わせてユーザー入力の処理が変わる音楽ゲームなどでも、MIDI楽曲の構造を応用できるでしょう。MIDIの基本的な機能がサポートされていることで、実現可能になる技術は、まだまだあるといえます。

4.1.2 AndroidのMIDIサポートに至るまでの状況

オーディオレイテンシーの現状

音楽アプリケーションの世界では、2015年現在でも、モバイル音楽アプリケーションは、ほぼiOSが独占していると評価するのが妥当でしょう。Androidの評価は、あまり良くありません。

主な原因是、Androidのオーディオ処理にかかるレイテンシー(遅延)にあります。低レイ

^{*1} この用途では、レコード会社と契約を結んでいるミュージシャンではなく、電子工作に携わる人が主なユーザーでしょう

^{*2} Vocaloid3になり、XML形式のフォーマットであるVSQXファイルになっても、演奏命令集合としての基本構造は変わっていません

テンシー (low latency) オーディオのパッケージを販売している superpowered が、Android のオーディオ レイテンシー測定に関するレポート^{*3}を公開していますが、Nexus 9 を使用したオーディオ バッファの再生でも 35 ミリ秒程度はかかるているようです（図 4.4 参照）。

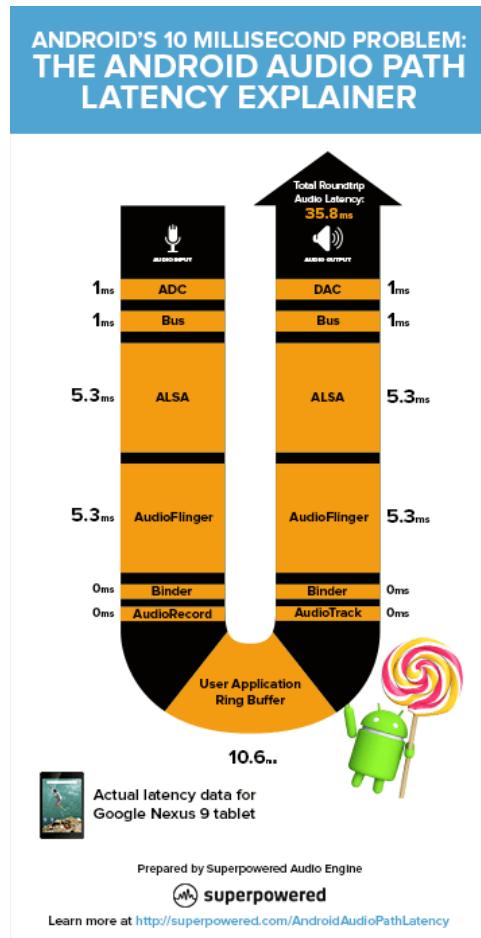


図 4.4 superpowered による Android オーディオ レイテンシー分析図

オーディオ レイテンシーの問題は、Android 2.3 における OpenSL ES の導入、Android 4.1 におけるデバイス別最適オーディオ パラメーター取得 API の導入^{*4}、Android 4.2 における低レイテンシー (low latency) のオーディオ プレイバックの実現^{*5}、Android 4.4 におけるハードウェア DSP のサポート^{*6}、Android 5.0 における float オーディオのサポート^{*7}など、

^{*3} <http://superpowered.com/androidaudiopathlatency/>

^{*4} <http://createdigitalmusic.com/2012/07/android-high-performance-audio-in-4-1-and-what-it-means-plus-libpd-goodness-today/>

^{*5} <http://developer.android.com/about/versions/jelly-bean.html#android-42>

^{*6} <http://www.androidpolice.com/2013/10/31/kitkat-feature-spotlight-audio-tunneling-to-dsp-dramatically-reduces-battery-consumption-when-playing-audio/>

^{*7} <http://geeknizer.com/audio-improvements-in-android-5-0-l-audiophile/>

段階的に改善されてきました。

それでもなお、最善の一般的な Android アプリケーションの環境でも、30-50ms のレイテンシーが発生している状況であり、概ね 10ms 以下のレイテンシーを実現している iOS 環境と比べると、まだまだ改善が期待されている状況です。

新しい MIDI 環境の登場

一方で、音楽アプリケーションの実行環境としても実用性を示した iOS では、Android に先んじて MIDI のサポートを追加し、既存の MIDI デバイスとの接続性も確保しつつ、BLE (Bluetooth Low Energy) など、新しい技術との統合にもチャレンジしてきました。BLE などを活用した、iOS アプリケーションと連携するデバイスは、技術的には Android とも接続可能であるにもかかわらず、iOS 専用のデバイスとして発売されている状態です。

こういう楽器デバイスが、Android で利用できないのは、もったいないことです。

MIDI のメッセージングは、オーディオ合成のようなレイテンシーが問題になることは無いようです^{*8}。Android のフレームワークのレベルで、MIDI の基本的な接続がサポートされるようになれば、これらの楽器デバイスを活用できる可能性が広がります。

MIDI 楽器の使い方はさまざまです。入力に使用する楽器もあれば、出力として接続する音源モジュールもあります。20世紀のように音楽の演奏に MIDI ファイルを使用する機会は、そうそうありません。特に最初から演奏内容が決まっている場合は、ソフトウェアでリアルタイム合成するよりは、最初から生成済みの PCM オーディオ楽曲を再生すれば事足ります。しかし、演奏内容が決まっていない場合や、ユーザーの入力に応じて音を変化させる必要がある場合には、演奏に近いかたちで音楽を再生する必要があるでしょう。

現状では Android のハードウェアに MIDI 音源がバンドルされていることはないので^{*10}、ユーザーの特殊なデバイス環境に依存しない音楽演奏のためには、ソフトウェア音源を使用するしかありません。このような場合には、MIDI 音源として機能するソフトウェアシンセサイザーが求められるでしょう。そして、音声をソフトウェアでリアルタイム合成する場合には、オーディオ レイテンシーが関わってくるのです。

本稿執筆時点の 2015 年 6 月現在、Android 上で動作するソフトウェア MIDI シンセサイザーは、筆者の知るところではまだありません。これには、オーディオ レイテンシーの他に

^{*8} stackoverflow にある「オーディオ処理の絡まない MIDI メッセージングでも遅延は発生するか?」という質問と回答^{*9}が参考になります。

^{*10} 2000 年代の日本の携帯電話には、SMAF という着メロ用ファイルを再生できる YAMAHA の LSI もありました

も、Linuxにおける一般的なオーディオ バックエンドであった ALSA や Jack の不在、Javaにおけるネイティブライブラリ相互運用（JNI）の困難など、さまざまな原因があると考えられます。iOS には、Roland の Sound Canvas for iOS が存在しています。Android も、このMIDI API を端緒として、この方面で iOS に追いつくことが期待されます。

Web MIDI API の登場

Android M が 2015 年にもなって MIDI をサポートするようになったもうひとつの背景が、Web 環境におけるオーディオ アプリケーションの進化です。HTML5 の進化がデスクトップ アプリケーションの機能を Web に持ち込んでいったように、Web ブラウザと JavaScript の環境でオーディオ合成などを扱うものとして、Mozilla の Audio Data API や Google の Web Audio API が登場しました。これらを実用化するために、ブラウザ上でも低レイテンシー オーディオが扱えるようになってきたことから、デスクトップ環境専用だったサウンドアプリケーションが、Web でも少しずつ見られるようになりました。

Web ブラウザでも MIDI を使えるようにしたい、という要求が上がってきたのも、この延長線上にあるといえます。MIDI 楽器をブラウザからシームレスに使えるようになれば、アプリケーションも Web ブラウザ上でクロスプラットフォームに実現できます。これを実現するものとして、W3C 仕様のドラフトとして、Web MIDI API が標準化されようとしており、Chrome 上で実装されるようになりました。

現在では、Android 版 Chrome でも、r43 以降は Web MIDI API を使用できます。Chromium にも含まれており、AOSP にも含まれています。Android M 以前でも使用できます。ただし、Android M Preview では、Chrome の Web MIDI API サポートと相性が悪く、Chrome 側の MIDI サポートが無効になってしまい、というレポートがあるので、注意が必要です。Chrome 開発チームには既知の問題であり、対応予定であることが表明されています。

Chrome は Android でも動作することを前提としたブラウザであり、Web Audio API や Web MIDI API の実装を経て、Android 自体にも、ネイティブレベルで、MIDI API を実現する環境が整ってきたといえます。

■コラム: JETPlayer

Android の MIDI サポートは Android M 以降ですが、M 以前にも、MIDI に類似する機能は存在していました。API としては JETPlayer という機能で公開されていました。内部的に

は Sonivox というソフトウェアを使用していて、これは実体としては DLS (Downloadable Sounds) を使用した MIDI のソフトウェア シンセサイザーのようなものです。JETPlayer は、単なる音楽演奏のための環境というよりは、ゲームなどインタラクティブに音楽を再生する目的で作られています。JETCreator を使用して、複数の音楽ファイルをストックして、場面に応じてその場で切り替える、といった使い方が想定されています。JETPlayer 自らが標準 MIDI ファイルを直接サポートしているわけではありませんが、JET フォーマットのヘッダーに SMF の内容をくっつけるだけで、JETPlayer で再生できるファイルを作ることは可能でした。MIDI を使用したい音楽アプリケーション開発者の中では、この Sonivox でもいいので使用したいという声はかつてはそれなりにありました。Android issue #8201 は^{*11}、古くから存在する、Android で MIDI をサポートしてほしい、という要望のスレッドですが、初期の要望は sonivox の API をパブリックにしてほしいというものでした。

4.1.3 Android M の MIDI サポート概要

Android M で、初めてフレームワークレベルで MIDI のサポートが実現したことになりますが、その内容は、MIDI デバイス実装をシステムで管理し Android サービスとして提供する仕組み、MIDI デバイス実装の基盤となる API、そして MIDI over BLE および USB MIDI のデバイス実装、とまとめられます。デバイスにほぼ直結した生の MIDI API に近いといえます。高水準のシーケンサー API などは存在しません^{*12}。

Android M Preview の時点では、MIDI サポートの部分は AOSP に存在せず、Android SDK コンポーネントのソース パッケージをインストールするしかありません。一方で、AOSP には MIDI に関するデバイスベンダー向けのページがすでに公開されています^{*13}。本稿でも、ソースコードで実装を追っている部分は、Android SDK コンポーネントのものを典拠としています。

Android M Preview のサンプルコードにも、MIDI API に関するものは存在せず、まだ Web 上でもこの API の使用例は皆無であったため、本稿で書かれている内容は、android.media.midi パッケージの概要で説明されているもの以外は、コードを書いて筆者の手

*12 従来の Android でも、MIDI ファイルは、android.media.MediaPlayer で再生することができます

*13 <https://source.android.com/devices/audio/midi.html>

元にある Nexus 9 上で動作確認したものが中心となります。

4.2 MIDI デバイスを使用する

4.2.1 MIDI API の概要

ここからは、Android M の新しい MIDI API の使い方を主軸において、具体的に解説していきます。この MIDI の API は、パッケージ `android.media.midi` として追加されています。

Android M 時点での MIDI API には、大きく分けて 2 つの機能カテゴリーがあります。アプリケーション開発者として「MIDI デバイスを使用する」ための API と、MIDI デバイスの提供者として「MIDI デバイスサービスを実装する」ための API です。MIDI を利用したアプリケーションを開発する場合は前者を、MIDI ソフトウェア シンセサイザーなどを公開して、他のアプリケーションで使用してもらう場合は後者を、それぞれ使うことになります。

この節では、MIDI デバイスを「利用する」ための API を中心に説明していきます。デバイスを提供するための API については、次の節で扱います。

どちら側の API にも共通することですが、MIDI API は、「MIDI デバイス」が複数の「入力ポート」「出力ポート」をもち、それらを開いて「MIDI メッセージ」を送受信するものです。この API の構造は、Android 以外の MIDI フレームワークにも共通する、一般的なものといえます。

MIDI デバイスを利用するアプリケーションの基本的な流れは、次のようになるでしょう。

- MIDI デバイスのリストを取得して、そのひとつをオープンする
- そのデバイスの MIDI 入力ポートあるいは MIDI 出力ポートをオープンする
- MIDI 入力デバイスを使用するなら、入力ポートからの MIDI メッセージを適宜処理する
- MIDI 出力デバイスを使用するなら、音を鳴らしたい（あるいは何らかのイベントを発生させたい）タイミングに合わせて、MIDI メッセージを送信する

Android M で追加された MIDI API は、あくまで基本的な MIDI デバイスの接続を可能にするもので、高水準の処理を行う機能（たとえば標準 MIDI ファイルの再生）はありません。それらは、アプリケーション開発者の実装に委ねられています。

4.2.2 利用可能なデバイスの情報を取得する

MIDI デバイスのリストと hotplug

Android MIDI API で MIDI デバイスを使用するには、リスト 4.1 のように、まずそのためのシステムサービスを `android.content.Context` クラスの `getSystemService()` メソッドから取得します。その戻り値は `MidiManager` クラスのインスタンスとなります。

リスト 4.1: `MidiManager` によるデバイスリストの取得

```
MidiManager manager = ((Context) this).getSystemService();
MidiDeviceInfo [] devices = MidiManager.getDevices();
```

`MidiManager.getDevices()` メソッドによって、現在接続されているデバイスの「デバイス情報」のリストが取得できます。MIDI 楽器が接続されておらず、MIDI デバイスサービス（後述）も何もインストールされていない状態では、本稿執筆時点での Android M Preview では、ここで返されるリストは空っぽです。

上記の方法では、あくまで `MidiManager.getDevices()` を呼び出した時点でのデバイスのリストが返されるのですが、`MidiManager.registerDeviceCallback()` を使用して、あとから接続されたデバイスを検出したり（hotplug）、デバイスの削除を検出したりすることも可能です。このメソッドに、`MidiManager.DeviceCallback` を渡すと、このオブジェクト上の、各デバイス接続イベントのコールバック メソッドが呼び出されます。

`MidiManager.getDevices()` の戻り値、あるいは `MidiManager.DeviceCallback` クラスの各メソッドによって渡されるパラメーターは、`MidiDeviceInfo` というクラスのオブジェクトになります。これは、「実際にデバイスをオープンして使用する」ときに使われる `MidiDevice` というクラスとは異なり、あくまで接続状態を含まない、デバイス情報のみを保持しています。

MIDI デバイスの種類については、あとで詳述しますが、ここでは次の 3 種類があることを記しておきます。

- USB MIDI デバイス
- BLE MIDI デバイス
- 仮想 MIDI デバイス

MIDI デバイス情報の詳細

`MidiDeviceInfo` には、その MIDI デバイスの詳細情報が含まれています。

MIDI 楽器には、伝統的に製造者 (manufacturer) や製品名などのメタ情報が存在します。これらは、`MidiDeviceInfo.getProperty()` メソッドで `android.os.Bundle` を取得し、その `getString()` メソッドを呼び出すかたちで取得できます。その引数には `MidiDeviceInfo.PROPERTY_*` フィールドが便利でしょう。

また、`MidiDeviceInfo.getPorts()` メソッドを使うと、そのデバイスのポート情報のリストを取得できます。このメソッドが返す `MidiDeviceInfo.PortInfo` クラスでは、ポート種別 (入力・出力)、ポート名、ポート番号が取得できます。

4.2.3 MIDI デバイスと MIDI ポートをオープンする

MIDI デバイスをオープンする

`MidiDeviceInfo` の情報をもとに、使用するデバイスが決まつたら、次は、リスト 4.2 のように、`MidiManager.openDevice()` メソッドを使って、そのデバイスをオープンします。

リスト 4.2: `MidiManager` でデバイスをオープンする

```
midiManager.openDevice(deviceInfo, new MidiManager.OnDeviceOpenedListener () {  
    @Override  
    public void onDeviceOpened(MidiDevice device) {  
        // 任意のアプリケーション コード  
    }  
}, null);
```

このメソッドは、デバイスをオープンする処理を非同期で開始し、自身はすぐに終了します。デバイスがオープンされたら、上記の `onDeviceOpened()` メソッドが呼び出され、接続状態になったデバイスをあらわす `MidiDevice` オブジェクトが利用可能になります。もしこのコールバック メソッドを、特定のスレッド コンテキストで呼び出したい場合は、`android.os.Handler` を `MidiManager.openDevice()` メソッドの 3 番目の引数として渡します。

MIDI ポートをオープンする

いったん `MidiDevice` が利用可能になったら、今度はリスト 4.3 のように、そのデバイスのポートを選択してオープンします。

リスト 4.3: MIDI ポートを開く

```
MidiInputPort inPort = midiDevice.openInputPort(inPortNumber);  
MidiOutputPort outPort = midiDevice.openOutputPort(outPortNumber);
```

使いたいポートのポート番号は、`MidiDeviceInfo.getPorts()` の戻り値から知ることができます。

MIDI ポートには入力と出力の 2 種類があり、それぞれが複数存在します。ひとつのデバイスが両方を有していることもあります。1 つの MIDI ポートで利用できる MIDI 出力デバイスの最大チャネル数は 16 なので、同時利用チャネル数が 16 以上ある MIDI 音源モジュールでは、複数のポートを公開しています。たとえば、ソフトウェアシンセサイザーの `timidity` には、出力ポートが 4 つあります。

MIDI アプリケーションは、上記のメソッドで返される `MidiInputPort` と `MidiOutputPort` を使用して、MIDI メッセージをやり取りすることになります。

他のプラットフォームでの MIDI API の使用経験がある人は、ここで注意すべきことがあります。Android M (少なくとも Preview 時点) における Input と Output の意味は、Web MIDI API や ALSA、CoreMIDI、WinMM などの、一般的な MIDI API のものとは真逆であるということです（図 4.5 参照）。



図 4.5 MIDI IN/OUT と `MidiInputPort/MidiOutputPort` の関係

MIDI デバイスの気持ちになって考えると、Input と Output の意味は逆になるので、「Input や Output の意味は、実は曖昧である」ということはできますが、筆者が知る範囲では、MIDI API としては、Android の API の方向性に近いのは、Java MIDI API (javax.sound.midi) のみです。

Android M Preview developer issue #2538 として変更要望を登録したので^{*14}、正式版ではもしかしたら逆になっているかもしれません、少なくとも現状の API では注意が必要です。

ちなみに、応用的な機能として、`MidiDevice.connectPorts()` というメソッドを使うと、任意の MIDI 入力ポートを、このデバイスの出力ポートに、ブリッジ接続することができます。

4.2.4 MIDI メッセージを送受信する

input、output、receiver、sender

いったん MIDI 入出力ポートを開いてしまえば、あとは MIDI メッセージを送受信するだけです。リスト 4.4 に示すメソッドを使用します。メッセージはバイト列でやり取りし、そこにタイムスタンプを指定することもできます。

リスト 4.4: ポートでメッセージを送受信する

```
MidiInputPort.send(byte[] msg, int offset, int count, long timestamp)  
MidiOutputPort.connect (MidiReceiver receiver);
```

`MidiOutputPort` を使用した MIDI IN デバイスからのメッセージ取得は、少し説明が必要そうです。`MidiReceiver` とは、MIDI メッセージを受信するためのクラスです。MIDI メッセージを受信したい場合は、リスト 4.5 のように、このクラスを派生させて、その `onSend()` メソッドに自分の行いたい処理を記述します。

リスト 4.5: `MidiReceiver` の使い方

```
MidiReceiver receiver = new MidiReceiver () {  
    @Override  
    public void onSend(byte[] msg, int offset, int count, long timestamp)  
    {
```

*14 <https://code.google.com/p/android-developer-preview/issues/detail?id=2538>

```
// アプリケーション コード  
}
```

いったん MidiReceiver のオブジェクトを作成したら、それを MidiOutputPort.connect() に渡すと、その MIDI 入力デバイスから送られてきたメッセージが、この MidiReceiver.onSend() の呼び出しとして届く、という流れです。

実は MIDI 出力を処理する実体である MidiInputPort も、この MidiReceiver から派生しています。逆に、MIDI 入力を処理する実体である MidiOutputPort は、MidiSender から派生しています。したがって、この MIDI 入力となる MidOutputPort に、MIDI 出力となる MidInputPort を connect() メソッドで接続することもできます。

MidiOutput に接続した MidiReceiver は、MidiOutputPort.disconnect() メソッドで接続解除できます。

タイムスタンプ

MIDI メッセージの送信、受信に共通する話ですが、MIDI メッセージに付随して、タイムスタンプが送信されてくることがあります。これは、MIDI OUT (をあらわす MidiInputPort) に接続した MidiSender にとっては「メッセージを処理すべき名目上の時間」であり、MidiReceiver を接続された MIDI IN (をあらわす MidiOutputPort) にとっては「メッセージを送信した名目上の時間」です。値としては java.lang.System.nanoTime() の戻り値が渡されます。

タイムスタンプは、さまざまな MIDI API に存在する概念で、あらゆるレベルで発生しうる MIDI メッセージのレイテンシー（遅延）を、プログラムのレベルで吸収する目的で、メッセージ本体と一緒に渡されるものです。Android MIDI API の実装でも、com.android.internal.midi.MidiEventScheduler という MIDI イベント処理のスケジューラーが存在し、USB MIDI と BLE MIDI の実装については、これが指定されたタイムスタンプの値にしたがってイベントを処理しています。

その他のメッセージング詳細

Android M における MIDI メッセージングの内部実装は、Android SDK コンポーネントのソースコードで見ることができますが、基本的には Android サービスとして設計されている MidiDeviceService (次の節で詳しく扱います) に接続して、Intent を送受信することになり

ます。

ちなみに、いったん `MidiDevice.connectPorts()` で `MidilInputPort` を出力ポートに接続すると、その `MidilInputPort` の `onSend()` メソッドはもう呼ばれなくなるので、注意が必要です^{*15}。

USB MIDI と BLE MIDI の実装で使用されている `MidiEventScheduler` は、実装のソースコードを見た限りでは、キューに登録されたメッセージが 200 件を超えると、新しいメッセージを受理せずに捨ててしまいます。通常の音楽演奏でそこまで同時にイベントがキューイングされることはないと思いますが、汎用メッセージング機構として使っている場合には、注意が必要かもしれません。

4.2.5 MIDI アプリケーションのサンプル

本文ではプログラムを掲載しませんが、`MidiManager` を使用したごく簡単な MIDI API のサンプルを、Web 上で掲載しています。スクリーンのほぼすべての領域にひとつだけの `SurfaceView` を配置し、パッドとしてタッチイベントに応じてピアノの音を鳴らすだけのアプリケーションです。動作を確認するには、USB MIDI デバイスあるいは MIDI デバイスとなるサービスがインストールされている必要があります。タッチパッドに触れると Note On メッセージ、離れると Note Off メッセージを送ります。

このサンプルコードは、MIDI API の使い方を示す目的で作られた、ごく単純なものであり、MIDI デバイスの選択、BLE MIDI デバイスの接続、音色の選択（プログラムチェンジ）、複数ノートメッセージ、画面へのフィードバックなど、さまざまな機能に対応していません。いろいろ機能追加の余地があるので、興味のある方は、各自試してみて下さい。

4.3 MIDI デバイスを作成して公開する

前節では、MIDI デバイスに接続して使用するアプリケーション側の API について説明してきました。ここからは、Android M の MIDI API における、MIDI デバイスの「提供者」が使用する機能、MIDI デバイスサービスについて説明していきます。

MIDI デバイスサービスの API を使用すると、任意の Android アプリケーションで使用できる、仮想的な MIDI デバイスを作成できます。

*15 それならば、`MidiReceiver` が `connect` されている `MidilInputPort` は引数に指定できない仕様にできたのではないか、とも思えますが、内部的に実現できないのかもしれません

4.3.1 誰がこの API を使うべきか

Android M でサポートされる MIDI デバイスは、次の 3 種類です。

- USB MIDI デバイス
- BLE MIDI デバイス
- 仮想 MIDI デバイス

このうち、USB MIDI デバイスと BLE MIDI デバイスの接続管理は、Android フレームワークが行うものであり、デバイスサービスを作成する必要はありません。仮想 MIDI デバイスを作成したい人は、このデバイスサービス API を使用します。

MIDI デバイスサービスを使用して作成された MIDI デバイスは、Android サービスとしてデバイス全体に公開されます。もし任意のアプリケーションから使用されることを前提としているのであれば、MIDI デバイスサービスを使用するべきではありません。

アプリケーションによっては、単に MidiSender あるいは MidiReceiver の実装を作って、同じアプリケーション内部で使用するだけで十分です。あるいは、MidiDeviceService ではない、別のプライベートな Service として登録して接続することもできるでしょう。

その意味では、一般的な MIDI デバイスとして振る舞うことが期待されている、といえますが、実際の MIDI デバイスでも、特殊なサウンドの生成を行うものはあります（たとえば VOCALOID などがそうです）。あくまで「任意のアプリケーションに利用されることを意図している（それで問題ない）」かどうかで、このデバイスサービスにするかどうかを判断すればよいと思います。

4.3.2 MIDI デバイスの種類

デバイスサービス API の具体的な説明に入る前に、Android M がサポートする MIDI デバイスについて、各論的に説明します。

(1) USB MIDI

USB MIDI デバイスを Android ホストに接続していれば、そのデバイス上で `MidiManager.getDevices()` を呼び出したときに、その USB MIDI デバイスをあらわす `MidiDeviceInfo` が返されます。

Android がフレームワークのレベルで USB デバイスを MIDI モードで接続して認識できるのは、Android M 以降のみですが、Android L 以前でも、`android.hardware.usb` の API を使用して、USB MIDI デバイスを利用可能にするアプリケーションを作ることは可能であり、実際に Google Play でもいくつか公開されています。これらは、USB MIDI デバイスとしての接続とメッセージングに必要な処理を、自前で行っているものです。

次は、筆者が動作確認のため、Android M Preview をインストールした Nexus 9 に、USB ホストケーブルを経由して、MIDI デバイスを接続してみたデバイスです。

- M-AUDIO KeyStation Mini 32 (MIDI キーボード) - 成功
- スイッチサイエンス eVY1 (YAMAHA NSX-1, 歌唱シンセサイザー/音源モジュール)
- 成功
- Roland SC-8820 (音源モジュール) - 失敗

この結果からは、入力、出力、いずれのデバイスであっても、正しく認識しています。SC-8820 はやや古いデバイスであり、認識されなくても仕方ないようにも思えますが、サードパーティの USB MIDI アプリケーションには認識されるようなので、Android M Preview issue #2527 としてレポートしてあります^{*16}。

(2) MIDI over BLE

MIDI over BLE は、BLE のプロトコル上で MIDI メッセージを送受信する仕様です。Android M 以降は、MIDI over BLE に対応した MIDI 楽器（となるデバイス）があれば、それを接続して利用することができます。

MIDI over BLE には、実はまだ正式な仕様はありません。BLE は比較的新しい仕様であり、Android API としても、対応しているのは 4.3 以降のみです。楽器デバイスベンダー向けに Apple が規定した仕様が元になっています^{*17}。Android SDK コンポーネントで取得できるソースの中に、`com.android.bluetoothmidiservice.BluetoothMidiService` というクラスがあり、そこでは Apple の仕様で定義されている GATT および characteristics と同一の UUID が定義されています。Android M における MIDI over BLE の実装は、この仕様に準拠しており、iOS に対応している MIDI over BLE デバイスであれば、仕様上は Android M でも接続できるはずです。

*16 <https://code.google.com/p/android-developer-preview/issues/detail?id=2527>

*17 <https://developer.apple.com/bluetooth/Apple-Bluetooth-Low-Energy-MIDI-Specification.pdf>

筆者が確認した範囲では、mi.1 (Wireless MIDI interface) と、第3世代 iPad 上でアドバータイズした midimittr が、Nexus 9 から接続できました (mi.1 は Roland SC-8820 に接続しました)。

(MIDI over BLE をサポートしている Android アプリケーションは、[BLE MIDI for Android](#) をはじめ^{*18}、Google Play 上でもいくつか散見されるのですが、筆者の手元には Android M Preview をインストールできるデバイスが Nexus 9 の1台しかなく、これ以前の Android では BLE ペリフェラルモードで MIDI デバイスをアドバータイズできないため、Android 同士での接続は確認できませんでした。)

BLE MIDI デバイスの注意点ですが、Android システム上で単に MIDI over BLE のデバイスをペアリングしても、自動的に MIDI デバイスとして利用可能になるわけではありません。BLE デバイスを MIDI 楽器として認識させるには、いったん Bluetooth API で android.bluetooth.BluetoothDevice のインスタンスを取得してから、それを MidiManager.openBluetoothDevice() メソッドに渡して、MIDI デバイスとして登録する必要があります。リスト 4.6 に例を示します。

リスト 4.6: BLE-MIDI デバイスをオープンする

```
BluetoothManager manager =
    (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
BluetoothAdapter adapter = manager.getAdapter();
BluetoothLeScanner scanner = adapter.getBluetoothLeScanner();
scanner.startLeScan(this); // ここでは this が下記インターフェースを実装

// BluetoothAdapter.LeScanCallback の実装
@Override
public void onLeScan (BluetoothDevice device, int rssi, byte[] scanRecord) {
    MidiManager midi = (MidiManager)getSystemService (Context.MidiService);
    // ここでは this が下記インターフェースを実装
    midi.openBluetoothDevice(device, this, null);
}

// MidiManager.OnDeviceOpenedListener の実装
@Override
public void onDeviceOpened(MidiDevice device) {
    // デバイスオープン後の処理 (もしあれば)
}
```

*18 <https://github.com/kshoji/BLE-MIDI-for-Android>

いったん BluetoothManager 経由で MIDI デバイスとして認識されれば、そのサービスインスタンスが実行している間は、認識させるコードを再度実行する必要はありません（デバイスを再起動したら、Bluetooth サービスも再起動するので、また登録する必要があります）。

(3) 仮想 MIDI デバイス

最後は、ソフトウェアレベルで実装される「MIDI デバイス」です。

仮想 MIDI デバイスの典型的な例は、やはりソフトウェア シンセサイザーでしょう。他には、USB でも BLE でも接続できない、何らかの特別な楽器デバイスを繋いで MIDI メッセージに反応するソフトウェアなどが考えられます。そのソフトウェアが「MIDI メッセージを受け取って、何らかの処理を行う」ものとして作成されてさえいれば、MIDI デバイスとして認識させることは可能なので、実際には楽器として機能する必要はまったくありません。音楽に合わせて発生するイベントのトリガーは、インタラクティブ アートの道具としても有用かもしれません。

ただし、本節の冒頭でも言及しましたが、MIDI デバイスサービスは、あくまで「任意のアプリケーションからリクエストされて使用されることを想定している」仮想 MIDI デバイスのためにあるものです。特定のアプリ間のみでメッセージをやり取りするのであれば、この MIDI デバイスサービスはあまり適切とはいえません。

ソフトウェア シンセサイザーを仮想 MIDI デバイスとして実装する（あるいは既存の実装を利用する）場合は、オーディオ処理のパフォーマンスという、困難な課題に直面することにもなるでしょう。

4.3.3 MidiDeviceService クラス

仮想 MIDI デバイスを公開するために必要な手順は、次のとおりです。

1. MidiDeviceService クラスの実装を作成する
2. AndroidManifest.xml で<service>として記述する
3. デバイス情報を記述する XML ファイルを作成する

(1) MidiDeviceService クラスの実装を作成する

MidiDeviceService は、onGetInputReceivers() という abstract メソッドをもつていて、これを実装する必要があります。ひとつの仮想的な MIDI OUT デバイスをもち、その MidiReceiver を返すような MidiDeviceService のコードは、リスト 4.7 のようになります。

リスト 4.7: MidiDeviceService クラスを実装する

```
public class MyDeviceService extends MidiDeviceService
{
    MidiReceiver receiver = ... // Midi OUT として機能する MidiReceiver の実装

    @Override
    public MidiReceiver [] onGetInputPortReceivers ()
    {
        return new MidiReceiver[] { receiver };
    }
}
```

(2) AndroidManifest.xml で<service>として記述する

AndroidManifest.xml に追加する内容は、概ねリスト 4.8 のようになります。

リスト 4.8: AndroidManifest.xml に service を追加する

```
<service android:name="仮想デバイスのクラス名"
         android:label="@string/service_name">
    <intent-filter>
        <action android:name="android.media.midi.MidiDeviceService" />
    </intent-filter>
    <meta-data android:name="android.media.midi.MidiDeviceService"
              android:resource="@xml/device_info" />
</service>
```

この例では、string/@service_name と@xml/device_info が指定されているので、前者は strings.xml に、後者は res/xml/device_info.xml としてデバイス情報を定義したファイルを作成します。

(3) デバイス情報を記述する XML ファイルを作成する

デバイス定義ファイルの内容は、リスト 4.9 のような XML になります（Android SDK のドキュメントとほぼ同様の内容です）。

リスト 4.9: デバイス定義 XML を作成する

```
<devices>
  <device manufacturer="製造者名" product="製品名">
    <input-port name="ポート名" />
    <output-port name="ポート名" />
  </device>
</devices>
```

デバイスサービスの実装

MidiDeviceService は、MIDI 入力ポートと MIDI 出力ポートを提供します。ただしそれを具体的に実現する仕組みは、入力ポートと出力ポートで少し異なります。

(1) <input>ポート - MIDI OUT として機能する MidiReceiver

まず、自分のデバイスでサポートする MidiInputPort（前節で説明したとおり、一般的には MIDI OUT として接続できるポート）は、MidiDeviceService.onGetInputPortReceivers() メソッドを実装するかたちで行います。先の MidiDeviceService クラスのひな形の部分に相当します。このメソッドの仕事は、MidiReceiver[] を返すことだけであり、実装作業は、ここで返される MidiReceiver 実装の作成が中心となるでしょう。

もしこの仮想 MIDI デバイスの receiver が、ネイティブリソースなどを管理しているのであれば、それらがリークしないように、onDeviceStatusChanged() メソッドや onDestroy() メソッドをオーバーライドして、開放処理をしっかり行う必要があります。

このデバイスが提供する MidiInputPort は、デバイス定義 XML の<input>要素で定義したとおりになります。複数ポートを用意することも可能です。複数ポートをサポートするのであれば、その数だけ MidiReceiver を返す必要があります（もし<input>で定義したポートの数が実際に onGetInpputPortReceivers() の数より大きいと、クライアントがそのポー

トを開いたときに返される `MidiInputPort` が `null` になります^{*19})。

(2) <output>ポート - MIDI IN に接続された `MidiReceiver`

一方、MIDI IN として機能する `MidiOutputPort` については、`MidiDeviceService` でオーバーライドすべきメソッドは見当たりません。ではどうやって MIDI IN を実装するのでしょうか？

基本的に MIDI イベントをクライアントに送信するタイミングやメッセージは、`MidiDeviceService` の実装が決ることです。この `MidiDeviceService` は、他の入力デバイスからイベントを受け取って、それを MIDI イベントにしてクライアントに送るのかもしれませんし、自分でタイマーなどを保持して、任意のタイミングで自ら生成してクライアントに送るのかもしれません。

以上を踏まえると、`MidiDeviceService` の実装に必要なのは、あくまで MIDI IN ポート(をあらわす `MidiOutputPort`) に接続したクライアントである `MidiReceiver` のリストのみである、ということになります。この「自身の MIDI IN ポートに接続している `MidiReceiver` のリスト」は、`MidiDeviceService.getOutputPortReceivers()` メソッドを呼び出すことで、取得できます。

では `MidiSender` はどうやって生成されているのでしょうか？もともとクライアントには `MidiSender` を直接渡すことではなく、あくまで `MidiDeviceService` は `Intent` を送るのみです。そして MIDI IN を表す `MidiOutputPort` は、デバイス定義 XML で記述した<output>要素の定義にしたがって生成されます。言い換えれば、デバイス定義 XML に<output>要素を記述しておけば、クライアントがうまいこと `MidiOutputPort` として利用できるように、`MidiDeviceService` が宜しく取り計らっている、ということです（図 4.6 参照）。

*19 これは Android 側のフレームワークがクライアントの責任にしないよう、もう少し配慮すべきところだとは思います

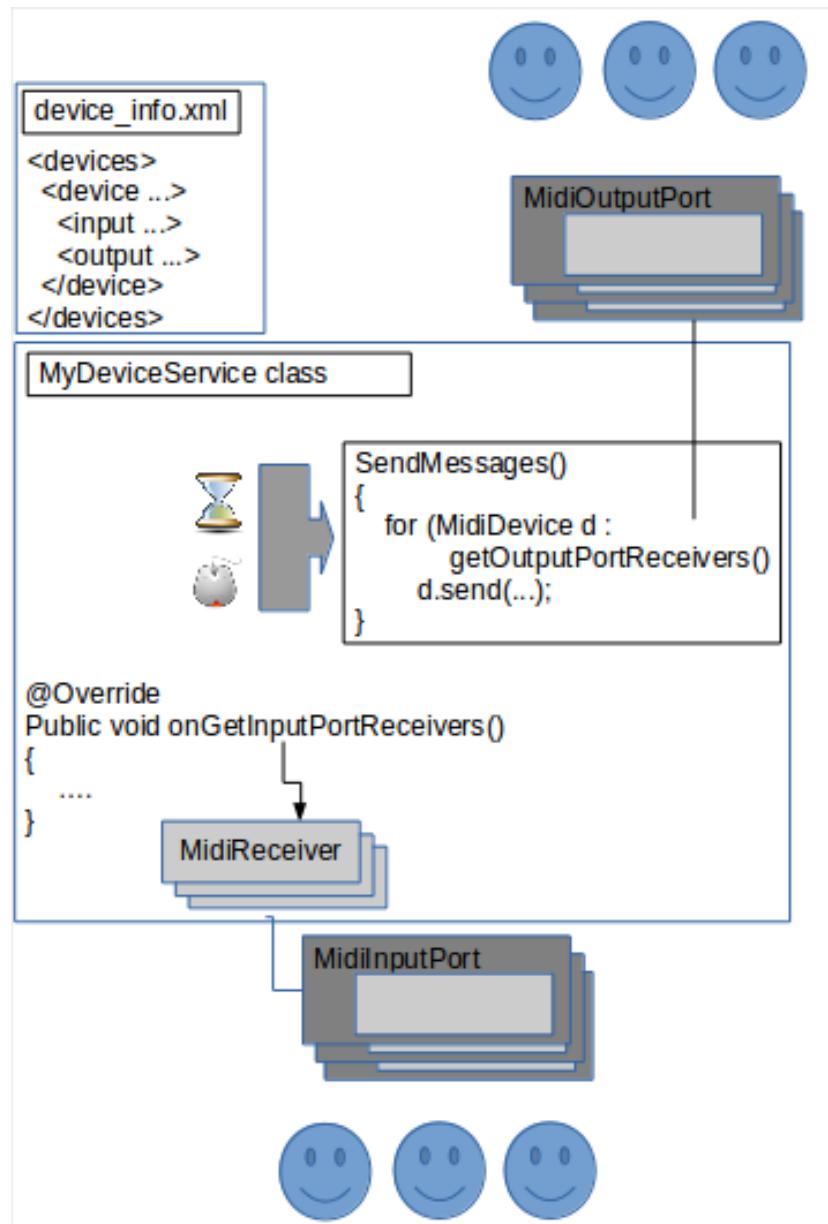


図 4.6 MIDI デバイスサービスのポート

MidiOutputPort については、開発者側でポート数と処理する MidiReceiver の数を合わせることはありませんが、`MidiDeviceService.getOutputPortReceivers()` メソッドは、(ソースコードを見た限りでは) `null` を返すこともあるので、注意が必要です。戻り値の配列の要素が `null` になることは無いようです^{*20}。

*20 あくまで M Preview の実装の話であり、将来的には変わる可能性もあります

4.4 最後に

本章では、Android M に追加された MIDI API について、この API が登場した歴史的背景などをふまえつつ、その使い方を中心的に説明してきました。Android における音楽アプリケーションは、まだ茨の道ではありますが、この MIDI サポートのような新しい API を上手く使いこなすことで、少しずつハードルが下がっていくとよいと思います。

第5章

Android Wear 5.1.1

Google I/O 2014 で Android Wear が発表されてから、一年が経過しました。

発表されて間もない頃は、各地で勉強会が開催されたり、いろんなアプリが公開されて、注目を浴びていた Android Wear ですが、最近ではあまり名前を聞かなくなつたように感じます。

みなさん Android Wear 使っていますか？

Android Wear は 2015 年春に Android バージョン 5.1.1 に更新され、Wearable Support Library は 1.2 に更新されました。

Android Wear 5.1.1 の主なアップデート内容は次のとおりです。

- 手首をひねるジェスチャーで通知カードをめくれるようになった
- Always-on アプリ（アプリのアンビエントモード）
- 手書き入力で絵文字が入力できるようになった
- Wi-Fi サポート

このアップデートの中で私が楽しいなと思っているのは、Always-on アプリです。

本章では、Always-on 対応の仕方と Wearable Support Library に追加された View について解説します。

5.1 Always-on アプリを実装する

これまで Android Wear アプリは立ち上げてしばらく操作しないでいると、アプリが強制終了されてウォッチフェイスに戻ってしまいましたが、Android 5.1.1 からはアプリもウォッチフェイスと同様に、アンビエントモード（図 5.1^{*1}）に移行できるようになりました。これが Always-on 機能です。

^{*1} 画像は Google 公式ブログより - <http://googleblog.blogspot.jp/2015/04/android-wear-wear-what-you-want-get.html>

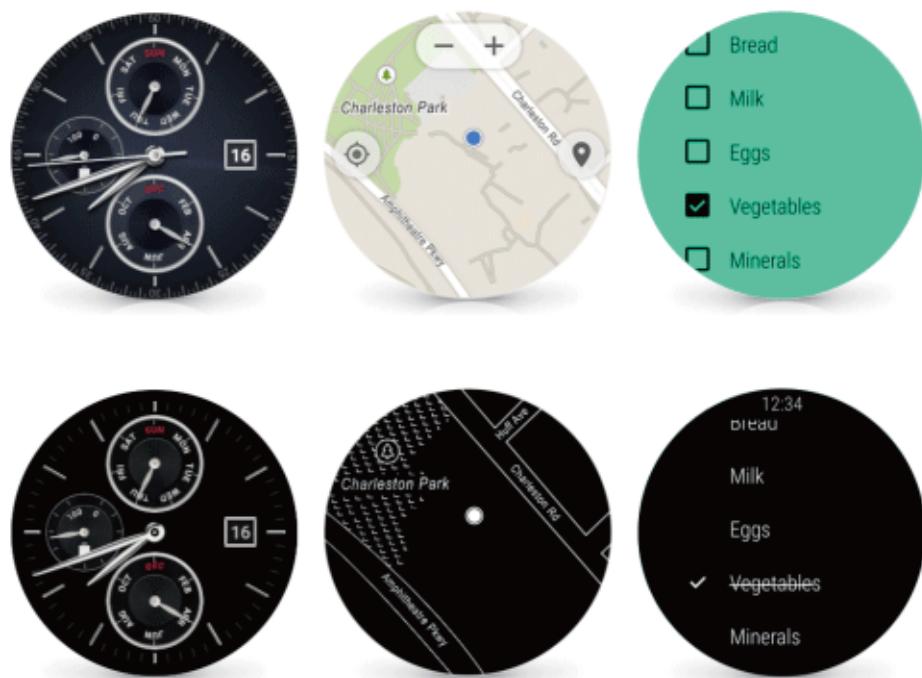


図 5.1 アンビエントモード

アンビエントモードとは、アプリがしばらく操作されてないときに、画面を完全に消灯するのではなく明るさを落としてバッテリーの消費を抑えつつ画面を表示したままにする状態のことをいいます。

アプリのアンビエントモードの画面表示のガイドラインはウォッチフェイスと同様です。アンビエントモードの間、アプリは黒背景に文字や画像は白でアンチエイリアスを無効にした画面を表示するようにします。

アプリのアンビエントモードは Google Keep、Google Maps などで確認できます。

アンビエントモードを試す際は、デバイスの設定の「常に画面表示 ON」にチェックを入れてください。「常に画面表示」がオフだと、アンビエントモードが有効にならず、通常どおり画面が消灯します。

また、Always-on は Andorid Wear 5.1 から導入された機能ですので、Android Wear 5.0 以下のデバイスではこの機能は使えません。

画面を長時間表示しておきたいアプリはアンビエントモード対応するとよいです。

ここでは Wear アプリをアンビエントモード対応させる方法について解説します。

5.1.1 プロジェクトの設定

wear/build.gradle に次の設定を追加します（リスト 5.1）。

リスト 5.1: 依存関係を追加

```
dependencies {  
    compile 'com.google.android.support:wearable:1.2.0'  
    provided 'com.google.android.wearable:wearable:1.0.0'  
}
```

次に AndroidManifest を編集します。リスト 5.2 を追加します。

リスト 5.2: ライブラリの設定を追加

```
<application>  
    <uses-library android:name="com.google.android.wearable" android:required="false" />  
    ...  
<application>
```

Android 5.1 未満をサポートする場合は false、Android 5.1 のみサポートする場合は true を設定してください。

最後に、アプリを点灯したままにするために WAKE_LOCK のパーミッションを追加します（リスト 5.3）。

リスト 5.3: WAKE_LOCK のパーミッションを追加

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

これでプロジェクトの準備ができました。

5.1.2 アンビエントモードに対応した Activity を作成する

アンビエントモードのイベントを取得するために、WearableActivity を継承したクラスを用意します（リスト 5.4）。

リスト 5.4: Activity を作成する

```
public class MyActivity extends WearableActivity {
```

WearableActivity はアンビエントモードの切り替わりを含むライフサイクルを通知してくれます。

それから、アンビエントモードを有効にするために、onCreate メソッド等で setAmbientEnabled メソッドを呼びます（リスト 5.5）。

リスト 5.5: アンビエントモードを有効にする

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

これでアプリがアンビエントモードに移行できるようになりました。

5.1.3 アンビエントモードになったとき

アプリがアンビエントモードになったときは、onEnterAmbient メソッドが呼ばれます。

ここで、背景を黒色、文字や画像は白色でアンチエイリアスをオフにしたものに更新しましょう（リスト 5.6）。

リスト 5.6: アンビエントモードになったとき

```
@Override  
public void onEnterAmbient(Bundle ambientDetails) {  
    super.onEnterAmbient(ambientDetails);  
  
    mStateTextView.setTextColor(Color.WHITE);  
    mStateTextView.getPaint().setAntiAlias(false);  
}
```

ちなみに、エミュレータでアンビエントモードを確認するには、端末の設定で Always-On Screen を On にしてから、F7 キーを押します。F7 キーでアンビエントモードと通常モードが切り替わるようになっているので、アンビエントモードの確認が簡単にできます。

5.1.4 アンビエントモードから抜けたとき

アプリがアンビエントモードから抜けたときは、onExitAmbient メソッドが呼ばれます。ここで、画面を通常の表示に更新しましょう（リスト 5.7）。

リスト 5.7: アンビエントモードから抜けたとき

```
@Override  
public void onExitAmbient() {  
    super.onExitAmbient();  
  
    mStateTextView.setTextColor(Color.GREEN);  
    mStateTextView.getPaint().setAntiAlias(true);  
}
```

5.1.5 アンビエントモードかどうか確認する

現在アンビエントモードかどうか確認するには、isAmbient メソッドを呼びます（リスト 5.8）。

リスト 5.8: アンビエントモードかどうか確認する

```
if (isAmbient()) {  
    // アンビエントモード  
} else {  
    // 通常モード  
}
```

5.1.6 一分毎にアンビエントモードの画面を更新する

バッテリーの消耗を抑えるために、アンビエントモードの間は頻繁に画面を更新してはいけません。アンビエントモードの間の画面更新は、一分に一回が推奨されていて、そのためのコールバックメソッドが用意されています。`onUpdateAmbient` メソッドはアンビエントモードの間、一分毎に呼ばれます。このメソッドを利用して画面更新をするといいでしょう（リスト 5.9）。

リスト 5.9: 一分毎にアンビエントモードの画面を更新する

```
@Override  
public void onUpdateAmbient() {  
    super.onUpdateAmbient();  
  
    // アンビエントモードの画面を更新する  
}
```

5.2 Wearable Support Library に追加された View

Wearable Support Library 1.1、1.2 で新たに追加された View は次のとおりです。

- CircularButton
- ActionLabel
- ActionPage
- DotsPageIndicator
- WearableFrameLayout

ここからは、Wearable Support Library に追加された View について解説していきます。

5.2.1 CircularButton

`CircularButton` は、`FloatingActionButton` の Wearable 版の View です（図 5.2 およびリスト 5.10）。



図 5.2 CircularButton

リスト 5.10: CircularButton

```
<android.support.wearable.view.CircularButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:color="@color/blue"  
    android:src="@drawable/action_reply"  
    app:rippleColor="@color/ripple_blue" />
```

FloatingActionButton と同様にタップしたときにリップルアニメーションします。

5.2.2 ActionLabel

ActionLabel は特殊なテキスト表示用 View です。設定した最大行数 (maxLines) と View の幅の制約を満たす、最大の文字サイズを計算して、その計算した文字サイズでテキストを表示します。

ActionLabel は次の属性がサポートされていて、TextView と同じように使えます（リスト 5.11、図 5.3）。

- text
- textColor

- maxLines
- minTextSize
- maxTextSize
- lineSpacingExtra
- lineSpacingMultiplier
- fontFamily
- typeface
- textStyle
- gravity

リスト 5.11: ActionLabel に短いテキストを設定した場合

```
<android.support.wearable.view.ActionLabel  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:maxLines="1"  
    android:text="あなたと Java"  
    android:textColor="@color/white" />
```



図 5.3 ActionLabel に短いテキストを設定した場合

リスト 5.11 のテキストをもう少し長いものに変更してみると（リスト 5.12）、図 5.4 のように文字サイズが自動で調整されることが分かります。

リスト 5.12: ActionLabel に長いテキストを設定した場合

```
<android.support.wearable.view.ActionLabel  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:maxLines="1"  
    android:text="あなたと Java 今すぐダウンロード"  
    android:textColor="@color/white" />
```



図 5.4 ActionLabel に長いテキストを設定した場合

5.2.3 ActionPage

ActionPage は、CircularButton と ActionLabel を持っている View です。図 5.5 のような Android Wear の通知カードを右に送ったときの画面を再現できます（リスト 5.13）。



図 5.5 ActionPage

リスト 5.13: ActionPage

```
<android.support.wearable.view.ActionPage
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/action_page"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:color="@color/blue"
    android:maxLines="1"
    android:src="@drawable/action_reply"
    android:text="@string/action_reply"
    app:rippleColor="@color/ripple_blue" />
```

ActionPage は、GridViewPager のページとして使うといいでしょう。

5.2.4 DotsPageIndicator

DotsPageIndicator は GridViewPager 用のインジケータです。Pager をセットして使います（図 5.6、リスト 5.14）。

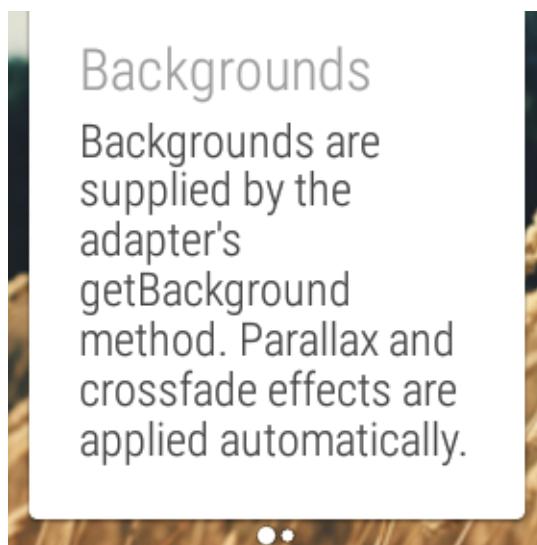


図 5.6 DotsPageIndicator

リスト 5.14: DotsPageIndicator

```
<android.support.wearable.view.GridViewPager  
    android:id="@+id/pager"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />  
  
<android.support.wearable.view.DotsPageIndicator  
    android:id="@+id/indicator"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_horizontal|bottom" />
```

DotsPageIndicator をプログラムから記述する方法は、次のとおりです（リスト 5.15）。

リスト 5.15: DotsPageIndicator

```
GridViewPager pager = (GridViewPager) findViewById(R.id.pager);  
DotsPageIndicator dotsPageIndicator =  
    (DotsPageIndicator) findViewById(R.id.indicator);  
dotsPageIndicator.setPager(pager);
```

5.2.5 WearableFrameLayout

WearableFrameLayout は丸型のデバイスで実行されたときのみ適応されるレイアウト属性を定義できる ViewGroup です。

FrameLayout と同じ挙動をしますが、後述する"~Round"属性を設定すると丸型のデバイスで実行されたときのみ、対応するレイアウト属性の値を上書きします。

これを使うと、丸型画面で View が見切れないように丸型デバイスのときのみ余白を多く設定することができます。

たとえば、リスト 5.16 のように TextView を配置すると、丸型のデバイス実行したときに図 5.7 のように見切れてしまします。

リスト 5.16: FrameLayout を使った場合

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:background="@color/blue"
        android:text="@string/text" />

</FrameLayout>
```

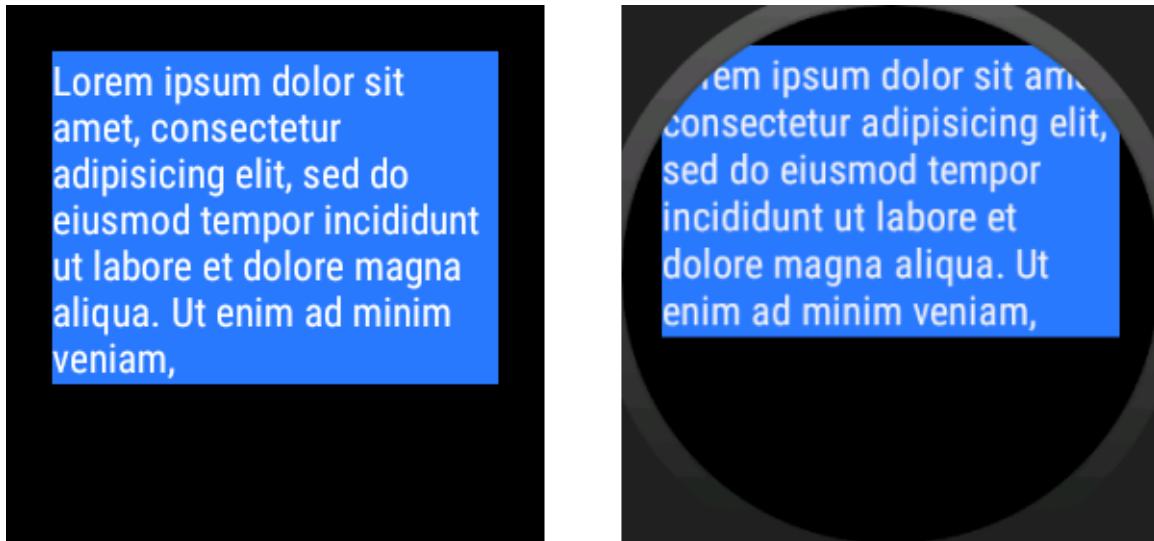


図 5.7 WearableFrameLayout

そこで、WearableFrameLayout を使って layout_marginRound で丸型画面用に余白を設定します（リスト 5.17）。

リスト 5.17: WearableFrameLayout

```
<android.support.wearable.view.WearableFrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:background="@color/blue"
        android:text="@string/text"
        app:layout_marginRound="32dp" />

</android.support.wearable.view.WearableFrameLayout>
```

TextView を画面内に収めることができました（図 5.8）。

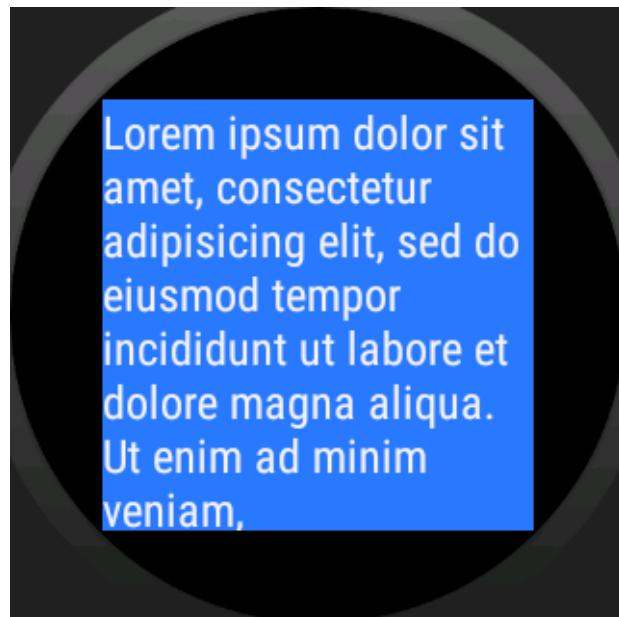


図 5.8 WearableFrameLayout2

例に出した以外にも次の属性を設定することができます。

- layout_gravityRound
- layout_widthRound
- layout_heightRound
- layout_marginRound
- layout_marginLeftRound
- layout_marginTopRound
- layout_marginRightRound
- layout_marginBottomRound

5.3 最後に

Play Store に Android Wear アプリを 3 つ配信している著者にとって、Wearable Support Library のアップデートは非常に嬉しいものでした。なぜなら、欲しいと思っていた View が提供されたからです。ActionPage など、今まで自分で Google 製のアプリの View を真似をして、既存の View を組み合わせて実装していたのが、公式から提供されて良かったです。

Android Wear も確実に進化していっているので、これからも Android Wear に注目してい

きたいです。

第6章

Android Studio で NDK ライフを送ろう

5月のGoogle I/OでAndroid StudioのNDK対応が発表されてからずっとNDK対応するする詐欺を1ヶ月半されてきましたが、7月10日のアップデート(Android Studio 1.3 RC1)でようやく待望のNDK対応が行われました。

長かったですね。僕はアップデートを256回くらいチェックしたと思います。ですが待望のこの機能は現時点非常にクセが強く、ハマりどころがそこかしこにあるので、快適にNDK!するためのいくつかのポイントを説明します。

そしてこの原稿が「そろそろ執筆完了かな？」と思っていた7月19日にRC2がアップデートされました。タイミングがいいですね！チクショウ！！

6.1 開発環境の準備

6.1.1 Canary Channel アップデートを行う

まずは覚悟を決めましょう。具体的には、「もしかしたら今の開発環境ぶつ壊れるんじゃないかな？でもまあいいかな？いいよね！！」みたいな覚悟です。

Android Studioはインストール後にアップデートを行うことができますが、このアップデートには3種類のチャンネルが用意されています。その中のひとつが、"Canary Channel"です。これは開発したての超最新機能を利用できますが、その代わり「人柱」になることを宿命付けられるものです。不具合の内容は軽微なものから、「そもそもビルドができなくなつた！」というものまであります。

この文章を書いている2015年7月上旬現在、NDK開発機能を利用するためにはCanary版を利用しなければなりません。

次の手順でCanaryを受取ります。本記事はMac版を前提としますので、他のOSの場合は適宜読み替えてください。

Android Studioを起動し、"Check for Updates"を開きます。そうすると、アップデート

第6章 Android Studio で NDK ライフを送ろう

チェックを行い、図 6.1 のようなダイアログが開きます。

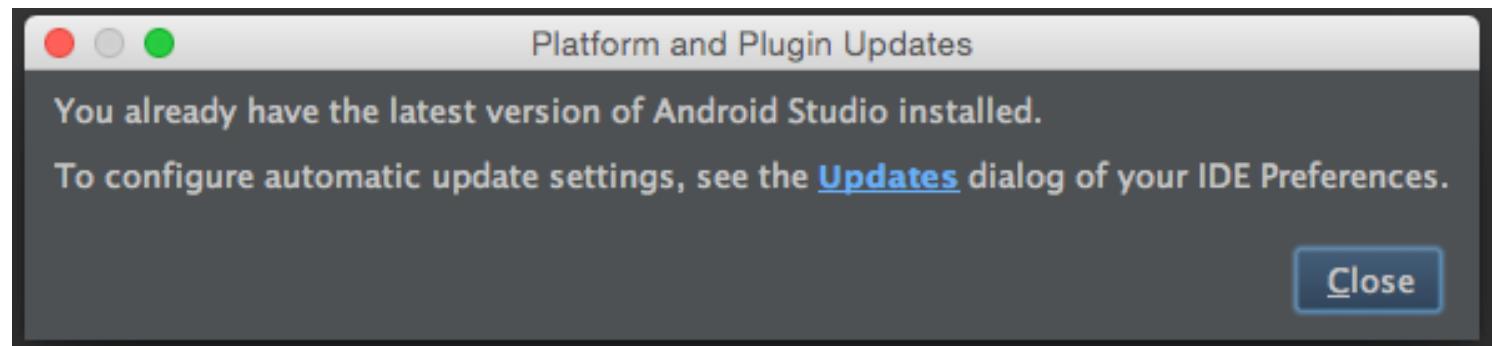


図 6.1 アップデート確認ダイアログ

ハイライトされている"Updates"をクリックして、右上のセレクトボックスの項目を"Canary Channel"に変更しましょう。再度アップデートを行うと、RC2 をインストールできます（図 6.2）。

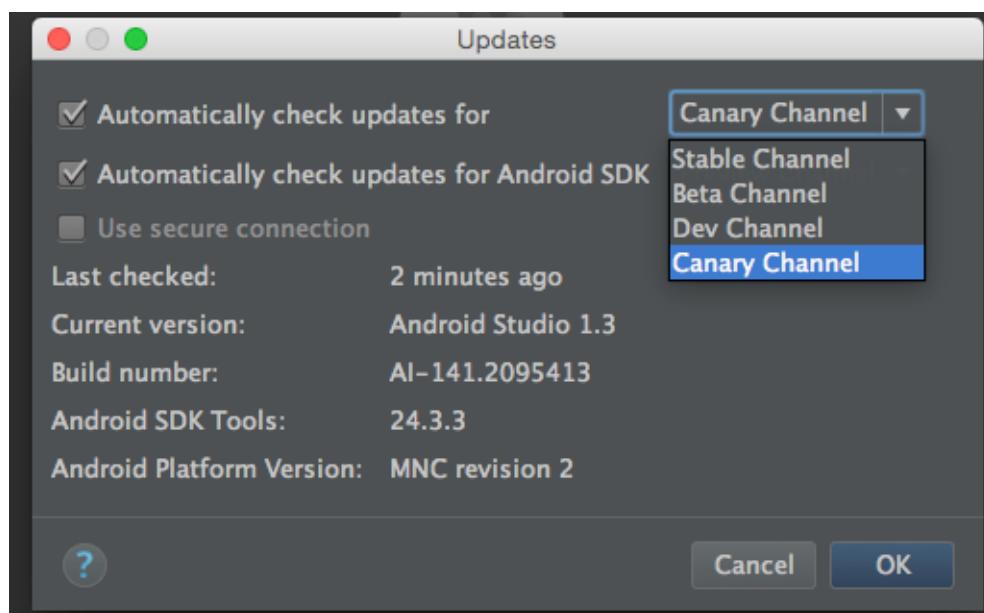


図 6.2 Canary Channel ダイアログ

6.1.2 これは Java8 ですか？ いいえ、これは Java7 です

NDK 機能を利用するためには、Java7（JDK1.7 系列）が必要です。ちなみに Java7 はすでに公式アップデートを終了していますので、最終バージョン 7u79^{*1}をインストールするといいでしょう。

もし自分の環境が Java8 である場合、追加でインストール（すでにインストール済みの場合は環境変更）を行う必要があります。システム全体の書き換えを行ってしまうと不便ですので、ここでは Android Studio が利用する JDK バージョンだけ書き換えましょう。

ただし、Android Studio の設定を変更した場合、コマンドラインでビルドを行った場合に失敗する（環境が Java8 のまま）可能性があることに注意してください。また、将来的には（時代の流れ的にも）Java8 への対応が行われることでしょう。

もしも今現在利用しているプラグインの中に Java8 でビルドされているものがあったばあい、「NDK 機能を使うためにプラグインを諦める」もしくは「プラグインを使うために NDK 機能を諦める」のどちらか、もしくは「Java7 でビルドし直す」のいずれかを選択しなければなりません。

6.1.3 Android NDKr10e のインストール

NDK 機能を利用するためには、r10e 以上のバージョンの Android NDK が必要です。NDK は単体でインストールするか、Android SDK 付属のものを利用することができます。今後のアップデートは Android SDK Manager を経由して受け取ることもできますので、特別な理由がない限りは素直に SDK 付属版を利用するとよいでしょう。

Preferences → Appearance & Behavior → System Settings → Android SDK を開き、SDK Tools のタブから Android NDK をインストールします。

6.2 build.gradle を大胆に、ときに繊細に書き換えよう

この時点でそこそこの時間をかけてしまっているかもしれません、まだ足りません。何故なら、build.gradle を大幅に書き換える必要があるからです。

^{*1} JDK1.7 は <http://www.oracle.com/technetwork/jp/java/javase/downloads/jdk7-downloads-1880260.html> からダウンロード可能です

Android NDK 対応は"Experimental"です。これはまだ正式な機能に格上げされておらず、その名のとおり試験的に運用を開始したものです。従来の Android プラグインとは build.gradle の記述方法が変更されているため、実案件で使うと将来痛い目に会うかもしれません。また、確認した限りいくつかのバグがまだあります。

まずは新しい Plugin のための classpath を修正しなければなりません。複数プロジェクトを扱う場合は従来のプラグインとも共存できますので、ビルトの構成に合わせて好きな方を選択してください（リスト 6.1 およびリスト 6.2）。

また、Gradle のバージョンが 2.5 以上に引き上げられています。Gradle 2.5 のインストールも忘れないようにしましょう。

リスト 6.1: build.gradle buildscript 部

```
buildscript {  
    repositories {  
        jcenter()  
    }  
    dependencies {  
        // 新バージョンのみを利用する  
        classpath 'com.android.tools.build:gradle-experimental:0.1.+'  
    }  
}
```

リスト 6.2: build.gradle buildscript 部（従来と共存する場合）

```
buildscript {  
    repositories {  
        jcenter()  
    }  
    dependencies {  
        // 従来のバージョンと共存する  
        classpath 'com.android.tools.build:gradle-experimental:0.1.+'  
        classpath 'com.android.tools.build:gradle:1.3.+'  
    }  
}
```

新しいPluginでは、`android`ブロックの構成がほぼ「別物」というレベルで変わっています。たとえば従来は"compileSdkVersion 22"のように記述していた部分が、"compileSd-

第6章 Android Studio で NDK ライフを送ろう

`kVersion = "22"`のように`"=`"で記述するようになりました。ですが、すべての設定がそうなのか？ というとそうではなくて、一部は従来のようにメソッドのままなので、統一感がありません。今後変更される可能性が大きいあります。

百聞は一見にしかずで、新たな構成の `build.gradle` をみてみましょう。次のようになっています（リスト 6.3）。

リスト 6.3: android プラグインの設定

```
// Plugin 名が変わっている
apply plugin: 'com.android.model.application'

// 全体が model ブロックで囲われている
model {
    android {
        // 設定を "=" で行う
        compileSdkVersion = 22
        buildToolsVersion = "23.0.0 rc3"

        // ".with"がブロックについている
        defaultConfig.with {
            applicationId = "com.example.eaglesakura.ndk.helllo_ndk"
            minSdkVersion.apiLevel = 15
            targetSdkVersion.apiLevel = 22
            versionCode = 1
            versionName = "1.0"
        }

        // パッケージングの除外ブロックには".with"が付いているが、
        // 内部は従来のままで"="は使用しない。
        packagingOptions.with {
            exclude 'LICENSE.txt'
            exclude 'META-INF/LICENSE'
            exclude 'META-INF/LICENSE.txt'
            exclude 'META-INF/NOTICE'
            exclude 'androidannotations-api.properties'
            exclude 'services/com.fasterxml.jackson.core.JsonFactory'
            exclude 'services/com.fasterxml.jackson.core.ObjectCodec'
        }
    }

    // NDK のビルド設定を android.ndk ブロックに記述する
    android.ndk {
        moduleName = "app"
```

```
}

// フレーバーを追加する
android.productFlavors {
    // 特定の abi だけビルドしたい
    create("arm7") {
        ndk.abiFilters += "armeabi-v7a"
    }

    // 全部の abi をビルドしたい
    create("fat") {
    }
}

}
```

だいたい似ているように見えて、よく見ると別物ですね。ブロック内に設定できる内容等は執筆時点ではほとんどドキュメント化されていないため、かなりの部分を手探りで変更する必要があります。特にパッケージング除外の"exclude"部が変則的である点等、ハマりどころ（もしくはテキトーにされているところ）が多く見受けられます。

前述のように build.gradle を書き換えると、ようやく Android Studio は NDK のビルドを行ってくれるようになります。

Android Studio の親切なのは、Java 言語側で"native"宣言されているメソッドが NDK 側で未実装だった場合に警告を出してくれる点です。そのため、シグニチャの記述ミス等の単純ミスを減らすことができるでしょう。

この状態でビルドを行うと、NDK 側を自動的にビルド＆パッケージングを行ってくれます。

6.3 デバッグをしようじゃないか

前述の状態では、まだデバッグを行うことはできません。この状態でドキュメントとおりに実行しようとしても、エラーが表示されて先に進むことができません。

ドキュメントには記載されていませんでしたが、build.gradle を修正して、NDK 側のデバッグオプションを ON にしなければなりません。

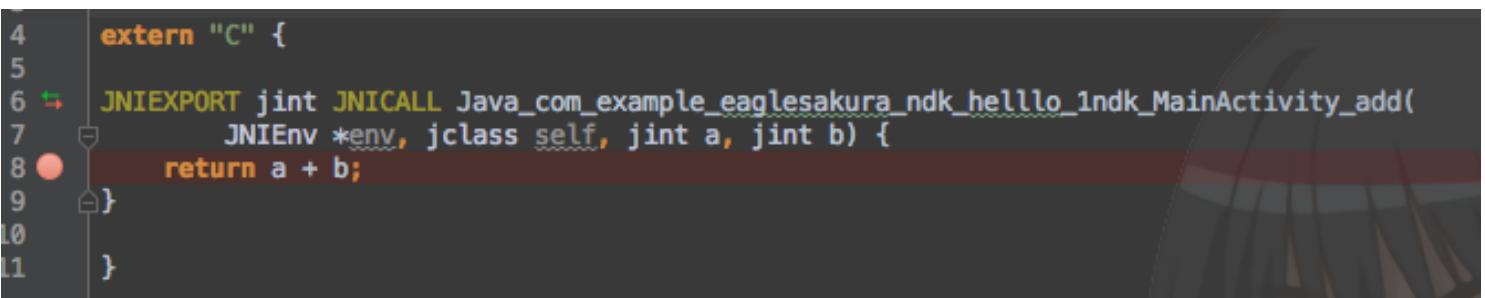
android.buildTypes ブロックを新たに追加し、"isJniDebuggable = true" の設定を追加することでようやく Android Studio はデバッグ情報付きでビルド＆実行を行ってくれるようになります（リスト 6.4）。

リスト 6.4: NDK 側のデバッグを ON に切り替える

```
model {
    android {
        中略...
    }

    // debug ビルドでは NDK 側のデバッグを ON にする
    android.buildTypes {
        debug {
            isJniDebuggable = true
        }
    }
    ... 中略...
}
```

デバッグの方法は Java と同じで、Android Studio で任意の行をクリックすることでブレークポイントを ON / OFF することができます。また、NDK のデバッガは値のチェックだけでなく、動的にメモリ内容の書き換え（変数内容の書き換え）をサポートしているので、非常に便利で強力な機能といえるでしょう（図 6.3、図 6.4、図 6.5）。

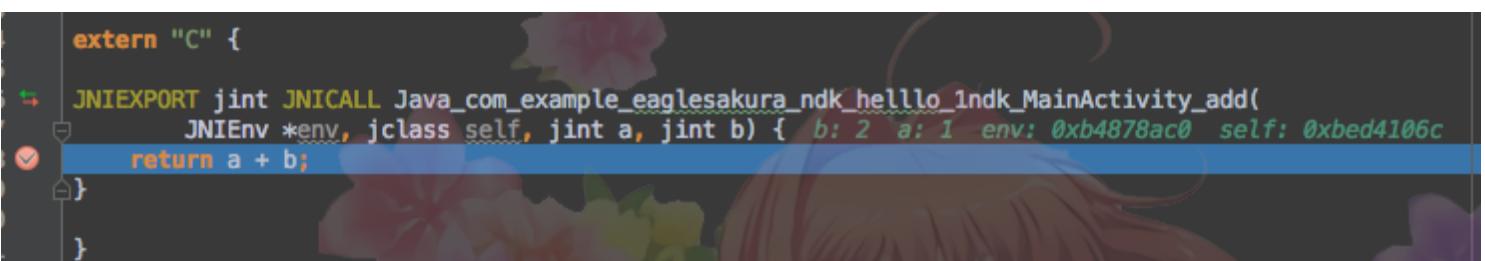


The screenshot shows a code editor window with a dark theme. A C file is open, showing the following code:

```
4 | extern "C" {
5 |
6 | ↗ JNIEXPORT jint JNICALL Java_com_example_eaglesakura_ndk_helllo_1ndk_MainActivity_add(
7 |     JNIEnv *env, jclass self, jint a, jint b) {
8 |     ● return a + b;
9 | }
10| }
11| }
```

A red dot, representing a breakpoint, is placed on the line containing the return statement. The line numbers 4 through 11 are visible on the left.

図 6.3 任意の箇所にブレークポイントを設定



The screenshot shows the Android Studio debugger interface. A C file is being debugged, with the same code as in the previous screenshot. A red dot indicates a breakpoint has been hit. The status bar at the bottom shows the current values of variables: b: 2, a: 1, env: 0xb4878ac0, and self: 0xbed4106c.

図 6.4 ブレークポイントを通過するとプログラムを停止させられる

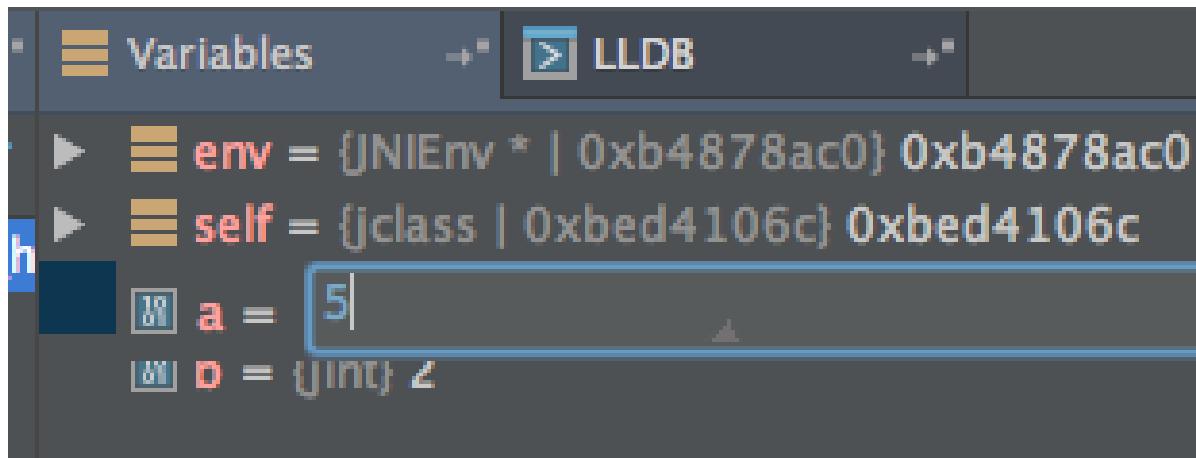


図 6.5 実行時にメモリを強制的に書き換えられる

ただし、NDK のデバッガは Java 側ほど統合されたものではないため、実行開始～デバッガが利用可能になるまでタイムラグが（最大 10 秒程度）あります。たとえば Activity の onCreate メソッドのように起動直後に呼びされるメソッド内で NDK 側の処理を行っている場合、このままではデバッグが非常に難航します（図 6.6）。

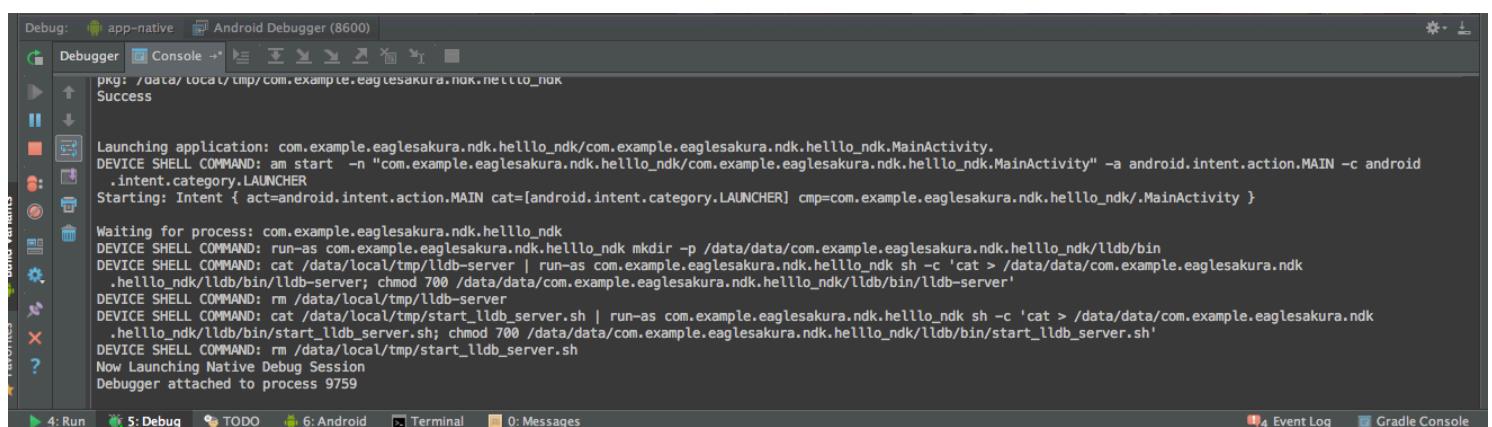


図 6.6 デバッガが利用可能な状態になるとコンソールにログが出力される

6.3.1 起動直後もデバッグしようじゃないか

起動直後の処理はデバッグできないのか？ と思うかもしれません、ちょっとした工夫でちゃんとデバッグできるようになります。Android にはデバッガが接続されるまで処理を強制的に停止するという機能があるので、それを使ってみましょう。

Android の開発者機能で"デバッグアプリを選択"をして、デバッグしたいアプリを選択し、

第6章 Android Studio で NDK ライフを送ろう

その後"デバッガを待機"を有効化します。これでデバッガが接続されるまで、アプリの起動を保留することができます（図 6.7）。



図 6.7 デバッガを待機 > デバッグアプリを選択

ただし NDK のデバッガが接続されても、Android は NDK 側のデバッガを「デバッガである」と認識しないため、いつまでたっても起動しません。そこで、Java 側のデバッガも追加でアタッチします（図 6.8）。

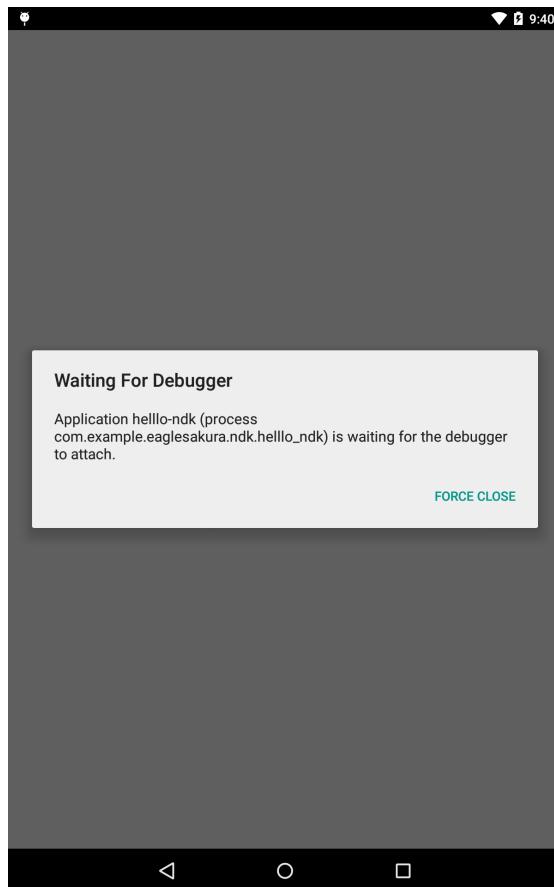


図 6.8 Java のデバッガを接続しないと、この画面から進まない

メニューの"Attach debugger to Android process"を選択して、接続したいアプリの package を選択して OK します（図 6.9）。

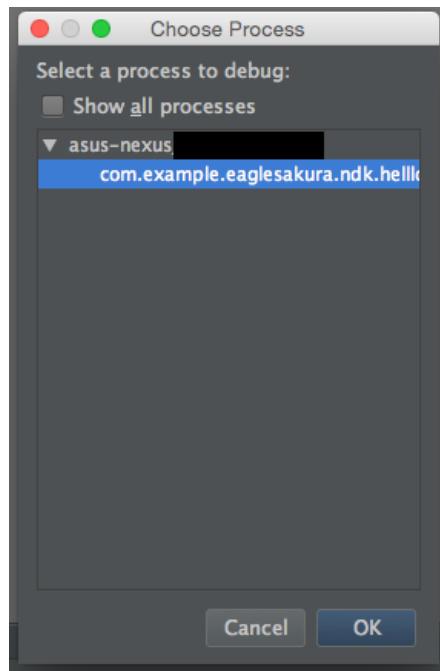


図 6.9 対象の Process を選択する

すると Android はデバッガ接続を認識してアプリ起動を続行してくれるので、あとはブレークポイントを待てば OK です。Java のデバッガも同時接続しているので動作は緩慢になりますが、この方法であれば NDK/SDK 両方のデバッグを行えます。便利ですね。

6.4 最後に

壮大に、そして壮絶にいろいろ頑張ってみましたが、これらはすべて開発中の機能です。特に build.gradle の Plugin 名等は変更される可能性が大いにあるといえます。

困ったときには、Google のサンプルやプラグインの解説が、とても参考になります。

- NDK サンプル - <https://github.com/googlesamples/android-ndk>
- 解説 - <http://tools.android.com/tech-docs/new-build-system/gradle-experimental>

Android SDK/NDK をひとつの IDE で扱えるというのは非常に便利で、たくさんの恩恵を受けることができます。もしも NDK を利用しているのであれば、リスクよりも開発効率・デバッグ効率の向上によるリターンのほうが数段上回っているといえるでしょう。

なお、残念なことに Android MNC では本記事執筆時点で NDK のデバッグ機能を利用できません (Lollipop を使ってください)。ひどいオチですね。

第7章

Unity 5.1 で入門する VR アプリ開発

7.1 はじめに

本章では、ゲームエンジンである Unity を用いて、Android スマートフォン向け VR アプリ開発に入門する方法について紹介します。

VR (Virtual Reality の略、ヴァーチャルリアリティ) という単語の知名度は高いでしょう。しかし、歴史上長らく、その VR なコンテンツを開発するには、専用の技術と環境が必要で、作るにも観るにもコストが高いものでした。

これは昨今の VR ブームを下支えする根拠でもありますが、実はカンタンに始めることができます。ぜひこの機会にチャレンジしてみるのはいかがでしょうか。

7.1.1 VR の R (Reality) と本章で扱う範囲について

Virtual Reality とは、直訳では「仮想の、事実上の現実感」です。実際に実物がそこにあるかのように人間に感じさせる技術を意味します。その技術体系は人間の五感に即して分類することができるものです。日常的に普及しているイヤホンやヘッドホンによる立体感のあるステレオ音声も、あたかも目の前で実物が音を出している Reality を実現した、聴覚に関する VR 技術の一種であるといえます。

本章では主に、高スペック化の進んだスマートフォンデバイスによって、視覚に関する Reality を Virtual に実現するアプリについて取り扱います。もう少し具体的にいうと、「首の向きに追従して、リアルタイムで絵が変わる」とこと、「左右の目に違う絵を表示することで立体的に見せる」ことです。

7.1.2 想定読者について

前半の読み物については特に想定読者を限定していません。後半の Unity によるアプリ開発の実践については次のような読者を想定しています。

1. Android アプリケーションの開発経験があり、Android SDK に関する基本的な知識を有する
2. Unity のいずれかのチュートリアルの経験があり、Unity に関する基本的な知識を有する

ただし、いずれも一般的なプログラミング経験、またはやる気があれば、インターネット上の情報を検索しながら半日～1日程度で必要な下知識を備えることができます。

7.2 VR とスマートフォンの関係

7.2.1 手持ちのスマートフォンで始めよう

スマートフォンの高スペック化によって、前述の「首の向きに追従して、リアルタイムで絵が変わる」とことと、それを「左右の目に別々に違う絵を表示する」ことが現実的になりました。

現実世界に比べて、スマートフォンのディスプレイサイズでは視野角が不足しています。その不足した視野角は、簡単な機構とレンズを備えた Google Cardboard などのハコメガネ型デバイス^{*1}で補助することで、驚くほどの現実感・没入感を得ています。

スマートフォンについては高スペックであればあるほどよいのですが、目安としては2年以内に発売されたものが望ましいでしょう。

ハコメガネ型デバイスは1,000～2,000円程度で購入できます。これらが揃っていれば、今すぐVRコンテンツの体験と開発を始めることができます。

7.2.2 具体的な機種とコンテンツの選び方について

スマートフォンの具体的な機種についてですが、Nexus 5(2013年11月発売、Snapdragon 800 MSM8974 搭載)と同等以上が望ましく、それよりスペックが劣るスマートフォンの際には、コンテンツを選ぶと考えください。

たとえば、360度全方向に敵が現れるシューティングゲームのような首振りの追従性が重要なゲームでは、高いFPS^{*2}を維持することが必須です。逆に、ほぼ首を振る必要がない、正面の狭い範囲でのみ世界を展開するコンテンツであれば低いFPSで問題ないケースも

^{*1} これらはVRゴーグルやVR HMD（ヘッドマウントディスプレイ）と呼ばれています

^{*2} Frame Per Second の略で、1秒間に何枚のフレームを描画しているかという指標。現行のスマートフォンのディスプレイ性能の最大値である60FPSを維持できることが望ましい

あります。

また、ハコメガネ型デバイスには左右の目で同一の絵を見る単眼式と、左右の目で違う絵を見る双眼式があります。立体感が重要でない、遠景の360度全天球パノラマ動画のようなコンテンツであれば、「左右の目に違う映像を表示することで立体的に見せる」ことを諦め、単眼式で十分といえます。双眼式であれば、1フレームで2回、左右別々の絵を描画しなければいけません。単眼式では、1回で描画を済ませることができます。より低い負荷で高いFPSを実現できるでしょう。

7.2.3 よりハイエンドなデバイスと、その価格について

少し話題が変わりますが、筆者がVRのデモを知人・友人に実施した際に、一番よく聞かれることは「(コンテンツを)自分でつくったの?」です。次いで聞かれることは「(手を出すには)いくらかかるの?」です。

スマートフォン自体、決して安価ともいえませんが、最近のVRブームの主役である、よりハイエンドなデバイスもあわせて、それらの価格を次に挙げます。

Oculus Rift DK2^{*3}、Oculus Rift CV1^{*4}、SONY Project Morpheus^{*5}については紹介のみとし、詳細な説明は省きます。スマートフォンを利用しないため、本章の対象外だからです。

- ハイエンドAndroidスマートフォン(6~8万円)/iPhone(5~10万円)+単眼ハコスコ/双眼Cardboard類(1,000~2,000円)
- GearVR(2.5万円)+Galaxy S6/S6 edge(8~10万円)
- Oculus Rift DK2(\$350+送料\$75、日本円換算で5~6万円)+PC(デスクトップであればビデオカードで3~5万円、ノートであれば18~25万)
- PS4(4~5万)+SONY Project Morpheus(2016年発売予定。価格未定)
- Oculus Rift CV1(2016年発売予定。価格未定)+デスクトップPC(おそらく5~10万円クラスのビデオカードが必要)

単体では10万円台から始められますが、気になったものを一通り揃えていくと、それなりのお金がかかります。今、手元に前述の条件に合うスマートフォンがあるのであれば、まずはそれで試してみることをおすすめします。

^{*3} <https://www.oculus.com/ja/dk2/>

^{*4} <https://www.oculus.com/ja/rift/>

^{*5} <https://www.playstation.com/en-gb/explore/ps4/features/project-morpheus/>

その後、Gear VR または Oculus Rift DK2 に手を伸ばすか、Oculus Rift の製品版である CV1 や PS4 に接続する SONY Project Morpheus の販売を待つか、あるいはもう一世代スマートフォンが高機能化するまで様子見をみる等の選択肢から選ぶことができます。

7.3 スマートフォン向け VR デバイスの紹介

本章のターゲットであるスマートフォン向け VR デバイスを紹介します。互換品が多数存在する Google Cardboard の存在など、入門者が混乱しやすい状況になっている点について、助け舟になれば幸いです（図 7.1、図 7.2）。



図 7.1 Google Cardboard（左）と Cardboard 互換 VR ゴーグル タオバイザー（右）



図 7.2 単眼（一眼）ハコスコ（左）と Oculus Gear VR（右）

7.3.1 Google Cardboard

Google Cardboard^{*6}は、ダンボールにアクリル製レンズをはめ込んだ Google 製の VR デバイスです。初出は Google I/O 2014 です。開発元の Google が設計図を無償で公開したため、クローンが多数存在します。

また、Google I/O 2014 発表時のものを Google Cardboard v1 と呼び、Google I/O 2015 発表のものを Google Cardboard v2 と呼びます。類似品と混同しないためのポイントを示します。

アプリの自動起動のために NFC タグが埋まっています。省かれているクローンもあります。v2 では廃止されています。

磁石によるスライドスイッチがあります。省略されているクローンもあります。磁力変化を bool 値として検出しています。スマートフォンの物理的なサイズが影響し、相性問題があります。v2 では磁石によるスライドスイッチは廃止されています。代わりに伝導素材を用いた画面をタップする機構が搭載されました。

Cardboard SDK では v1 のスライドスイッチと v2 の画面のタップは同等に扱われます。スライドスイッチが無いデバイスでも、画面を直接タップできる構造になっていれば、代用が可能です。

*6 <https://www.google.com/get/cardboard/>

公式に Google Cardboard 互換をうたっているデバイスでない場合の注意点として、表示サイズとレンズの位置が噛み合わない可能性があります。立体的に見えづらい等、不自然に見える環境での長時間の利用は避けたほうが良いでしょう。

Google は、Cardboard 互換デバイスを認証する取り組みとして View Profile Generator^{*7} を運用しています。デバイスの光学的な情報^{*8} を含めた QR コードを生成します。QR コードは、Cardboard 公式アプリ経由で読み取れます。Cardboard SDK を利用したアプリはスマートフォンのディスプレイサイズ・解像度にあわせた適切な描画を自動的に行います。Cardboard v1 と v2 の光学的性質の違いについては、公式アプリと QR コード経由での設定で調整が可能である点は覚えておいてください。

7.3.2 ハコスコ

ハコスコ社製のダンボールとフレネルレンズで構成された、単眼（一眼）の覗き込み箱^{*9} です。

単にハコスコと表記する際には、一眼レンズモデル^{*10} を差します。ハコスコ社は双眼の Google Cardboard クローンも販売しています。後述の Cardboard SDK は単眼式にも対応しているため、ハコスコ向けのコンテンツを作ることも利用できます。

7.3.3 Oculus Gear VR

Oculus 社と Samsung 社が協力して開発した VR ゴーグル^{*11} です。Galaxy スマートフォンを利用します。

日本では発売されなかった Galaxy Note 4 向けの初代と、日本でも発売されている Galaxy S6/S6 edge 向けの二代目があります。それぞれにデバイス接続互換性が無いことと、他のスマートフォンを Gear VR に装着して同様に利用することはできないことにご注意ください

Oculus Rift DK2 との違いは、スマートフォンが処理の主体であるためケーブルレスであること、DK2 の 1920x1080 より高解像度の 2560x1600 の液晶パネルをベースにしているた

*7 Viewer Profile Generator <http://www.google.com/get/cardboard/viewerprofilegenerator/>

*8 レンズとスクリーンの距離、レンズ間の距離、曲率が設定等がパラメータとして設定可能です。これらがまったく違うデバイスでも Cardboard 互換デバイスになり得えます

*9 <http://hacosco.com/>

*10 ハコスコ公式サイトでは、一眼レンズモデルをレギュラータイプと呼んでいます。<http://hacosco.com/product/>

*11 <http://www.samsung.com/jp/product/SM-R321/>

め網目感^{*12}が減少していること、ポジショントラッキングに対応していないことです。

Cardboardとの違いは、首振り予測やAsynchronous TimeWarp等、ハードとソフトの連携による高精度なヘッドトラッキングに対応していることです。Gear VR本体に埋め込まれた専用のモーションセンサを用いています。

また、ストア機能を含む専用ランチャーも大きな特徴のひとつです。Gear VRを装着したまま、専用アプリのラインナップの閲覧と購入とインストールを行うことができます。

7.4 VR と 3D とコンテンツ開発手法

VRデバイスに続いて、VRコンテンツの開発手法を解説します。

7.4.1 VR コンテンツは 3D で、3D なら Unity で

視覚的なVRを実現するには3Dグラフィックスを扱う必要がありますが、スマートフォンで3Dグラフィックスを扱うために、主に次のふたつの手段があります。

1. OpenGLを用いて直接的な3Dプログラミングを行う
2. UnityやUnreal Engine等のスマートフォンに対応したゲームエンジンを利用する

前者のOpenGLは抽象化されたグラフィックスハードウェアを扱う低次のAPI（とそのライブラリ）であるため扱うには、3Dに関する高度な知識が必要です^{*13}。

反面、後者はGUIを持ち、直感的にコンテンツ要素の作り込みが可能です。入門には、ゲームエンジンの利用を推奨します。現時点ではCardboardの際には、後述のCardboard SDK for Unityの存在により、Unityの利用をお勧めします。

7.4.2 Cardboard SDK for Unity のメリット

Cardboard SDK for Unity^{*14}の採用には、次の特徴、メリットがあります。

- Cardboardを使ってVR事業を推進しているGoogleが無料で公開

*12 液晶のドット格子模様がよく見えてしまうことをこう表現します

*13 VRコンテンツを開発しようとすると最終的には3Dの知識がひととおり必要になってしまいますが…

*14 <https://developers.google.com/cardboard/unity/>

- ・ライセンス料が無料^{*15}の Unity 5 Personal すべての機能を利用可能
- ・樽型歪み補正^{*16}に標準で対応
- ・双眼式と単眼式を動的に切り替えが可能
- ・クロスプラットフォーム。Android と iOS に対応

また、Cardboard SDK には Native Java 向けの Cardboard SDK for Android もあります。初心者には決して推奨できませんが、OpenGL アプリを書き慣れていたり、すでに手持ちの資産があったりする場合には採用を検討してください。

Cardboard SDK for Unity と同様にフリーの Unity 向けの VR アプリ開発ライブラリとして、Dive Unity Plugin^{*17}があります。

以前は有力な候補でしたが、今現在はおすすめしていません。ライセンス表示が必須であること、樽型歪み補正に対応していないこと、ドリフト補正^{*18}がないことなど、残念ながらいくつかの機能差があるため、Cardboard SDK for Unity をおすすめします。

Gear VR についても、Unity は 5.1.1^{*19}から標準でサポートしています。Cardboard SDK では、サポート外のため、メリットのひとつといえるでしょう。特別な SDK を追加することなく、Gear VR 向けアプリを開発できます。

■コラム: つくるてみるメリット

適切な閲覧が難しい出来の悪い VR コンテンツを、一般ユーザが体験すると（おいしくないウニを食べてしまったのと同様に）その不幸な体験により、「VR 嫌い」になってしまう恐れがあります。これを避けて、効率よくおいしく VR コンテンツを調理するには適切な知識、ノウハウが必要です。これを手っ取り早く身につける方法が「つくるてみる」というわけです。

また、VR 業界の現状として、キャンペーン・広告目的で、魅力的なコンテンツも期間限定であるケースが多いです。つくるための情報を収集していると、コンテンツの情報も効率よくキャッチアップすることができ、素敵な VR コンテンツを逃すことなく楽しむことがで

*15 売上が年間で US\$100,000 を超える際には、Unity 5 Professional ライセンスが必要です

*16 Barrel Distortion Correction。レンズを通したように歪んで見える絵を補正して表示できる

*17 <https://www.durovis.com/sdk.html>

*18 ジャイロセンサのみで姿勢（頭の向き）をとっていると一定方向にまわり続けるドリフト現象が起きてしまうので、他のセンサを用いてこれを補正すること。この補正がないと徐々に回転してしまうため、コンテンツに影響が大きい

*19 Unity 5.1.1 リリースノート <http://unity3d.com/jp/unity/whats-new/unity-5.1.1>

きるようになります。

「進撃の巨人展」360° 体感シアター"哮"や、ニコニコ超会議 2015 のロートデジアイ初音ミク VR スペシャルライブ「ALIVE」のように、運用までしっかりとサポートされたものは事前知識のない人でもきちんと楽しめるようになっていました。マシンスペック、適切な運用、適切なユーザの動作を揃えるには、それなりのコストがかかるものです。

7.5 VR アプリ開発環境の構築

さて、ここからが本題です。実際に Unity と Cardboard SDK を用いた開発環境を構築し、Android デバイス向けの VR アプリを開発してみましょう。

読者として Android アプリの開発経験者を想定しているため、JDK や Android SDK の詳細なインストール手順は省略し、それぞれの環境構築においての注意事項のみ記載しています。また、本節以降は主に OS X (Mac) を対象としています。OS 毎の差分が大きいところでは両方について記載しましたが、Windows 環境の方は適時読み替えください。

7.5.1 手順のおおまかな流れ

Getting Started with Unity for Android^{*20}をベースとし、環境構築の手順は次のとおりです。

1. JDK 7 (の最新) をインストールします
2. Unity から Android アプリをビルドするために、Android SDK を未インストールであればダウンロードして展開します。IDE (Android Studio) は必須ではないため、"SDK Tools Only" option is sufficient (SDK Tools のみインストールすれば十分である) という表記にしたがって、SDK のみをインストールします
3. Unity 5.x (の最新) をインストールします
4. Cardboard SDK for Unity をダウンロードします

^{*20} Getting Started with Unity for Android <https://developers.google.com/cardboard/unity/get-started>

7.5.2 Android SDK とインストール

Android SDK のインストールにあたっては、その時点での最新の正式リリース（preview 等がついていない一番新しい）の Packages が SDK Manager 起動時にデフォルトでチェックされていますので、それらをダウンロードし、インストールしましょう。

以降は執筆時（2015年8月）の情報に基づいていますが、Unity を用いた開発では、最新の Android SDK でなくとも問題のないケースが多いため、そのままでも参考になるでしょう。

Android Developers 公式 SDK ダウンロードサイト^{*21}より、android-sdk_r24.3.3-macosx.zip (Windows であれば android-sdk_r24.3.3-windows.zip) をダウンロードして展開します。

zip を展開して、android-sdk-macosx/tools/android を開くと、SDK Manager が起動します（Windows であれば android-sdk-windows/SDK Manager.exe です）。「Android M (API 22, MNC Preview)」のチェックは外しても構いません。

また、Windows であれば「Google USB Driver」のチェックを有効にして、Install xx packages のボタンを選択します。あとはインストールした Android SDK のパスを、Unity からビルド時に指定するだけです。

7.5.3 Unity のインストールとバージョン

Unity は正式リリース^{*22}と Patch リリース^{*23}があります。Patch リリースには「パッチリリースの利用は、修正されたバグに影響されているときのみ利用されることをお勧めしています。」と記載されています。原則は両方のうち、最新版を利用するとよいでしょう。

しかし、執筆時（2015年8月）、筆者の Mac では幾つかの古いバージョンで Gear VR 向けの設定で Unity のクラッシュが発生しました^{*24}。本章では最新の 5.1.2p2 確認をしていますが、クラッシュの発生時には古いバージョンを試してください。

^{*21} Android Developers - Other Download Options - SDK Tools Only <http://developer.android.com/sdk/index.html#Other>

^{*22} Unity Release <http://unity3d.com/jp/get-unity/update>

^{*23} Unity Patch Release <https://unity3d.com/jp/unity/qa/patch-releases>

^{*24} 他の人は無事という話も聞いたので、適時バージョンを変更してください

7.5.4 Cardboard SDK for Unity のダウンロード

Cardboard SDK for Unity 公式サイト^{*25}を参考に、「CardboardDemoForUnity.unitypackage」と「CardboardSDKForUnity.unitypackage」の2つをダウンロードしてください。執筆時（2015年8月）最新のv0.5.1で動作確認しています。

7.6 Cardboard 向け、VR アプリの Hello, World

では、環境構築に引き続いだり、Cardboard 向けアプリをビルドし、Android 実機で実行するまでのHello, Worldを試してみましょう。

7.6.1 新規 Unity プロジェクトの作成

Unity を起動し、NEW PROJECT を選択します（図7.3）。「Sign into your Unity Account」と Unityへのサインインを求められますが、Work Offlineを選択してもユーザ登録（無料）をしてもどちらでも構いません。

^{*25} Download and Samples - Cardboard SDK for Unity and Demo <https://developers.google.com/cardboard/unity/download>

第7章 Unity 5.1 で入門する VR アプリ開発

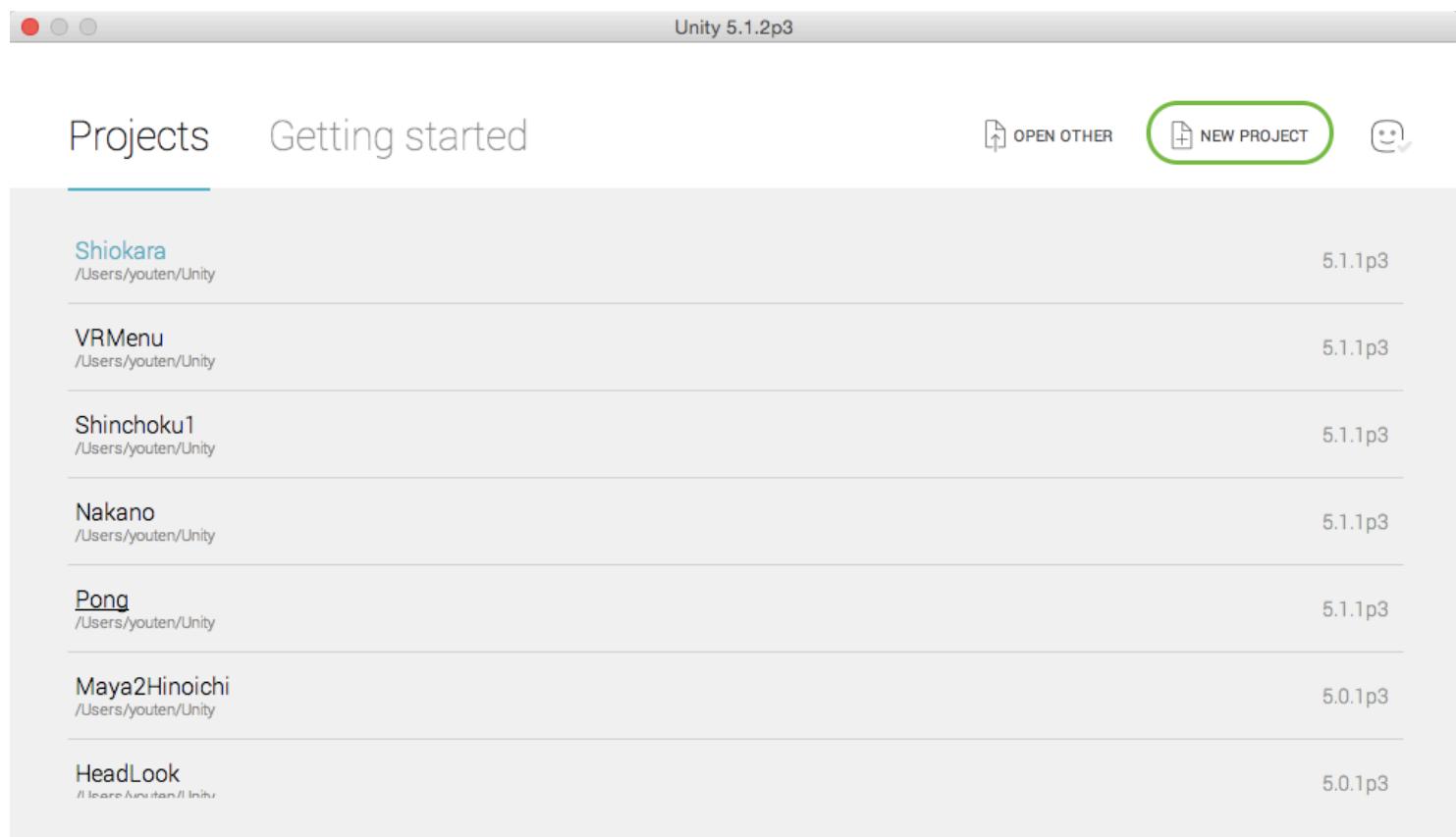


図 7.3 Unity 起動画面（最近開いたプロジェクト一覧）

Project name に名前と、Location にプロジェクトのパスを指定し、Create project を選択します（図 7.4）。3D/2D 選択はデフォルトの 3D のままでかまいません。

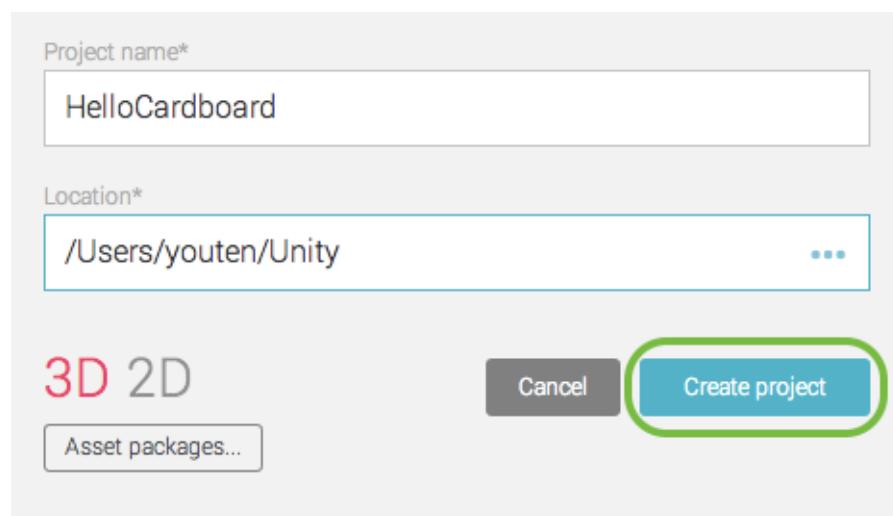


図 7.4 プロジェクト名とパスの指定

新規プロジェクト作成直後は Unity のメイン画面が表示されます。各ビューの配置は好き

に重ねたり、移動したり並べたり変更できます。いくつかのテンプレートパターンが用意されていますので、「Window」>「Layout」から変更してみてください。筆者は"「Tall」Layout から Game ビューを下段に配置した図 7.5 のとおりの配置を好んで使っています。

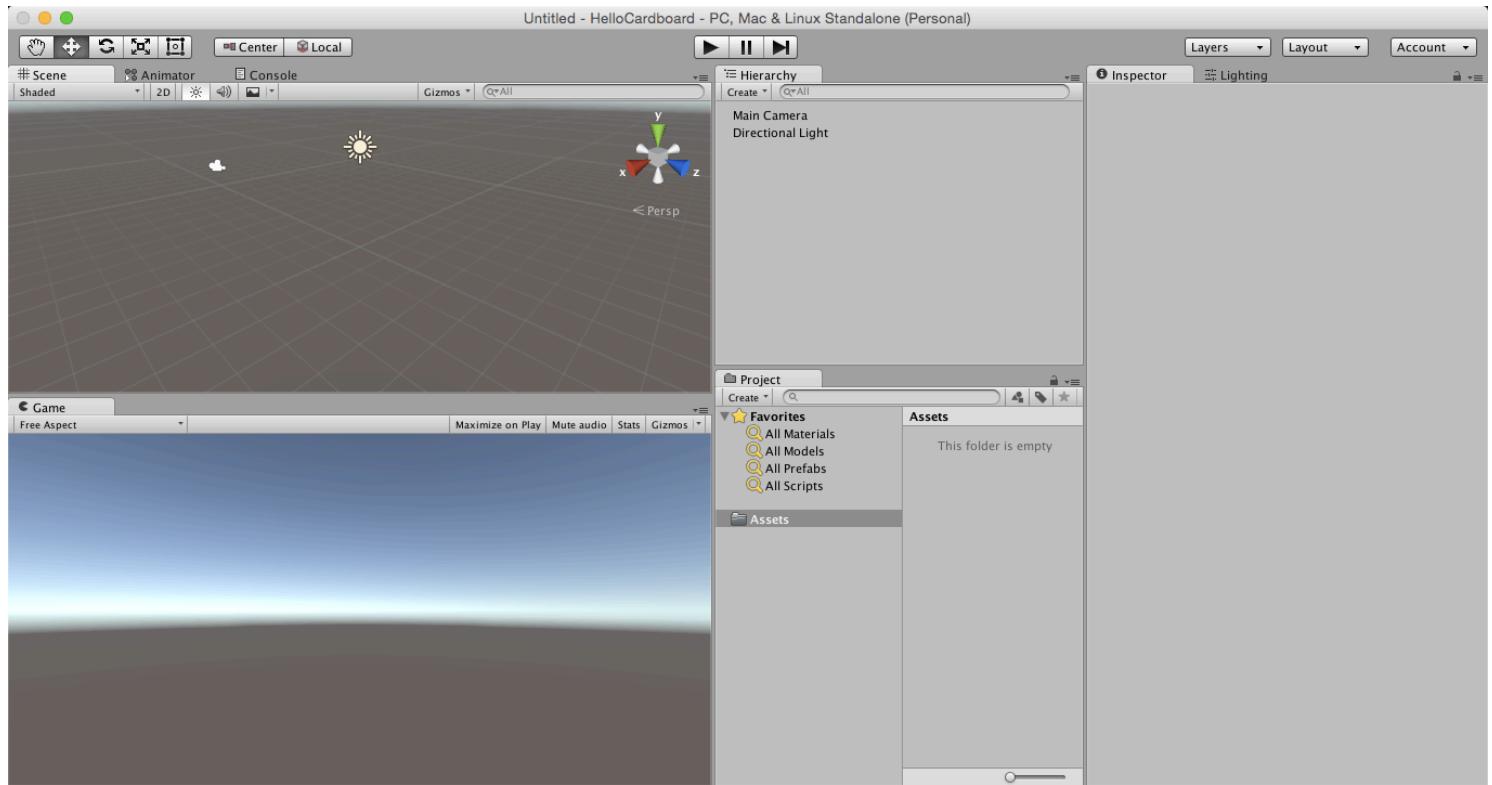


図 7.5 プロジェクト新規作成直後のメイン画面

ディスプレイ解像度にもよりますが、作業毎に適切な配置は違うと思いますので、面倒くさがらずにこまめに変えてみることをお勧めします。

7.6.2 Cardboard SDK for Unity のインポート

次に、Cardboard SDK for Unity をインポートします。「CardboardSDK-ForUnity.unitypackage」をダブルクリックで開いてください。Unity の起動中であれば自動的にインポート確認画面が表示されますので、Import ボタンを選択します（図 7.6）。

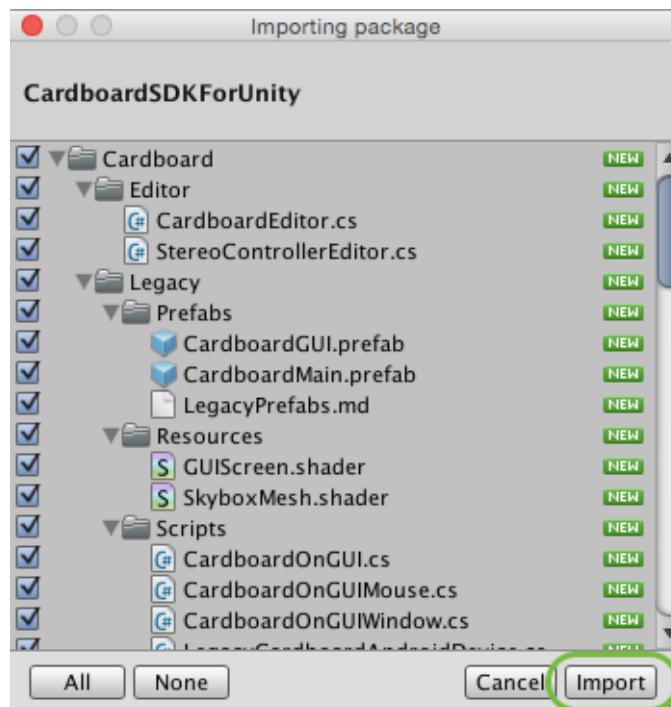


図 7.6 Cardboard SDK for Unity unitypackage のインポート

続けて、Demo シーンもインポートします。「CardboardDemoForUnity.unitypackage」を同様にダブルクリックで開き、Import ボタンを選択します。(図 7.7)



図 7.7 Cardboard SDK Demo for Unity unitypackage のインポート

インポートが成功したら、Project ビューの Assets/Cardboard/DemoScene/DemoScene シーンアイコンをダブルクリックし、DemoScene を開きましょう（図 7.8）。

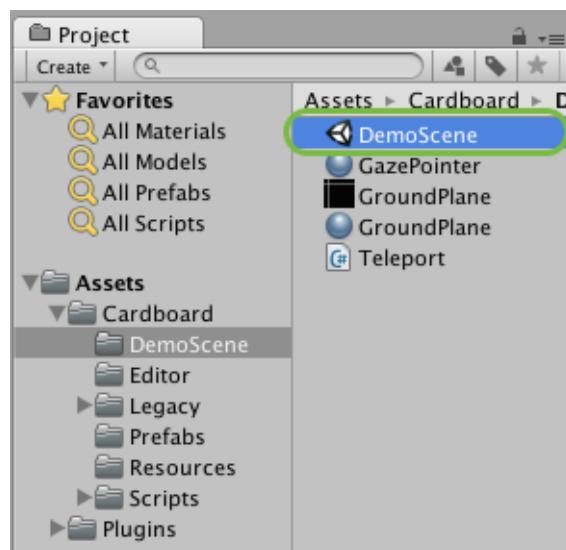


図 7.8 DemoScene

格子状の白い線が走った黒い床の上に、Cube（立方体）が浮かんでいる Scene が読み込まれたら成功です。

7.6.3 プレビューの再生

続けて再生ボタンをクリックし（Command + p でも実行可能です）、プレビューを再生させてみましょう（図 7.9）。



図 7.9 プレビューの再生

空中に浮かぶ Cube から小さな光球が湧き出るエフェクトが出ていれば正常にプレビュー再生できています。

option を押しながらドラッグ、control を押しながらドラッグで回転操作をることができます。

これは、Cardboard にスマートフォンをセットした状態での首振りをエミュレートしてい

ます。

ちなみに、このデモゲームはエフェクトを出すCubeがテレポートして飛び回るので、首をぐるぐるまわして探し回ろう、というものです（図7.10）。

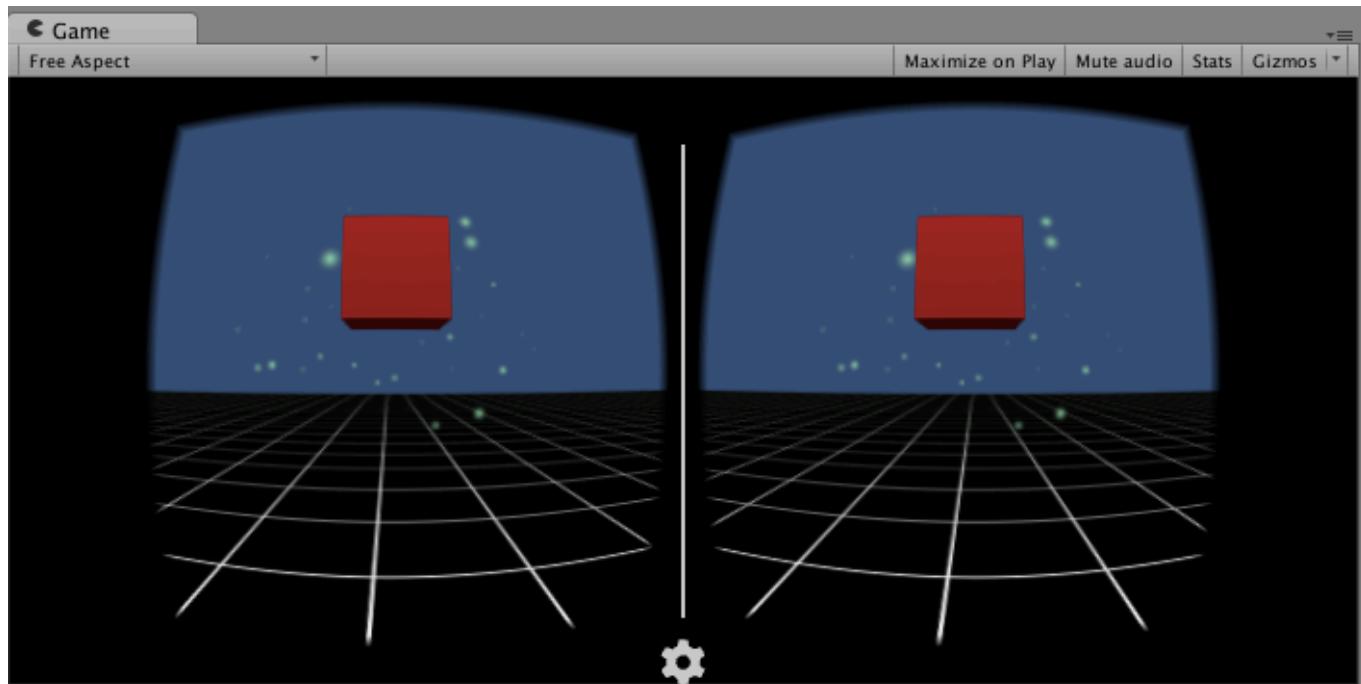


図7.10 DemoScene プレビュー再生中

Cubeに視点をあわせてみましょう^{*26}。視線があれば、Cubeの色が赤から緑に変わります。この状態でクリック（Cardboardではスライド磁石スイッチまたは押し込みスイッチに該当）をすると、他の場所にCubeがテレポートします。

このプレビューの動作までがHello, Worldとしては前半戦です。次項からは、設定とビルトを実施し、Android実機にインストールして動作確認をするまでの後半戦に入ります。

7.6.4 Android プラットフォーム向けの設定

「File」>「Build Settings...」を開きます（図7.11）。

^{*26} Unityでは仮想的な光線で物体への衝突判定を行うRaycastという機能（API）があり、照準や視線をあわせることをRaycastと表現することがあります



図 7.11 Build Settings

現在開いている Demo Scene をビルド対象に追加、Android を選択して Switch Platform を実施後、Development Build のチェックを ON にして「Player Settings...」を開きます。

Inspector ビューの PlayerSettings の中、まずは Company Name を適当な名前に変更します（図 7.12）。

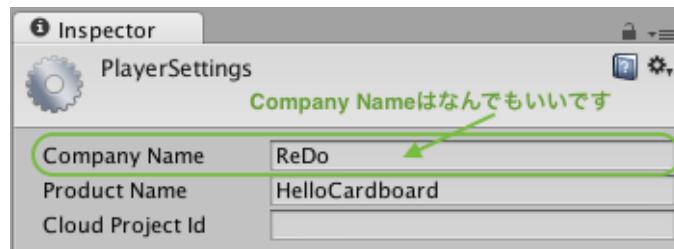


図 7.12 Company Name を変更

続けて Resolution and Presentation から Default Orientation は Landscape Left を選びます（図 7.13）。

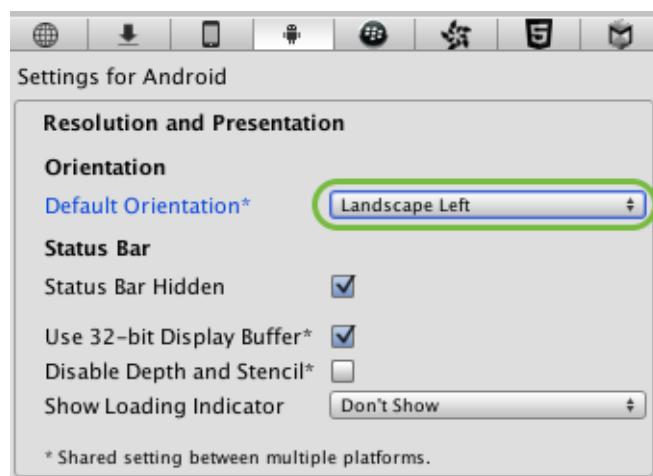


図 7.13 Default Orientation は Landscape Left に指定

Other Settings の Bundle Identifier は適当なパッケージ名+プロダクト名を入力してください（図 7.14）。

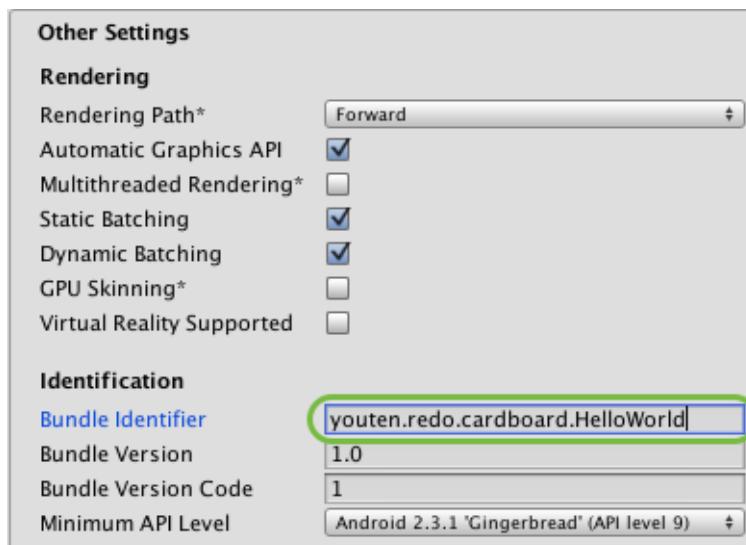


図 7.14 Bundle Identifier は適当なパッケージ＋プロダクト名を入力

Rendering 周辺の設定を誤って変更しないように注意してください。また、ARM アーキテクチャの Android でしか動かすつもりが無いのであれば、Device Filter を FAT (ARMv7 + x86) から ARMv7 に変更しておくと、apk のサイズが少し縮小されます。

7.6.5 ビルドとインストールと実機での実行

ここまで設定が終わったら、再度「Build Settings」を開きます。今度は「Build And Run」を選択すると、apk のファイル名を聞かれますので、「HelloCardboard」と入力してください。

初回のビルド時には Android SDK のパスを聞かれるので、前述の Android SDK を展開したディレクトリを指定します。Debug モード (adb - Android Debug Bridge) が有効になった Android が接続されていれば、自動的にインストール (adb install) され、アプリが起動します。

Android デバイスが接続されていない際にはインストールに失敗しますので、「Build And Run」ではなく、「Build」を選択してください（図 7.15）。

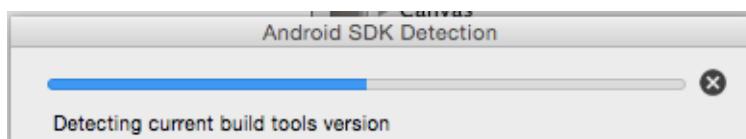


図 7.15 ビルド中

ビルドが成功すると、HelloCardboard.apk がプロジェクトファイル直下に出力されるので、Android 端末にインストールし、起動してみてください。Unity ロゴのスプラッシュスクリーンに続いて、プレビューと同様の画面が出れば成功です（図 7.16）。

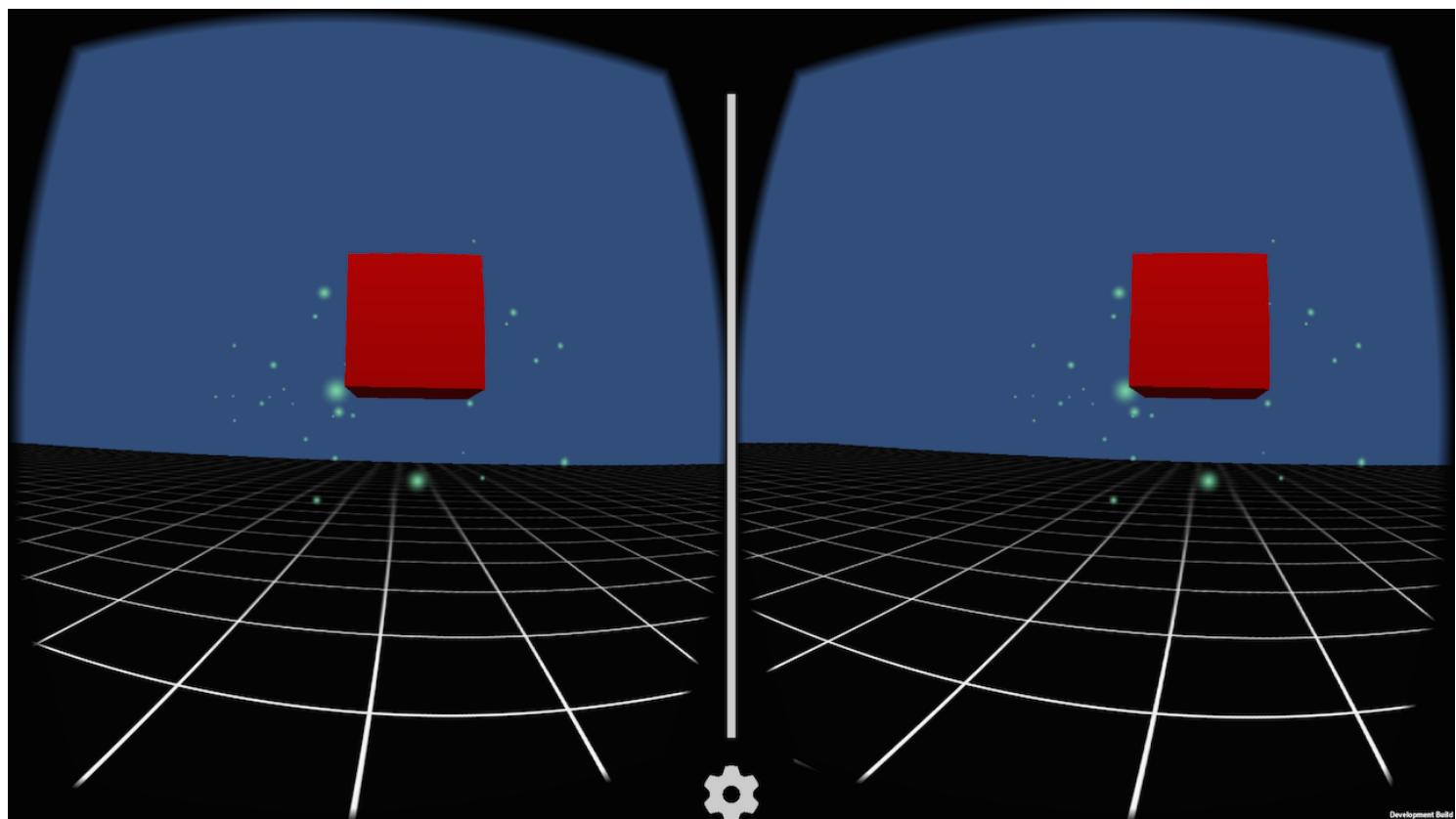


図 7.16 Demo アプリ起動画面

7.7 PONG アプリを Cardboard 向けに対応させる

さて、Hello, World まで終わったあとは、実践編に移りましょう。もう少し Unity らしい、物理演算を利用したアプリを、複数の VR デバイス向けに VR 対応していく事例を紹介していきます。

ここでは、物理演算を利用した Unity アプリのサンプルとして、PONG^{*27}を参考に、永遠に一人で壁打ちする「PONG のようなもの」を用意しました。

^{*27} Google I/O 2015 の keynote でも登場した、歴史的なアタリ社の対戦型卓球ビデオゲームです

7.7.1 新規プロジェクトと PONG unitypackage のインポート

CBPong という名前で新規プロジェクトを作成し、次の GitHub ページより、「PongStage.unitypackage」をダウンロードして、インポートしてください（図 7.17）。



図 7.17 GitHub から PONG の unitypackage のダウンロード

- Pong - Pong-like Unity app using Google Cardboard SDK Sample.
 - <https://github.com/youten/Pong>

この unitypackage には PONG のようなもののゲーム要素がひとまとめにして入っています。また、GitHub リポジトリは全体で完成品の Unity プロジェクトにもなっているため、clone してビルドすることもできます。

インポートしたあと、Project ビューの Assets/Pong/PongScene をダブルクリックで開いてください。

プレビュー再生を実行すると、PONG アプリ^{*28}が動作し、左右キーで自機のバーを移動できることを確認しましょう（図 7.18）。

^{*28} PONG というには対戦型でもないですし、ピンポンではなくエアホッケーぽいですし、むしろブロックなしのブロック崩しという感じですが、細かいことには目をつむってください

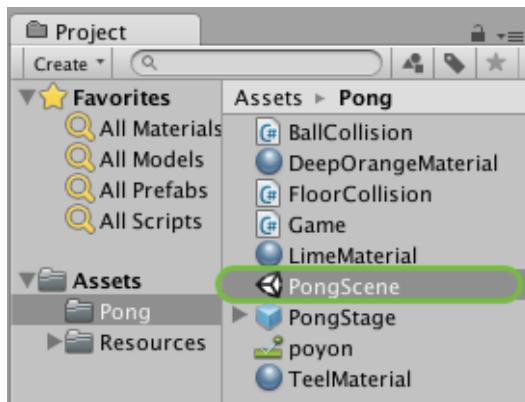


図 7.18 PongScene

7.7.2 Android プラットフォーム向けの設定

まずは前節の手順と同様に、Android プラットフォーム向けに各種設定を変更しましょう。手順を示します。

1. 「File」>「Build Settings...」を開いて、Platform で Android を選択、Switch Platform を実行する
2. 「Add Current」を選択し、Pong/PongScene.unity をビルド対象のシーン一覧に追加する
3. 「PlayerSettings...」ボタンから Inspector ビューに「PlayerSettings」を開く
4. Company Name を変更し、Resolution and Presentation の Default Orientation を Landscape Left に変更する
5. 「Other Settings」より Identification の Bundle Identifier を適当なパッケージ名に変更する
6. 「Build Settings」を再度開き、Development Build のチェックを入れて、「Build And Run」を選択、ビルドする。apk 名称は CBPong.apk を指定する

Android 実機上で PONG のようなアプリが起動したら成功です。この時点では自機のバーを一切操作することができません。ここから、Cardboard SDK を適用し、VR アプリとして仕上げていきましょう。

7.7.3 Cardboard SDK の適用

Cardboard 対応は基本的には、通常の Main Camera を、Cardboard 向けのカメラである CardboardMain に入れ替えることで完了します。カンタンですね。

前節と同様に「CardboardSDKForUnity.unitypackage」をインポートします。次に、Project ビューの Assets/Cardboard/Prefabs/CardboardMain を Hierarchy ビューにドラッグ & ドロップし、追加します。元からあった Main Camera は選択して右クリックメニューから「Delete」または Command + delete にて削除しましょう（図 7.19）。

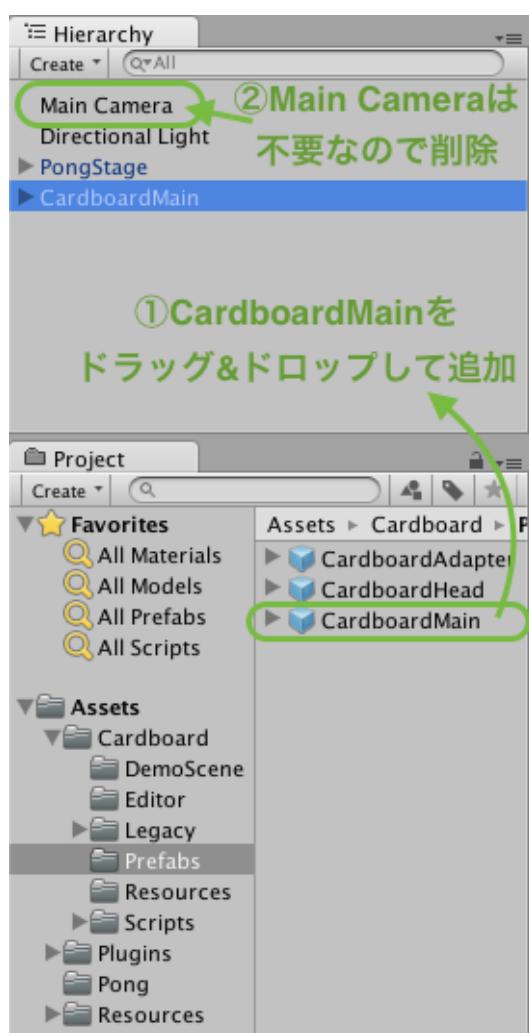


図 7.19 Camera を CardboardMain に入れ替え

CardboardMain の Transform を Position を X=0、Y=3、Z=-13 に変更します。Rotation は最終的には X=0、Y=0、Z=0 がよいですが、開発途中は X=20 ぐらいにしておくと少し見下ろした形での Game ビューとなり、確認がしやすくなります（図 7.20）。

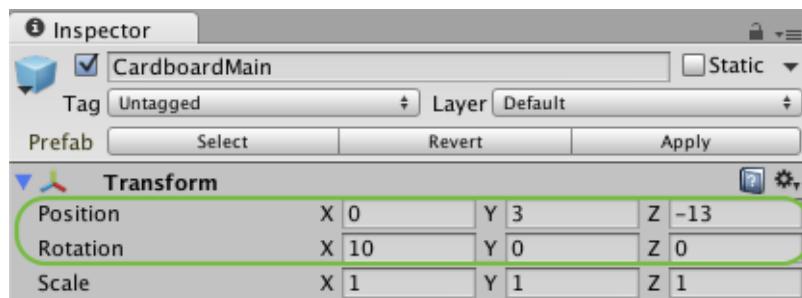


図 7.20 CardboardMain の Transform を調整

7.7.4 VR らしい自機制御スクリプト

続いて、VR らしい自機制御スクリプトということで、Cardboard を覗き込んだ状態で左右に首を振ることで、自機のバーが左右に移動するようにスクリプトを書きましょう。

Project ビューの Assets を選択した状態で、「Create」>「C# Script」を選択、MeHead と名前を入力します。MeHead スクリプトをダブルクリックすると（Unity インストール時のデフォルトでは）MonoDevelop というエディタが起動します（図 7.21、図 7.22）。

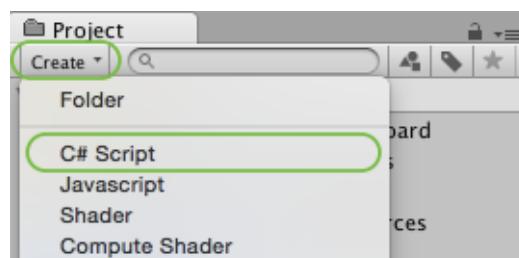


図 7.21 MeHead.cs C# スクリプトの追加

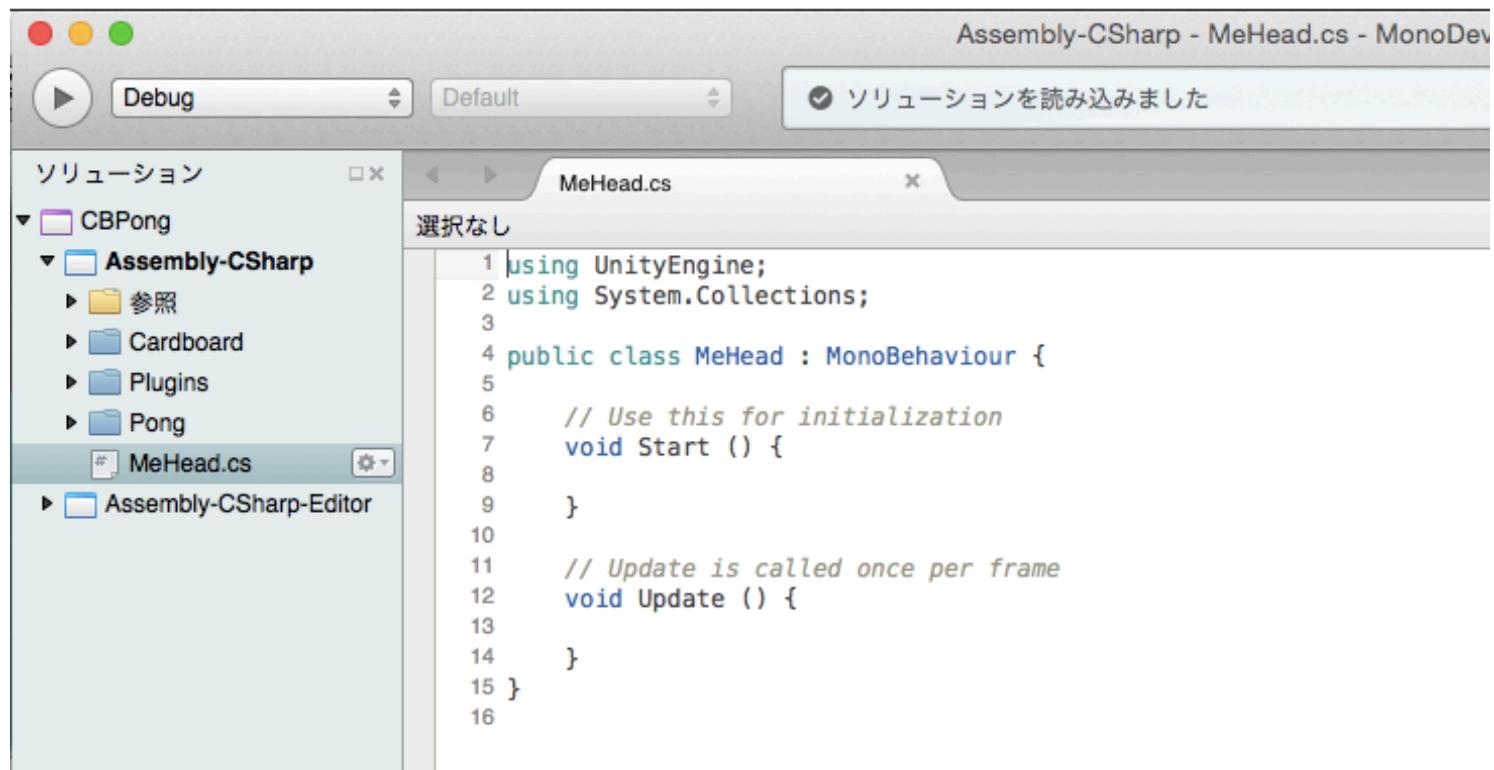


図 7.22 MonoDevelopment 起動時

MeHead.cs をリスト 7.1 のとおりに修正してください。

リスト 7.1: MeHead.cs - 首振りに合わせて自機のバーを移動するスクリプト for Cardboard

```

using UnityEngine;
using System.Collections;

public class MeHead : MonoBehaviour {
    public GameObject MeObject;

    // Update is called once per frame
    void Update () {
        Quaternion head = Cardboard.SDK.HeadRotation;
        float posX = head.y * 20.0f;
        if (posX < -Game.MAX_ME_X) {
            posX = -Game.MAX_ME_X;
        } else if (Game.MAX_ME_X < posX) {
            posX = Game.MAX_ME_X;
        }
        // move bar
        MeObject.transform.position =
            new Vector3 (posX,

```

```

        MeObject.transform.position.y,
        MeObject.transform.position.z);
}
}

```

まず更新したMeHeadスクリプトをCardboardMainにドラッグ&ドロップして追加し、続けて、追加したスクリプトのMe Objectプロパティに、HierarchyビューのPongStage/Meを指定します(図7.23)。

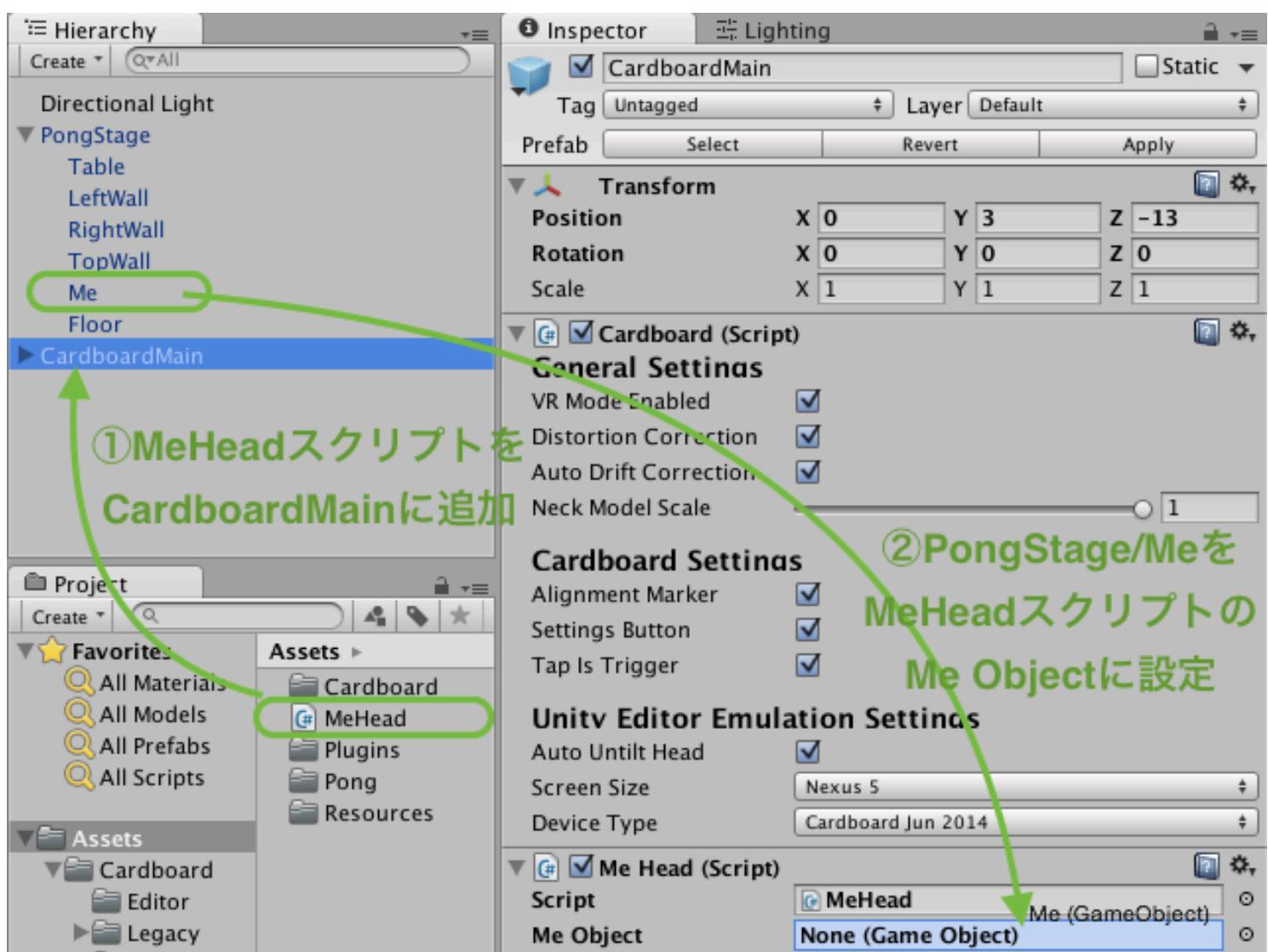


図7.23 MeHeadスクリプトの追加と設定

プレビュー再生中であればoptionを押しながらドラッグ、Android実デバイス上で起動している場合には首振りによって自機のバーが移動し、ボールを打ち返すことができれば成功で

す(図7.24)。

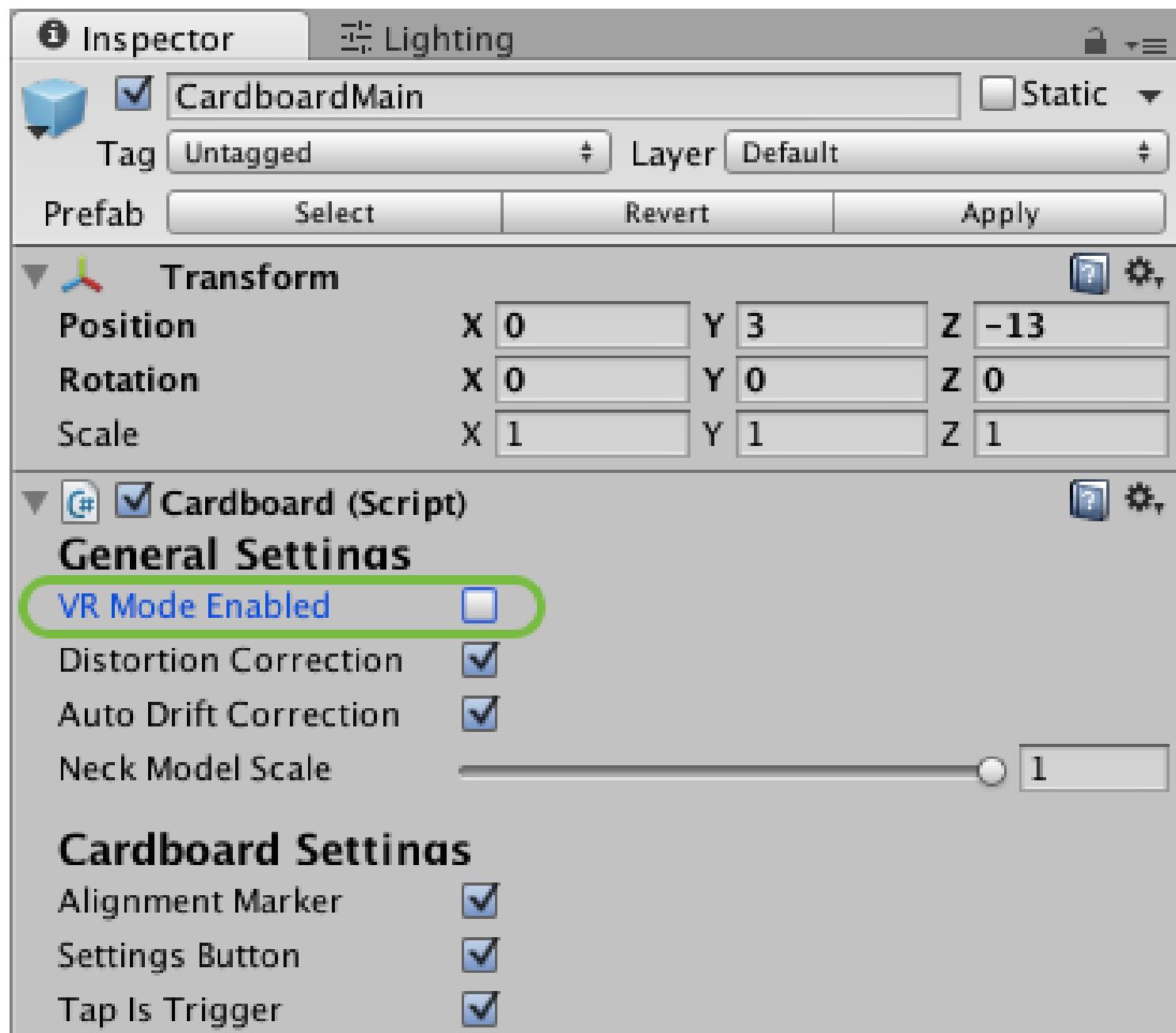


図7.24 PONG のようなもの for Cardboard

7.8 PONG アプリをハコスコ向けに対応させる

Cardboard SDK はデフォルトでは双眼式のステレオ表示ですが、ハコスコの様な単眼デバイス向け表示にも対応しています。

Inspector ビューから、CardboardMain にアタッチされている Cardboard スクリプトの

VR Mode Enabled というチェックボックス^{*29}を OFF にすると、シングル単眼向け表示になります（図 7.25）。このプロパティは Boolean 値になっており、スクリプトから動的に切り替えることもできます。

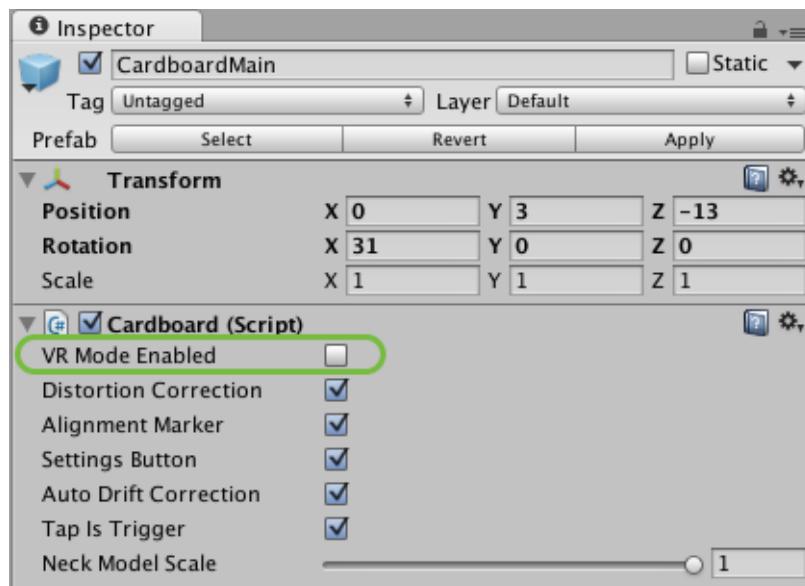


図 7.25 VR Mode Enabled で単眼向けシングル/双眼向けステレオ表示を切り替え

7.9 PONG アプリを Gear VR 向けに対応させる

最後に PONG のようなものを Gear VR に対応させましょう。本節は、Gear VR を持っている開発者向けです。前述のとおり、Unity 5.1.1 から Gear VR に対応したため、追加 SDK は不要です^{*30}。

7.9.1 新規プロジェクトと Virtual Reality Supported チェックボックス

「File」>「New Project...」で新しいプロジェクトを作成、名前は「GearPong」とします。「PongStage.unitypackage」をダブルクリックして、インポートし、Pong/PongSceneを開くところまではこれまでと同じ手順です。続けて、Gear VR 向けの設定を適用していきましょう。

^{*29} 双眼モード=VRモードというのは少し不自然に思えますが、Cardboard SDK ではこのような表現になっています

^{*30} それ以前は、Oculus Mobile SDK が必要でした

1. 「File」 > 「Build Settings...」 から Build Settings ウィンドウを開きます。Android を選択、Switch Platform を実行する
2. 「Add Current」 ボタンを選択し、Pong/PongScene.unity をビルド対象のシーンに追加する
3. Development Build のチェックは不要です。「Player Setings...」 ボタンを選択し、Inspector ビューに変更する。
4. 任意の Company Name を入力後、Other Setings の Virtual Reality Supported を ON にする
5. Bundle Identifier に適切なパッケージ名を入力する
6. Galaxy のみがターゲットのため「Device Filter」で ARMv7 限定に設定を変更する。オプションですが Multithreaded Rendering を有効にしてもよいでしょう。

プレビュー再生時に音が割れたりした際には、プロジェクトを保存後、Unity を再起動してプロジェクトを読み直してみてください。Virtual Reality Supported チェックボックスの ON/OFF を切り替えた際の再読み込みが何か影響して不具合となっているようです（図 7.26）。

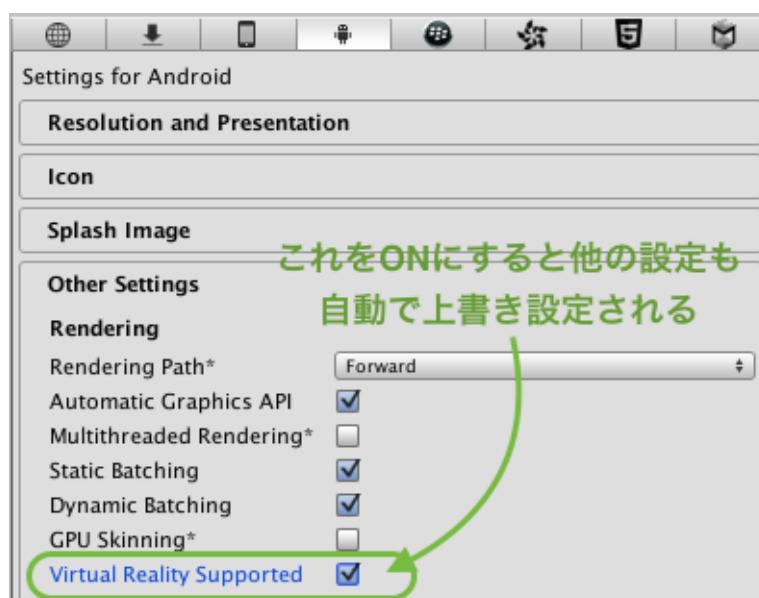


図 7.26 Virtual Reality Supported チェックボックスを ON に

7.9.2 Osig ファイル (Oculus Signature File) の準備

さて、設定が終わって Galaxy 実機での動作確認を…という前に、通常の Android デバイス開発向け準備（開発者モードを有効にしたり、Windows であれば adb 接続のための USB ドライバまわりの設定を変更したり）とは別に、Gear VR 向けにふたつほどポイントがあります。本項と次項にわたって説明します。

ひとつめのポイントは、Gear VR 向け apk には、特定の Galaxy デバイスでしか動作しないように、Device Id を元にした、osig ファイル (Oculus Signature File) を含める必要があります。

Oculus 開発者向け公式サイトの Oculus Signature File (osig) Generator^{*31}で、Device Id を入力して osig ファイルを生成します。Device Id は adb devices コマンドで確認できます。

ダウンロードした osig ファイルは、Assets 配下、Plugins/Android/assets/oculussig_xxxx というパスに配置します。Finder やエクスプローラから直接操作してもいいですし、Unity の Project ビューにファイルをドラッグ＆ドロップすることもできます（図 7.27）。

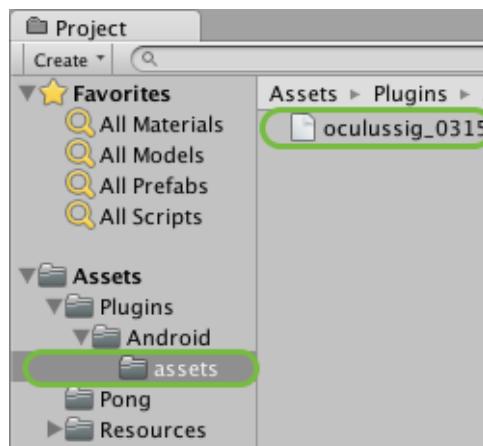


図 7.27 osig ファイルをプロジェクトに配置

osig ファイルの配備が完了したら、「File」>「Build Settings...」から Build And Run を選択します。apk 名を入力すると、Cardboard の際と同様に、ビルドが始まって、インストール、起動と進みます。アプリの起動時に、Galaxy を Gear VR に接続していなければ図 7.28

*31 <https://developer.oculus.com/osig/>

のように、Gear VR への挿入を促すダイアログが表示されます。ダイアログを表示させたまま Gear VR に Galaxy をセットすると、継続してアプリが起動します（図 7.29）。



図 7.28 未挿入状態で Gear VR 向けアプリを起動した場合

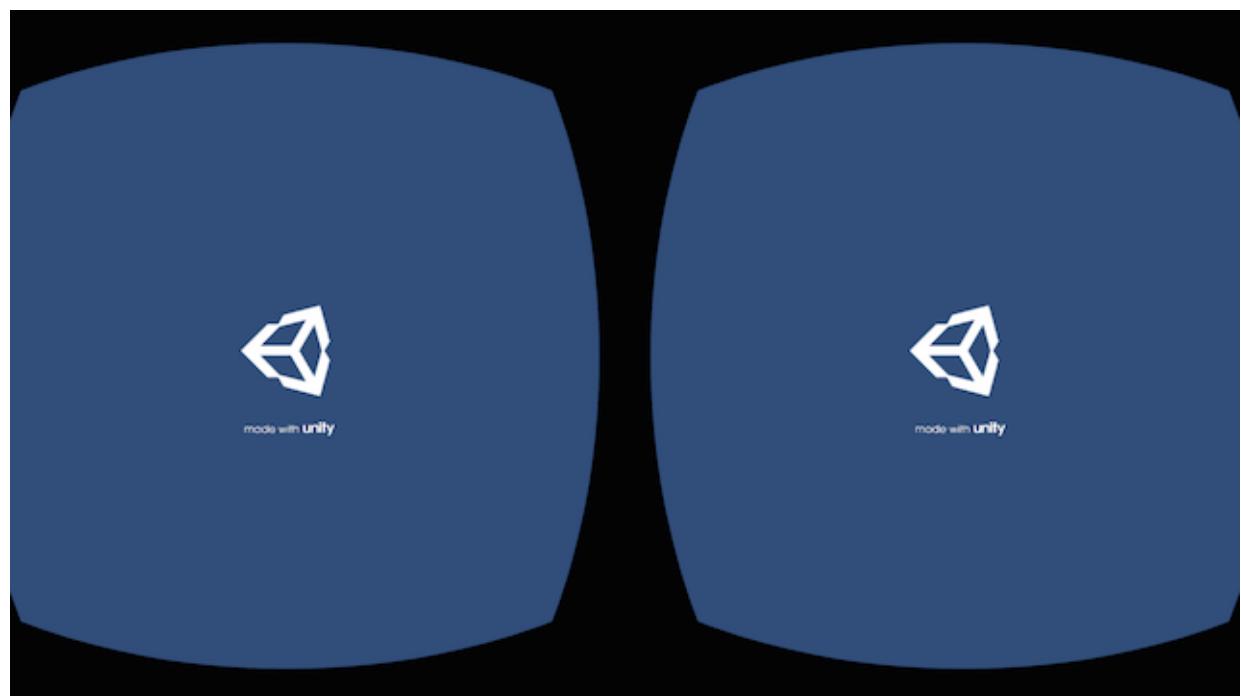


図 7.29 VR 向け Unity スプラッシュスクリーン

7.9.3 Gear VR Service 開発者モードの有効化

ふたつめのポイントは、Gear VR Service の開発者モードの存在です。Gear VR Service は Gear VR に対応した Galaxy にプリインストールされている、Gear VR 向けアプリの管理アプリのようなものです。この Gear VR Service の開発者モードが有効になつていれば、Gear VR に挿入していない状態でも、Gear VR 向けアプリを起動させることができます（図 7.30）。



図 7.30 Gear VR Service の開発者モード

Galaxy の「設定」>「アプリケーション」>「アプリケーション管理」>「Gear VR Service」

> 「ストレージを管理」> 「バージョン」項目を連打^{*32}して有効します。ただし、スマートフォン内のアプリをスキャンして、1つ以上のGear VR向けアプリ^{*33}が入っていないと有効にできません。

また、Gear VR Serviceの開発者モードを有効にする以外に、開発時にGear VRとGalaxyの付け外し回数を削減したい際には、Wi-Fi経由のadbを利用するという方法もあります。ただし、Unityアプリは、Unityゲームエンジンとしてのコア部分が決して小さくないため、apkサイズが肥大化しやすく、Wi-Fi経由のadb installにはとても時間がかかります。

開発の際には、adb経由でのlogcatデバッグやスクリプトにブレイクポイントを貼るデバッグも利用できますが、コンシューマゲームでもよくある「デバッグモード」「開発者モード」のように、アプリ、ゲーム内機能としてデバッグ機能を作り込んでしまうのも有効だとうのを頭の片隅に置いておいてください。

続けて、Cardboardの際と同様に、首のフリに応じて自機のバーが移動するスクリプトを書きましょう。

ProjectビューのAssetsを選択した状態で、「Create」>「C# Script」を選択、MeHeadVRという名前を入力し、MeHeadVR.csをリスト7.2のとおりに修正します。

リスト7.2: MeHeadVR.cs - 首振りに合わせて自機のバーを移動するスクリプト for Gear VR

```
using UnityEngine;
using UnityEngine.VR; // added
using System.Collections;

public class MeHeadVR : MonoBehaviour {
    public GameObject MeObject;

    // Update is called once per frame
    void Update () {
        // changed
        Quaternion head = InputTracking.GetLocalRotation (VRNode.Head);
        float posX = head.y * 20.0f;
        if (posX < -Game.MAX_ME_X) {
            posX = -Game.MAX_ME_X;
        } else if (Game.MAX_ME_X < posX) {
            posX = Game.MAX_ME_X;
        }
    }
}
```

*32 6回らしいです。<https://docs.unrealengine.com/latest/JPN/Platforms/GearVR/Debugging/index.html>

*33 詳細条件は不明です。AndroidManifestやosigファイル入りかどうかを見ていると思われます

```
// move bar
MeObject.transform.position =
    new Vector3 (posX,
        MeObject.transform.position.y,
        MeObject.transform.position.z);
}
}
```

前述のスクリプトとそっくりですね^{*34}。

この MeHeadVR スクリプトを Main Camera にドラッグ&ドロップして追加し、追加したスクリプトの Me Object プロパティに、Hierarchy ビューの PongStage/Me を指定します。

ここまで読み進めた読者であれば、どちらのデバイスを使っても開発手法は同じということを理解したでしょう。

Cardboard SDK にしかない Trigger (スライドスイッチ・押し込みスイッチの押下) や Tilt (Cardboard ごとデバイスを縦に 90 度回すことの検出) の API を使ったり、Gear VR にしかない右側面のタッチパッド^{*35}を使ったりしようとすると差分が出てきます。

7.10 おわりに

駆け足でしたが、Unity とスマートフォンで VR コンテンツに入門するというテーマで解説してきました。

VR コンテンツは、表現の手法として多くの分野と関連があります。お絵かきや DTM に代表される、消費者が内容を生成していく CGM 文化や iOS や Android の台頭を支えたサードパーティ製のコンテンツ、スマートフォンを中心に定着したインディーズゲームのマーケットなどです。いずれも VR を利用することで新たな表現の場を形成しつつあります。

もっとハイレベルな Reality を実現する世界に向けて盛り上がりしていくためにも、これらの要素すべてに関係を持つ VR という分野でプロフェッショナル、アマチュアを問わず多くのコンテンツ開発者が増えることが重要です。

このような流れの中、本章を読んで、VR コンテンツ開発者が少しでも増えってくれればいい

^{*34} 頭の向きをとって自機のバーの位置に反映させるという意味では同じなので、当然といえば当然なのですが

^{*35} どの部分を触ってもまずは X=1280, Y=720 という座標を返し、ドラッグしただけ移動した座標を返すつくりになっています

など筆者は考え^{*36}ています。

^{*36} そして、アメリカンなドンパチとゾンビものに負けずに、日本らしい VR コンテンツを増やしたい仲間になつていただけだと大変嬉しいです

著者紹介

第1章 なっぴー / @napplecomputer

寝室にエアコン必要ゼッタイ

第2章、第3章 明日鍵 / @tomorrowkey

最近仕事では Android をやらずに Ruby 書いてます。

第4章 Atsushi Eno / @atsushieno

記事に合わせて作っていたコードが全然出来上がらなくて全部削ったなんてこと全然ないし！

第5章 てし / @teshi04

腐女子向けアプリを作りたい

第6章 摺下 / @eaglesakura

めえめえ

第7章 ようてん / @youten_redo

α 6000 を買いました。スマホより重いのでこれで VR に必要な筋肉を鍛えたいと思っています。

表紙イラスト / shati

ダンジョンで迷子になるだけの人生です

表紙デザイン / siosio

娘が一歳になりました。

編集 / mhidaka

今回も、みんなで頑張りました。楽しんで読んでもらえたなら嬉しいです

Android Masters!

2015 年 8 月 16 日 初版発行 v1.0.0

2015 年 8 月 24 日 電子書籍発行 v1.0.1

著 者 TechBooster

編 集 mhidaka

発行所 TechBooster

(C) 2015 TechBooster