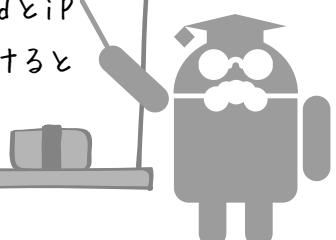


第8章

UIの基礎知識

著：秋葉ちひろ

この節では、ユーザーインターフェースとは何か、なぜそれらを学ぶ必要があるのか、ということを学びます。また、ユーザーインターフェースは、同じスマートフォンでもAndroidとiPhoneでは細かい違いがあります。まずは違いを見つけるところから始めていきましょう。



この節で学ぶこと

- 1 ユーザーインターフェースとは？
- 2 AndroidとiPhoneのユーザーインターフェースの違いと、それらが違う理由

この節で出てくるキーワード一覧

ユーザーインターフェース(UI)
アフオーダンス
タッチデバイス
Androidのガイドライン
Java
Objective-C／Swift



8-1-1 ユーザーインターフェース (UI) とは？

実際にアプリを作っていく上で必要となる、画面のレイアウトやデザイン。これらを、“ユーザーが操作していくもの”という意味も含めて、「ユーザーインターフェース (UI)」と呼びます。たとえば、テキストを入力するフォームや、押すと何かが起こるボタンなどです。

また、ユーザーが操作するわけではないですが、テキストや画像などもユーザーインターフェースの重要な要素のひとつです。

ユーザーインターフェースは、機械と人間とをつなぐ重要な接点であり、何をすれば何が起こるのかということを、瞬時にユーザーが理解できなければなりません。ボタンは押すことができますが、そもそも「押せる」ということを、ユーザーに伝えなければなりません。

これについては、ユーザーに「アフォーダンス」(注釈参照)を与えることが重要だと言われています。



図1:アフォーダンスを利用したユーザーインターフェースの例

図1を見ると、その意味が少しあわかるでしょうか。

アプリやWeb上のボタンは、「押せる」ことを強調するために、少し立体的に見せる工夫をしてあることが多いでしょう。また、クリック時(押されたとき)には凹んだように見せることによって、ユーザーに「ボタンを押した」ことを認識させています。

パソコンとスマートフォンでは、ユーザーインターフェースは大きく異なる

普段使っているパソコンにも、たくさんのアプリケーションがあります。文書の作成であればWord、表計算ならExcel、写真の加工であればAdobe Photoshopなどのグラフィックツール、他にもたくさんあるでしょう。

そういったパソコンで使うアプリケーションのインターフェースも、いろいろとアフォーダンスを与える工夫がなされています。ですが、これらはキーボードとマウスで使うことに最適化されて考えられています。

少し前まではそれだけでもよかったのですが、2007年にiPhoneが発売されてからは、ガラッと様子が変わりました。iPhoneやAndroidをはじめとしたスマートフォンは、指で触れることによって操作するからです。

アプリやWebのデザイナーは、キーボードとマウスを使うようなユーザーインターフェースを作ることには慣れており、ノウハウの蓄積もありました。しかし、スマートフォンなどのタッチデバイスにおけるユーザーインターフェースのデザインは、誰もが初めての経験でした。しかも、パソコンとは違って画面サイズもかなり小さいので、使える領域も限られています。このような制約の中でユーザーインターフェースをつくることは、多くの試行錯誤を必要としました。

ですが、iPhoneの登場から7年が経った今、タッチデバイスでのユーザーインターフェースのノウハウも貯まりつつあります。

「アフォーダンス」(affordance)とは、物はどう取り扱ったらよいかについての強い手がかりを示してくれるものです。たとえば、ドアノブがなくて、その代わりに平らな金属片が付いたドアは、その金属片を押せばよいことを示しています。逆に、引き手のついたタンスは、引けばよいことを示しています。これらは体験に基づいて、説明なしで取り扱うことができます。1988年、認知科学者のナルド・ノーマンはデザインに関する認知心理学的研究の中で、モノに備わった、ヒトが知覚できる「行為の可能性」という意味で、アフォーダンスという言葉を用いました。この文脈によるアフォーダンスという語義が、ユーザーインターフェースやデザインの領域において使われるようになりました(Wikipediaより抜粋・編集)。

そして昨今においては、ユーザーインターフェースの設計は、デザイナーもプログラマーも、みんなが知っておかなければならぬ重要なものとなりつつあります。実際にアプリケーションを作るときにも避けては通れない道です。

ただ、その重要性のわりには、体系的に教えることが難しい分野でもあります。なぜなら、暗黙的なノウハウ、経験値に支えられているからです。興味のある方は、ぜひ書店などでデザインの専門書を読むことをおすすめします。きっと、世の中に対する見方が変わるでしょう。

そこまでしなくとも、普段みなさんが使っているアプリケーションで（パソコンでもスマートフォンでもかまいません）、どこにどういったかたちでアフォーダンスが使われているか考えてみましょう。よいユーザーインターフェースの勉強になりますので、ぜひ実践してみてください。



8-1-2 Android アプリのユーザーインターフェース

さて、一般的なユーザーインターフェースについては前述のとおりですが、Androidアプリを作るときには、それに沿ったユーザーインターフェースを知っておかなければなりません。なぜなら、Android OSを提供しているグーグルが、「Androidアプリはこう作りなさい」というガイドラインを公開しているからです。

The screenshot shows the top navigation bar with 'Android Developers' dropdown, 'Design', 'Develop', 'Distribute', a search bar, and a menu icon. Below this is a large image of a smartphone and a tablet displaying the Android interface. To the right, there's a section titled 'L Developer Preview' with a description: 'The L Developer Preview lets you design and develop against the next major release of Android. Take the time to test and build your app before the platform officially launches.' A 'Learn More' button is present. At the bottom, there are three cards: 'Building Apps for Wearables' (Alice Yang, Almost home!), 'Material Design' (Learn how to apply material design to your apps.), and 'Android Studio' (Learn about the new features in the beta release of our new IDE.). At the very bottom, there are links: 'Get the SDK', 'Browse Samples', 'Watch Videos', and 'Manage Your Apps'.

図2:グーグルが定めているAndroidアプリのガイドライン(<http://developer.android.com>)

このガイドラインの中には、「Design」「Develop」「Distribute」という3つのカテゴリーがあり、Androidのすべてのユーザーインターフェースを理解するには、それを実際にコードで書いて実現できなければなりません。

しかし、今の段階ではそこまでせずとも、まずは「どんなユーザーインターフェースがあるのか」を、だいたいで良いので知っておきましょう。概要だけでも、このガイドライン

を知っていなければ、アプリの設計ができないからです。

もちろん、しっかりとしたAndroidアプリを作りたいならば、早いうちからこのガイドラインを読んでおくべきです。英語なので読めない——とは言わず、Googleの翻訳機能などを使ってでもいいので、頭に入れておくようにしましょう。

もし、内容がよくわからなくても、ここにある図だけでも見ておくとよいでしょう。これらはすべて、ガイドラインに沿ったAndroidアプリのデザインになっていますので、実際に設計するときに役に立ちます。

少なくとも目を通しておかなければならぬのは、以下の3つです。

- ・Design → Style(図3)
- ・Design → Patterns(図4)
- ・Design → Building Blocks(図5)

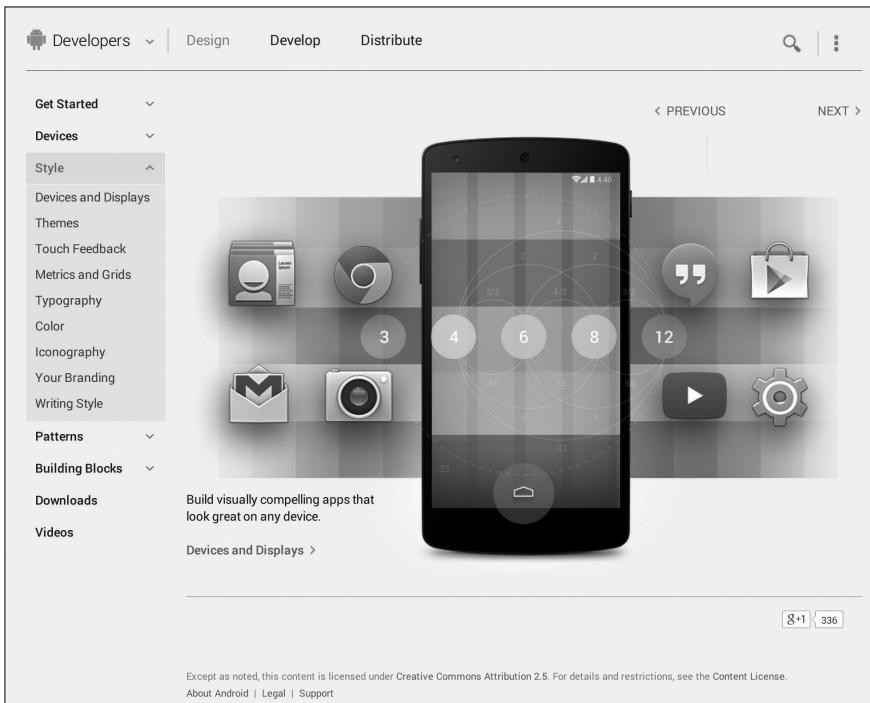


図3:ガイドラインの「Style」

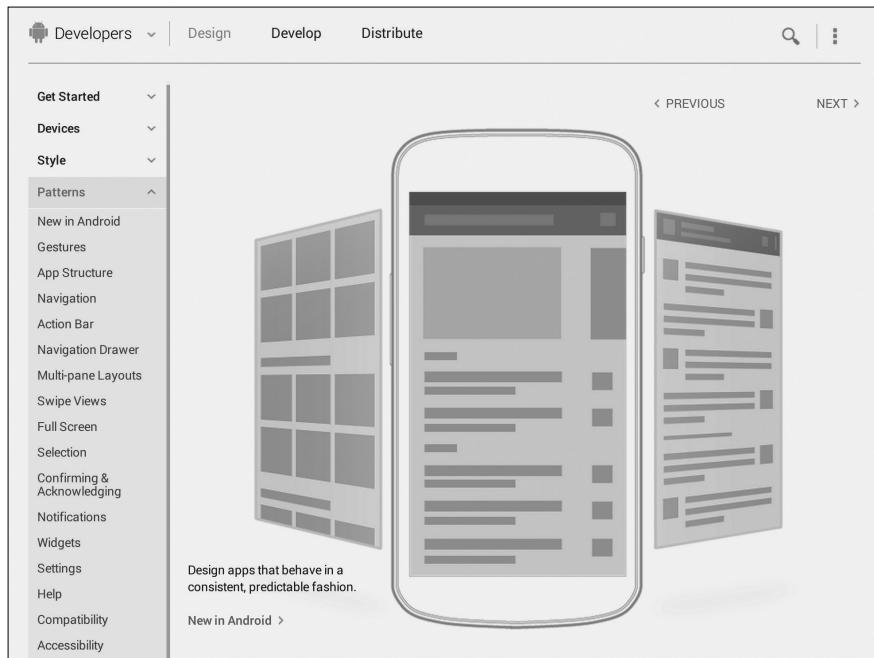


図4:ガイドラインの「Patterns」

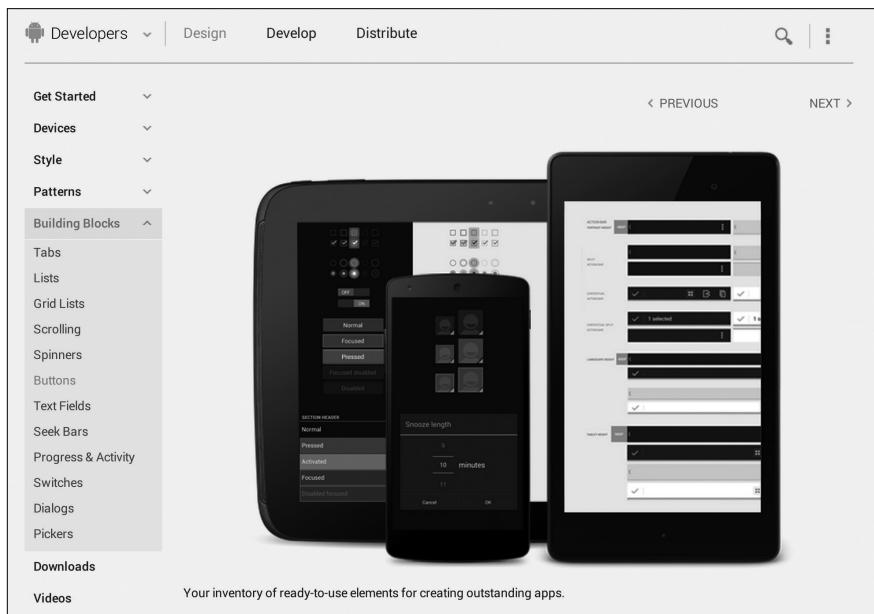


図5:ガイドラインの「Building Blocks」



8-1-3 iPhoneアプリとAndroidアプリのインターフェースの比較

Androidアプリだけでなく、もちろんiPhoneアプリにもガイドラインはあります。

iPhoneアプリのほうが厳しくて、アプリを公開するときには、ガイドラインに準拠しているかという審査があります。iPhoneアプリの場合は、少しでもガイドラインから逸脱した部分があれば、リジェクトされてアプリストアで公開することができません。

それではAndroidのほうは審査がないのかと、安心してはいけません。AndroidアプリのGoogle Playでの公開は、事後審査になります。公開時の審査は基本的にはありませんが、公開後に、ウェブの検索エンジンと同様、Google Playでキー

ワードなどをクロールして、NGワードなどがあれば警告されるようになっているようです。警告を無視し続けると、Googleアカウント自体が停止され、Googleに関するすべてのサービスが一時利用できなくなってしまうので注意してください。

また、iPhoneアプリとAndroidアプリでは、推奨されるユーザーインターフェースにも違いがあります。これは、それぞれのOSを構成しているプログラミング言語、フレームワークの思想の違いによるものです。

iPhoneアプリは「Objective-C」または「Swift」(Swift、iOS 8以降)という言語や、インターフェースビルダーという専用のレイアウトツール、ストーリーボードという構組みを基準にインターフェースが定められています。

一方、Androidアプリを作成する言語はJavaであり、レイアウトはXMLで作ることになります。こちらも、Android OSに合うようなインターフェースが定められています。

なぜこのようなことを説明するかというと、iPhoneアプリのインターフェースをそのままマネしたようなAndroidアプリをよく見かけるからです。

iPhoneアプリのインターフェースをAndroidアプリでマネする必要は、まったくありません。前述のとおり、それぞれのOSによって構成や思想が違い、さらにプログラミング言語やツールによって最適化できるインターフェースも違います。無理やりどちらかのOSのインターフェースに合わせることは、アプリを作っていくうえで無駄な設計が多くなってしまうことに注意してください。

いちばんわかりやすいのは、設定アプリ(設定メニュー)を比較することです。



図6:Androidの設定アプリ(メニュー)の画面



図7:iPhoneの設定アプリ(メニュー)の画面

図6と図7では、いろいろな相違点があります。どういったところが違うか、グループで話し合い、箇条書きで書き出してみましょう。

アカウントが停止されることを「アカBAN」(Account ban)といい、開発者が最も恐れていることです。「リジェクトされないAndroidアプリの運用」(<http://technica.spee.jp/803>)などを参照。

Androidアプリに設定画面を設けるとしたら、このようなデフォルトの設定アプリに近いデザインの採用が推奨されています。また、同じようなデザインであれば、事前に用意されたデフォルトのコードを使い、少しだけカスタマイズをするだけで実装できるでしょう。



8-1-4 その他のアプリの比較

他にも、iPhoneとAndroidの両OSに同じサービスを提供しているアプリのユーザーインターフェースを比較すると、それぞれのOSのインターフェースを学ぶ上で大変参考になります。Facebook、Twitter、Foursquare、Dropboxの例を挙げておきます。これらに対しても、それぞれの相違点を箇条書きで書き出してみましょう。

Facebookアプリのインターフェース



図8:AndroidのFacebookアプリ



図9:iPhoneのFacebookアプリ

Facebookアプリのインターフェースの相違点を、以下に書いてみましょう（筆者の回答例はP.17を参照）。

MEMO

Twitterアプリのインターフェース



図10:AndroidのTwitterアプリ



図11:AndroidのTwitterアプリのメニュー画面



図12:iPhoneのTwitterアプリ

Twitterアプリのインターフェースの相違点を、以下に書いてみましょう。

MEMO

Foursquareアプリのインターフェース



図13:Android のFoursquareアプリ

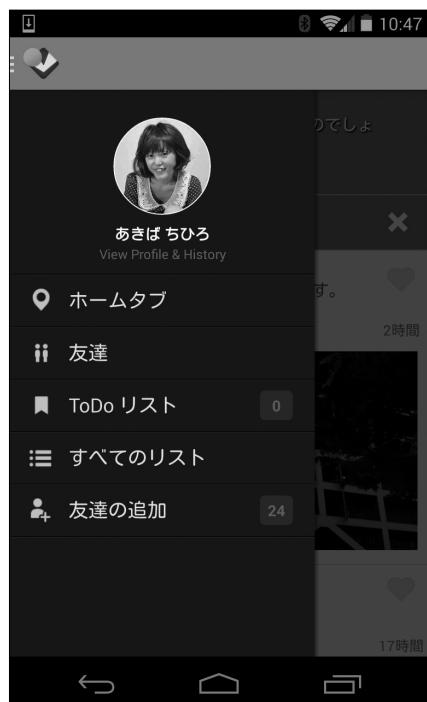


図14:Android のFoursquareアプリ、展開時



図15:iPhoneのFoursquareアプリ

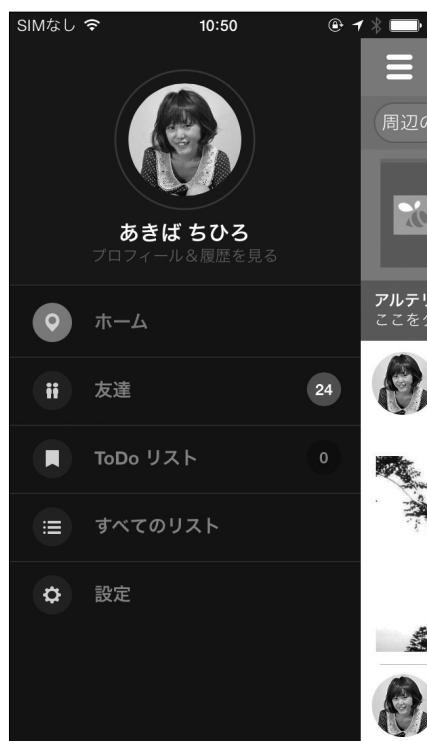


図16:iPhoneのFoursquareアプリ、展開時

Foursquareアプリのインターフェースの相違点を、以下に書いてみましょう。

MEMO

Dropboxアプリのインターフェース

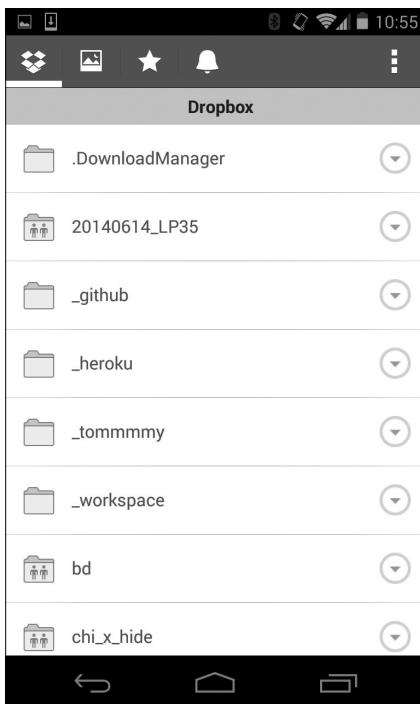


図17:AndroidのDropboxアプリ

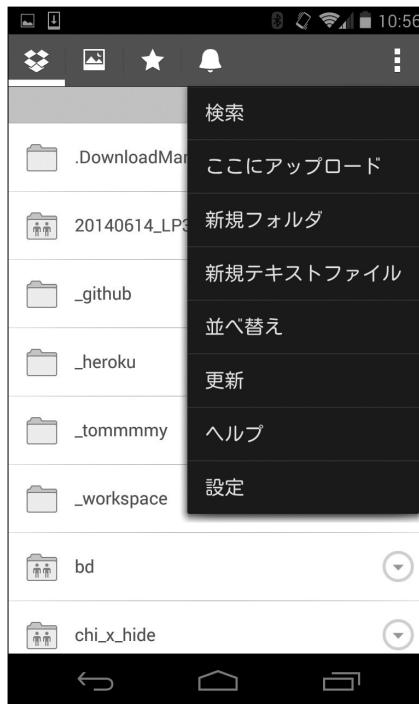


図18:AndroidのDropboxアプリのメニュー



図19:iPhoneのDropboxアプリ



図20:iPhoneのDropboxアプリのメニュー

Dropboxアプリのインターフェースの相違点を、以下に書いてみましょう。

MEMO

このように見てみると、AndroidのユーザーインターフェースにはAndroidの、iPhoneにはiPhoneの特徴が見えてきます。これらをしっかりと認識し、AndroidとiPhoneをごちゃまぜにしないようにしましょう。

こういった比較は、普段使っているアプリでもできるので、アプリをただ使うだけではなく、どういうインターフェースが使われているか、注意してみましょう。そうすると、アプリに対する見方が180度変わってきます。

また、Androidアプリを作るならば、普段からAndroidを使って、その習慣に慣れておくことが望ましいです。ときどき、Androidアプリのデザインをしているデザイナーが（プロのデザイナーが、です）、実はAndroidを使ったことすらなく、Androidのユーザーインターフェースについてもまったく知らないでビックリすることがあります。

アプリ開発は、プログラムの技術があればよい、というものではありません。今や、デザインのほうが重要だとも言われる時代です。使う人に「使いやすい」「使って楽しい」と言ってもらえないとい、すぐに飽きられてしまいます。

アプリを作ることも大切ですが、その前にまずこの章に書かれていることをしっかりと理解し、ガイドラインを読み、AndroidというOSのユーザーインターフェースや習慣を知っておくべきです。今からそれに気をつけていれば、きっと素晴らしいアプリが作れるようになるでしょう。

まとめ

それぞれのインターフェース比較について

本節で見てきたユーザーインターフェースの比較について、筆者の考える回答を挙げておきます。

■設定アプリ

- ・Androidは黒背景に白文字、iPhoneは白背景に黒文字
- ・Androidには見出しがあるが、iPhoneは見出しがなく、グレーの空白行がある
- ・iPhoneのアイコンはカラーである
- ・iPhoneは、リストの右端に、右向きの矢印がある
- ・iPhoneは、リストの右端に、設定された値が入っているものがある

■Facebookアプリ

- ・iPhoneはタイトルバーに検索ボックスがあるが、Androidは検索ボタンしかない
- ・Androidは、左上にロゴ、そのとなりに今いる画面のタイトルがあるが、iPhoneはない
- ・タブによるページ切り替えバーが、Androidは上側にあるが、iPhoneは下側にある
- ・それとは反対に、「近況」「写真」「チェックイン」のアクションを示すボタンバーは、Androidは下側にあるが、iPhoneは上側にある
- ・ボタンバーについて、Androidは青い背景色に白い字だが、iPhoneはグレーの背景色にグレーの字
- ・iPhoneは、タブのアイコンの下に添え字があるが、Androidはない

■Twitterアプリ

- ・iPhoneはタブによるページ切り替えが「タイムライン」「通知」「メッセージ」「アカウント」だが、Androidではこれらは右上のバー内にアイコンとして配置されている
- ・Androidは、タブに「ホーム」「見つける」「アクティビティ」があるが、iPhoneはない(厳密にはiPhoneではこれらを「ホーム」下の3つの円で切り替える)
- ・Androidは、いちばん下に新規投稿エリアがあるが、iPhoneはない。代わりに、右上に新規投稿ボタンがある

■Foursquareアプリ

- ・Androidは左上にロゴマークがあるが、iPhoneはない
- ・iPhoneはタイトルバーに検索ボックスがあるが、Androidは検索ボタンしかない
- ・iPhoneは「周辺のスポット」「全世界」の切り替えと地図のマークがあるが、Androidにはない
- ・「Swarm by Foursquareは何なのでしょう？もっと詳しく。」が、iPhoneは枠で囲まれているが、Androidは囲まれていない
- ・「アルテリア・ベーカリーにいますか？」の右端にある×印の色が違う
- ・タイムラインの部分の、名前と場所名の色が、iPhoneはブルーである
- ・iPhoneはハートマークが丸で囲われているが、Androidは囲われておらず、少し大きさが大きい
- ・メニューを開いたときの「ホーム」の色が、iPhoneはブルーである
- ・メニューを開いたときのアイコンが、iPhoneはグレーで囲まれている
- ・メニューを開いたとき、「友達の追加」がAndroidにはあり、「設定」がiPhoneにある

■Dropboxアプリ

- ・iPhoneはタイトルバーの下部に検索ボックスがあるが、Androidはメニュー内に検索メニューがある
- ・iPhoneはタブバーが下にあるが、Androidではそれに相当するものは左上になっている
- ・メニューは、Androidは右上に出ているが、iPhoneは上から出ている
- ・メニューの背景色が、Androidは黒だが、iPhoneは白い
- ・iPhoneには、インデックス（「か」「た」「B」「C」などの見出し）があるが、Androidには、ない
- ・「設定」が、iPhoneはタブバー内にあるが、Androidはメニュー内に入っている
- ・Androidのリストの右端には下向きのマークがあるが、iPhoneにはない

これらの違いを見ていると、

- ・iPhoneは、タブバーは常に下にある
- ・Androidはタブにあたるものは常に上にある
- ・iPhoneは、検索ボックスが出ているものが多い

などの、それぞれのユーザーインターフェースの共通性を発見できます。これらのアプリは、「実際にそれぞれのガイドラインに従って作るところなる」という好例です。また、こういったインターフェースを見て、「なぜこの配置になったのか」を考えることも大事です。普段アプリを使うときにも、こういったことを考えるようにしましょう。

8-2 UIの基礎知識（1）

著：秋葉ちひろ

この節では、ADTに付属する「Graphical Layout」を使ってレイアウトを完成させます。細かい部分に気を取られず、Androidにはどのようなパーツがあり、どんなときに使うのかをざっと把握することが目的です。



この節で学ぶこと

- 1 Graphical Layoutがどんなものかを知る
- 2 Graphical Layoutを使ってレイアウトを作成する
- 3 ただパーツを配置するだけではなく、ユーザーが使いやすいことを考える

この節で出てくるキーワード一覧

Graphical Layout

XML

Button

Text/Text Field

IME

RadioGroup



8-2-1 Graphical Layout を使ってレイアウトを作ってみよう

ユーザーインターフェースのことを全般的に理解できたら、実際にそれらを作っていくようにしなければなりません。

慣れてきたら自分でXMLを作成するのですが、Androidにどんなパーツがあるのかがまだわからないうちは、いきなりXMLを書くのは少し難しいでしょう。

最初は、ドラッグアンドドロップで簡単にインターフェースを作ることができる「Graphical Layout」で、Androidにはどのようなパーツがあるのかを見ていきます。

Webサイトを作るときは、HTMLとCSSを使ってユーザーインターフェースをデザインしますが、Androidプロジェクトでは、それがXMLになります。

Androidアプリのユーザーインターフェースは、プロジェクトの「resフォルダー」内のXMLファイルに記述していきます。

新規プロジェクトを作成し、「Graphical Layout」を表示する

ADTを起動し、新たにAndroidプロジェクトを作成します。ここでは、「UIBasic1」という名前の新規プロジェクトを作成し(図1)、「Empty Activity」を選択します(図2)。

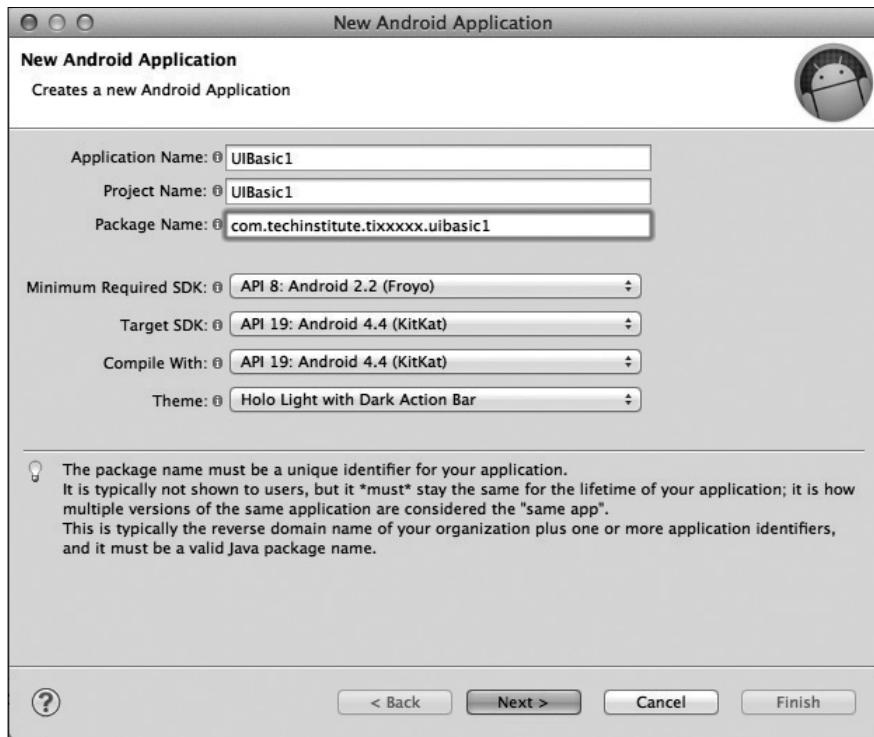


図1:UIBasic1という名前で新規プロジェクトを作成

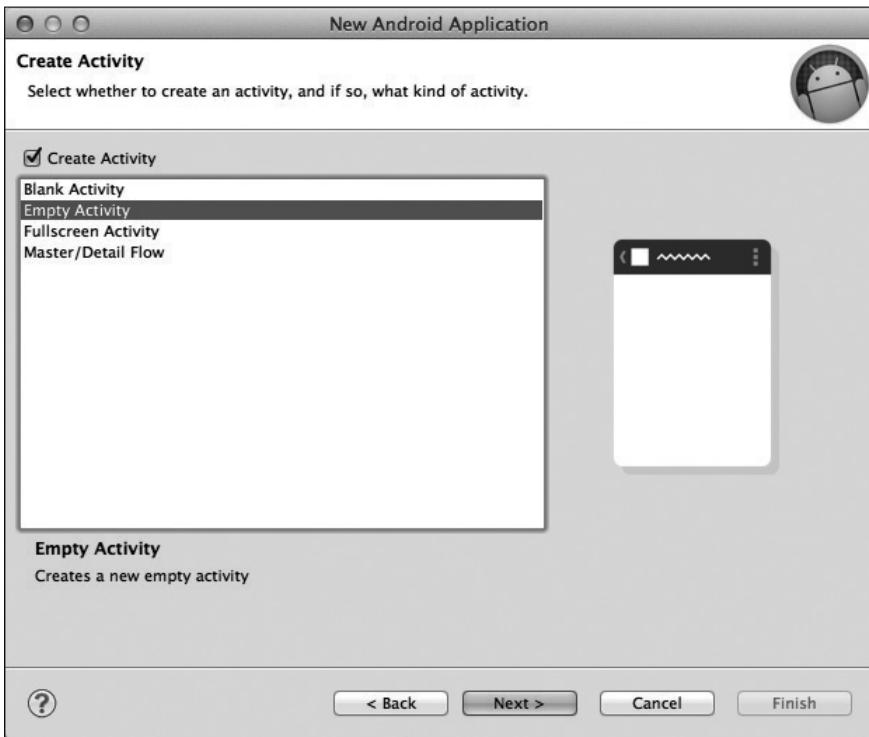


図2:Empty Activityを選択

そうすると、「MainActivity.java」「activity_main.xml」の2つのファイルが開かれた状態になります。画面上側のタブで「activity_main.xml」を選び、下側のタブが「Graphical Layout」になっていることを確認します。

Graphical Layout の見方を確認しよう

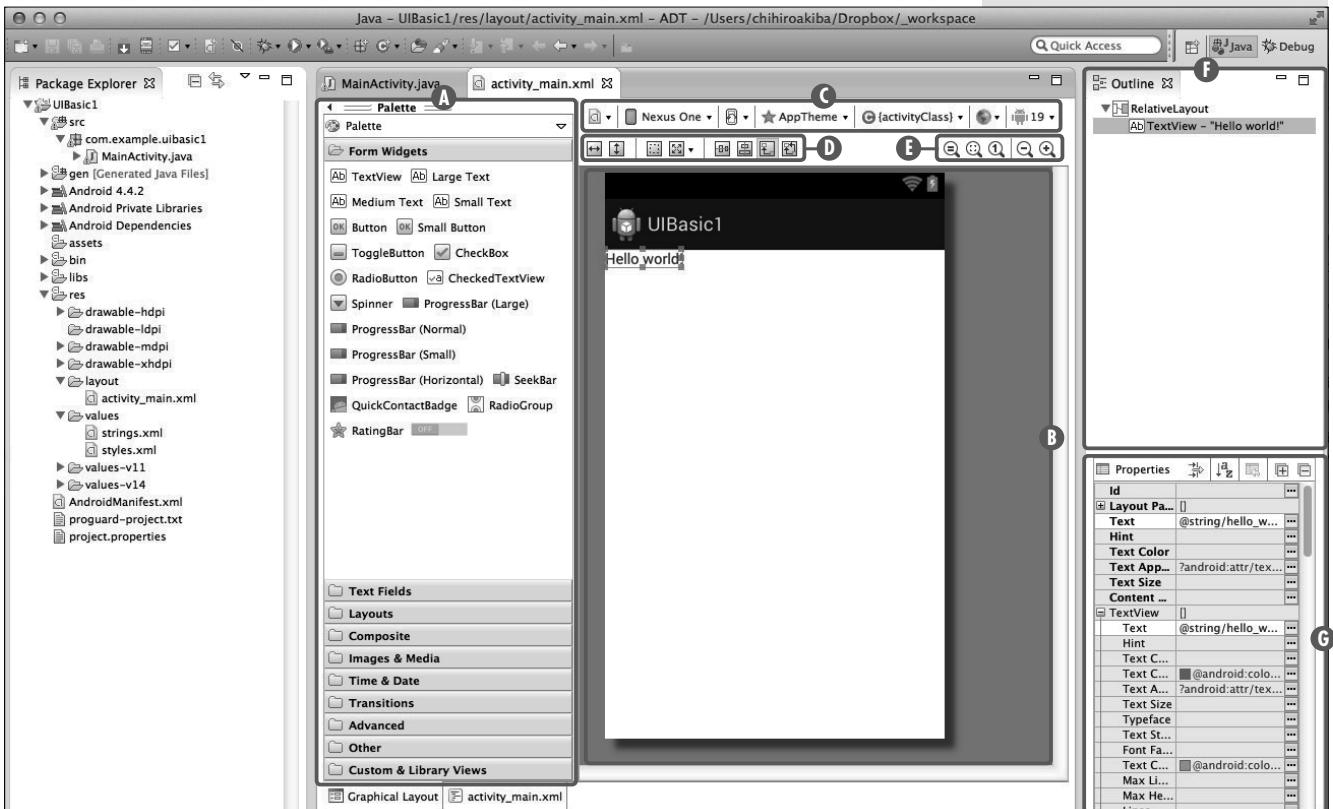


図3:Graphical Layoutの画面

Graphical Layout を開くと、Androidの画面が大きくあらわれます。初期状態では、「Hello world!」というテキストがひとつ書かれているのがわかります(図3)。

他にもいくつかのパネルがあります。

- Ⓐ パレット：各パーツが入っている
- Ⓑ プレビュー：現在設定されているものがプレビューされます
- Ⓒ プレビューに関する端末の設定：プレビューを表示する端末の種類やバージョンを設定します
- Ⓓ パーツのレイアウトに関する設定：パーツのレイアウト設定をします
- Ⓔ プレビューに関するビューの設定：プレビューを表示する大きさを設定します
- Ⓕ アウトライン：配置したパーツがどういう階層構造になっているのかを設定します
- Ⓖ プロパティ：配置したパーツの詳細設定をしていきます

最初からすべてを把握するのは大変です。必要なものはそのときどきで説明していきますので、まずはパーツを配置してみることからはじめていきましょう。

パーツを配置し、カスタマイズしてみよう

Ⓐ のパレットから好きなパーツを選んで、Ⓑ のプレビュー領域にドラッグアンドドロップしてみましょう。

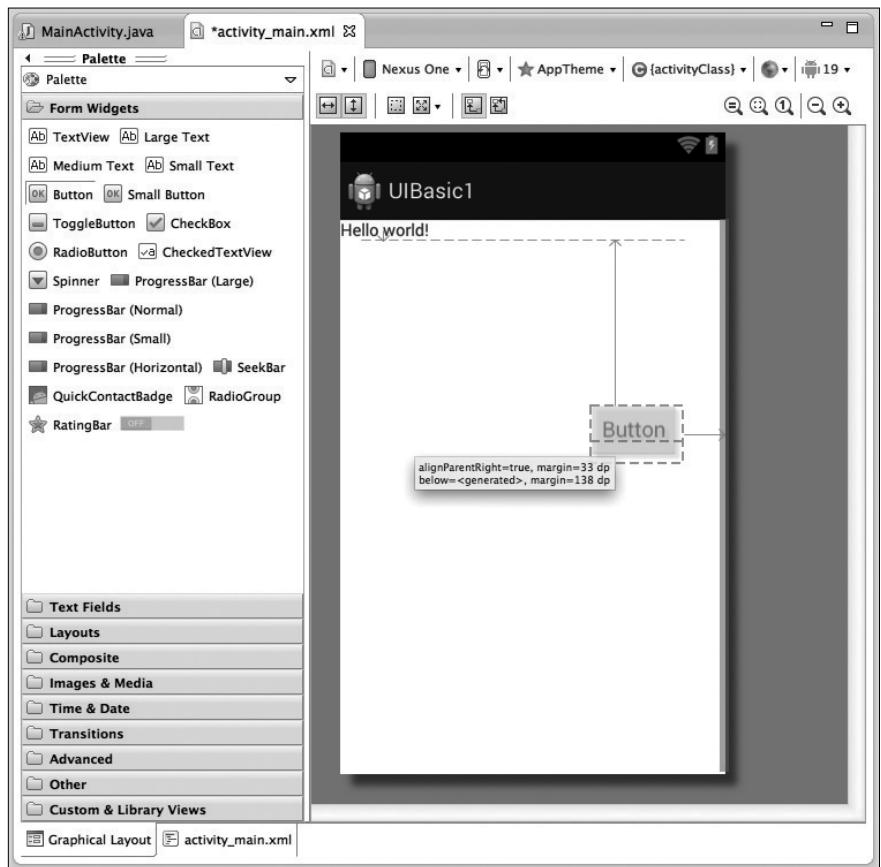


図4:Button/パーツをドラッグ中

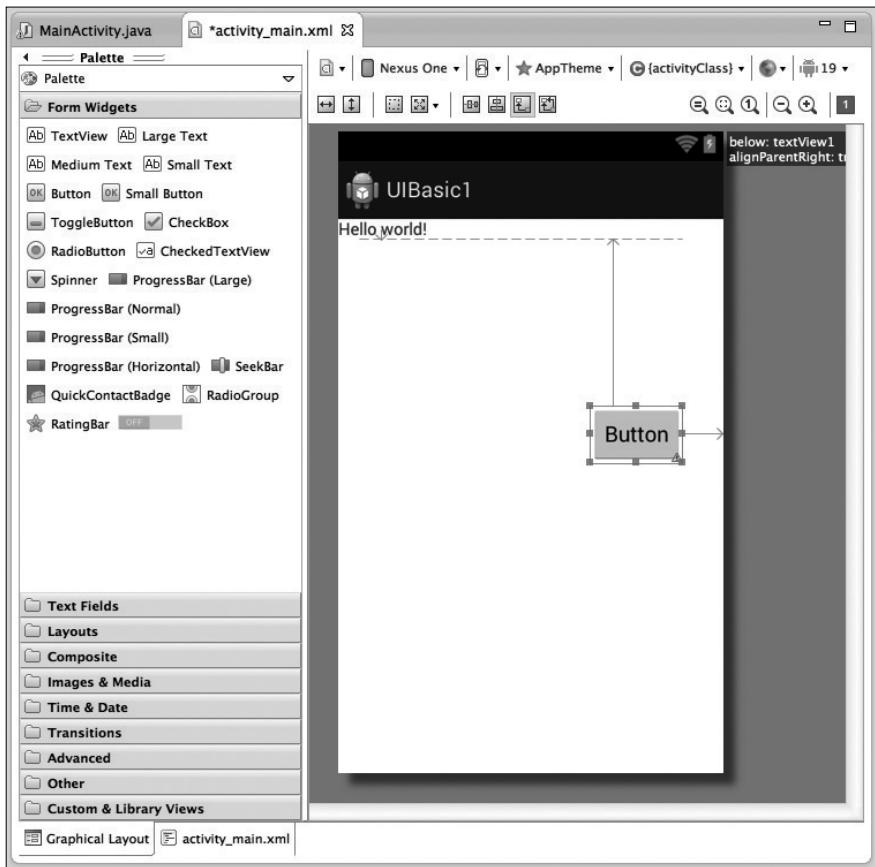


図5:バーツのドラッグ完了

図4、図5では、①パレットの「Form Widgets」の中から「Button」を選んでドラッグアンドドロップし、Buttonを配置しました。

ここで配置したボタンは、「Button」と書かれています。このままではいったい何のボタンかわかりませんので、文字を変更します。

プレビュー画面のボタンを選択した状態で、プロパティパネルを見ると、ボタンのプロパティがたくさん出ています(図6)。

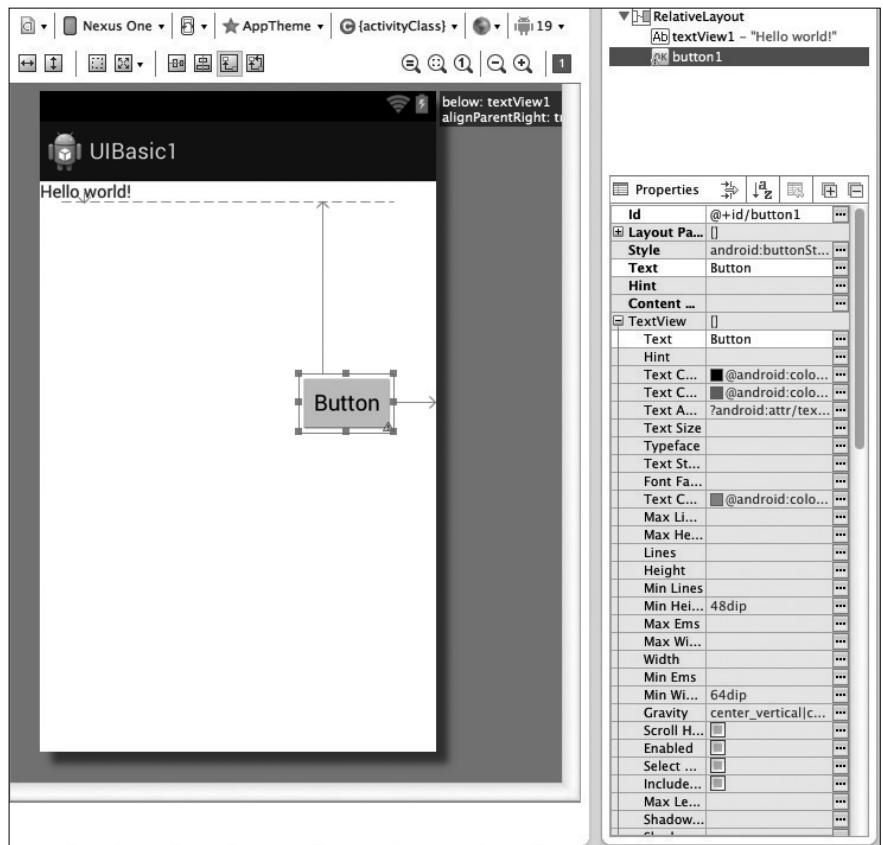


図6:ボタンのプロパティ

この中にある「Text」という項目が、ボタンのラベルになっていますので、右の列の「Button」と書かれてあるところをダブルクリックし、「送信」に変えてみましょう(図7、Textは2ヶ所ありますが、どちらも同じです)。



図7:ボタンのラベルを「送信」に変える

そうすると、プレビューエリアのボタンのラベルも変わりました(図8)。



図8: プレビュー領域のボタンのラベルも「送信」に変わる

このように、プロパティパネルにある値をいろいろと変えていくことで、パーツのカスタマイズができます。

8-2-2 Graphical Layout でどんどんパーツを使ってみよう

各パーツは、パレットエリアにカテゴリーごとにフォルダ一分けされています。他のフォルダーの中も見てみましょう(図9～)。項目の名前を見てもいまいちピンとこない場合は、実際にプレビュー領域にドラッグアンドドロップして、パーツを配置してみるとよいでしょう

中には、図9のように右下に小さな赤い「×」印が出るものがあります。これは、Eclipseでのエラーを示していますので、このままではビルドできない状態になっています。ここではあまり気にしなくともいいので、どんなパーツかがなんとなくわかったら、エラーの出るパーツは削除して、どんどん先に進みましょう。

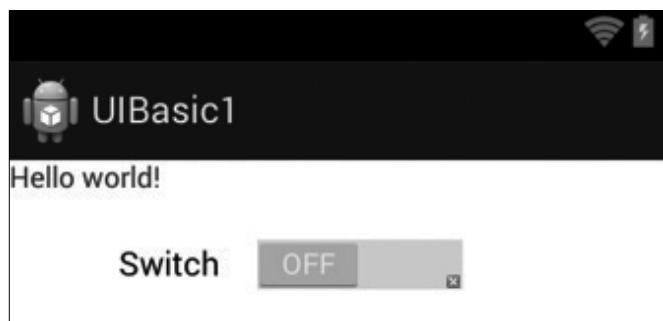


図9:「エラーが出たパーツの例。右下に小さい「×」印が表示されている

なお、エラー「×」印のところにマウスを合わせると、エラー内容がでできます。図9のスイッチでは「View requires API level 14 (current min is 8): <switch>」というエラーが出ています。これはどういうことかというと、「スイッチ」というパーツはAPIレベル14以降で使用可能なのですが、このプロジェクトでは「APIレベル8以上」という設定になっている(新規プロジェクト作成時にそう指定していた)ので、

Androidに慣れていない人は、どんどんパーツを配置してみて、どんなものがあるのかをしっかり確認しましょう。配置ができたら、実機でも確認してみましょう。実機に表示して指で触ってみることでわかることもたくさんあります。

エラーとなっているわけです。

Graphical Layoutのパート一覧



図10:「Form Widgets」
テキストビュー やボタン、トグル
ボタン(ON/OFFの切り替えをす
るもの)、チェックボックス、ラジ
オボタン、プログレスバー(円形
にぐるぐる回るものや水平線状
のもの)、シークバー(スライド
バーともいう)、レーティング
バー やスイッチなどが入ってい
ます

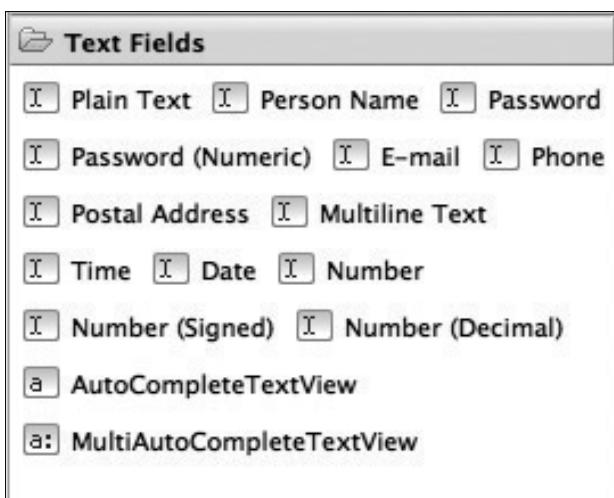


図11:「Text Fields」
いろいろな形式のテキストフィー
ルドが入っています

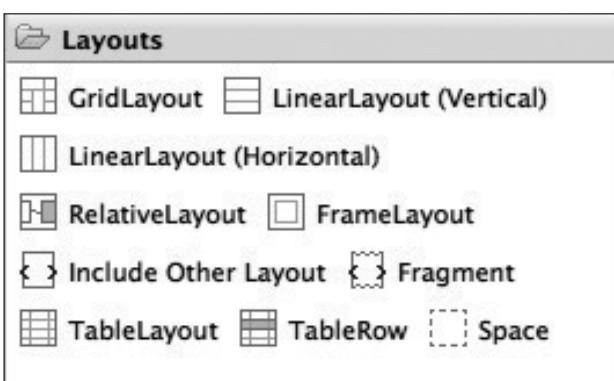


図12:「Layouts」
レイアウトに関するパートが入っ
ていますが、Graphical Layout
だけではちょっと使いづらいも
の、もしくは使えないもので
ので、XMLといっしょに学習してい
きましょう

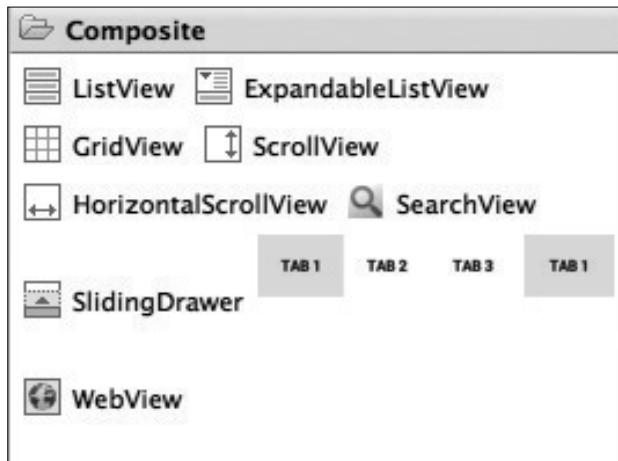


図13:「Composite」
同じようにレイアウトに関する
パーツが入っていますが、プログ
ラムと連携しなければならない
ものが多いです



図14:「Images & Media」
画像の表示や、音声や動画を表示
するものが入っています。



図15:「Time & Date」
時計やカレンダー、また日時を指
定するピッカーが入っています

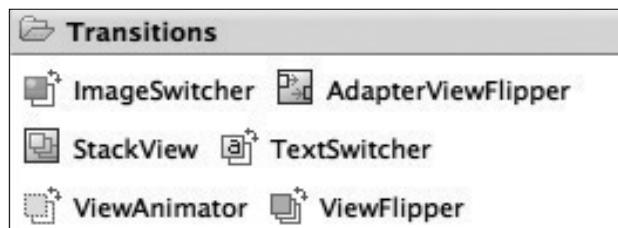


図16:「Transitions」
動きに関するパーツが入ってい
ますが、プログラムと連携しなけ
ればならないものが多いです

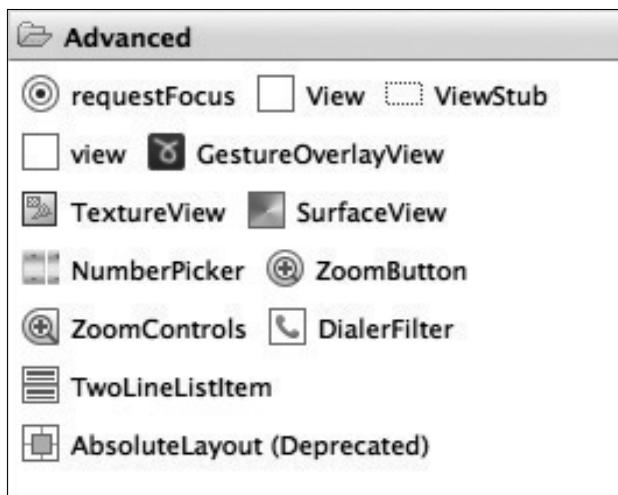


図17:「Advanced」
その他のパーツが入っています
が、Graphical Layout単独では
動かないものばかりです。「Zoom
Controls」などはそのまま貼り付
けることができますが、このま
だと触っても何も起こりません

パレットの中に入っているパーツは、配置するだけで、アプリ内でそのまま使えるものもあります。たとえば、テキストや画像などはパーツを表示するためですので、配置するだけで十分、役目を果たします。

しかし多くの場合はそうではなく、たとえば、テキストを入力してボタンを押したらどうなるか、というところをプログラミングしなければ、アプリとして成立しません。配置するだけでも「レイアウトデザイン」として見せかけの画面はできあがりますが、アプリとして何か使えるようになるわけではないのです。

また、プログラミングによって意味付けをしなければ、レイアウトデザインとしても使えないパーツがたくさんあります。「Composite」フォルダーの中などはそういうものばかりです。

ここでは、ボタンを押しても何も起こらないような、見せかけだけのレイアウトを作っているんだ、ということを念頭にすすめていきましょう。



8-2-3 実習1: Graphical Layout を使ってインターフェースを作成する

Graphical Layoutを使って、次のユーザーインターフェースを作成してみましょう(図18)。

The screenshot shows the Android Studio Graphical Layout editor with the title bar "UIBasic1". The main area displays a "プロフィール設定" (Profile Settings) screen. It includes fields for "お名前" (Name), "Email", "お電話番号" (Phone Number), and "パスワード" (Password). Below these are gender selection options: "性別" (Gender) with radio buttons for "男性" (Male), "女性" (Female), and "どちらでもない" (Neither). At the bottom, there's a section for "持っている端末" (Devices I own) with checkboxes for "iPhone", "Android", and "その他" (Other). A large gray button at the bottom is labeled "登録する" (Register).

図18:実習として完成させるサンプルインターフェース

基本的には、あてはまるパーツをパレットからドラッグアンドドロップして配置することで、完成します。ただし、いくつかの注意点があります。

1. Text Fieldsの種類

このサンプルには、

- ・お名前
- ・Email
- ・お電話番号
- ・パスワード

の4つのテキストフィールドがあり、それぞれ「Text Fields」フォルダーの中から選んで配置しますが、個々の「Input Type」は用途に合わせたものを選んでくるのがよいでしょう。

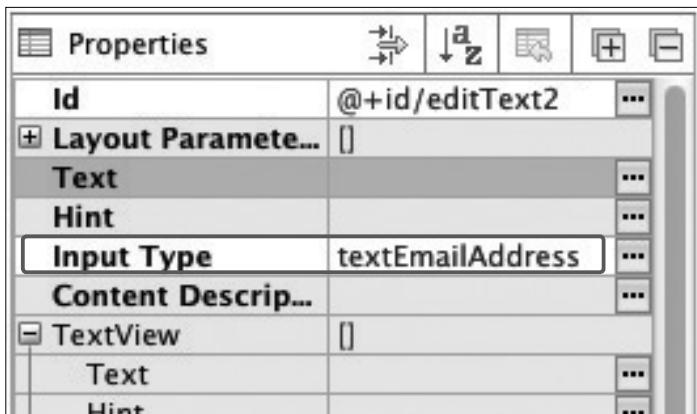


図19:E-mailでは、Input Typeの値が「textEmailAddress」になっている

また、それぞれのテキストフィールドをタップすると、IME(日本語入力)のモードが変わることを確認しましょう。



図20:日本語入力モード



図21:英語(Eメール)入力モード



図22:数字入力モード



図23:英語入力モード

ここでは、IMEに「Simeji」を使っています。自動で表示させるためには、IME側にも制御する仕組みが必要となります。

通常の英数字キーボードには、「@」は出ておらず、Shiftキーなどを押して切り替えると出てくる場合が多いのですが、Emailを入力するときには「@」が必須なため、あらかじめ表示されているほうが親切です。

こうすることで、Android OSが最適なIME(日本語入力)キーボードを自動で表示することができます。

たとえば、「Plain Text」のときは通常の日本語入力キーボードですが、「Email」にしたときは英数字キーボードで、しかも「@」も表示されているのがわかります。

また電話番号は、Input Typeを「Phone」にすれば数字キーボードになります。パスワードも「Password」にすると、入力済みの文字が「●」で表示され、見えなくなるようになっています。

すべて同じ「Plain Text」にしても間違いではないのですが、もし自分が入力する立場になったときのことを考えると、メールアドレスを入力したいときに、日本語キーボードが表示されていたら、英数字キーボードに切り替えるのは手間ですよね。

Input Typeで表示させるキーボードを切り分けることによって、よりユーザーが使いやすいインターフェースを表現できるのです。

2. ラジオグループ

ラジオボタンは、ひとまとまりのグループにおいて、1つの項目しか選択できない状態でなければなりません。

そうするためには、「Form Widgets」の中の「RadioGroup」を選ぶべきです。「RadioButton」を選んでしまうと、それぞれが単独で選べることになってしまいますので、注意しましょう(図24)。

- 男性
- 女性
- どちらでもない

図24:ラジオボタンをそれぞれ単独ですべて選ぶことができる状態。こうならないように注意

3.「登録する」というボタン

普通にボタンを置いた状態では、ボタンの横幅は文字数に応じたものになってしまいます(図25)。これでは、ボタンが小さくて押しづらくなってしまいます。



図25:ボタンの横幅が、「登録する」の4文字分しかない

このボタンは、すべての項目を入力したとの「入力完了」という意味も含めて押すものなので、横幅いっぱいに大きく配置しましょう。

まずはボタンを配置し、「登録する」というラベルに変更したあとに、マウスをうまく使って横幅いっぱいに伸ばします(図26)。



図26:ボタンの横幅を調整する

ここまで完成したら、次はXMLでインターフェースを作っていきます。

8-3 UIの基礎知識（3）

著：秋葉ちひろ

Graphical Layoutでは「ドラッグアンドドロップ」で「だれでも」レイアウトを作ることができますが、細かいレイアウトの設定は困難です。より細かい設定はXMLを使って作ります。この節では、XMLを使って一般的によく使うパートを作れるようになります。



この節で学ぶこと

- 1 XMLの役割
- 2 XMLを書いて簡単なインターフェースを作成する

- ・ボタン
- ・テキスト
- ・画像
- ・テキストフィールド

この節で出てくるキーワード一覧

<Button>	wrap_content
<TextView>	match_parent
<ImageView>	drawable
<EditText>	
android:layout_width	
android:layout_height	



8-3-1 そもそも、なぜXMLによるレイアウトが必要なのか？

6章では、Java(MainActivity.java)でボタンをつくり、背景を設定したりしたと思います。

Javaでボタンをデザインした例

```
Button btn = new Button(this);
btn.setBackgroundColor(Color.GREEN);
setContentView(btn);
```

ボタンというインターフェースは、前ページのコードのようにJavaファイルでももちろん生成できますが、できるだけXMLで記述することが推奨されています。

アプリを作っていくためには、プログラムが絶対に必要になります。たとえば、ボタンを押すと別の画面に移動するようにするために、ボタンに対して「イベントリスナー」を登録し、そのリスナーを呼び出して別の画面に移動する処理を書かなければなりません。つまり、ユーザーが何かを操作するということには、必ずプログラムが必要であるということになります。

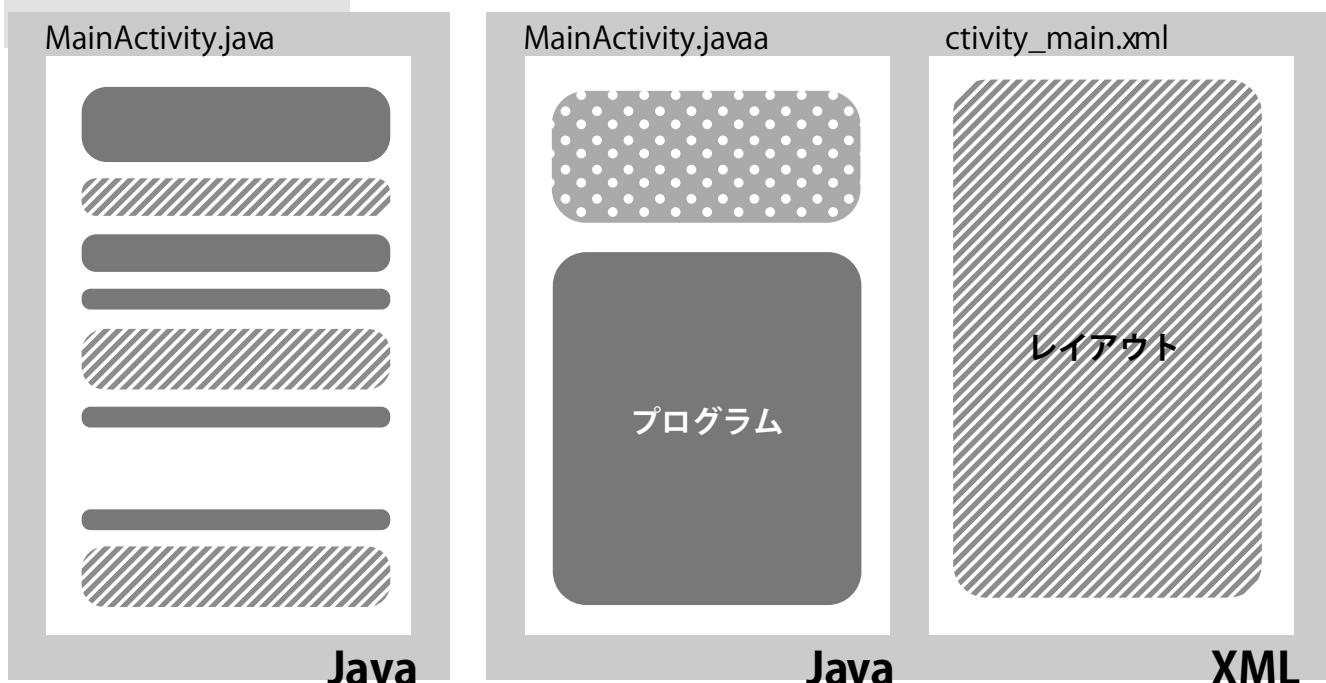
それと同時に、ユーザーインターフェースとして、押すためのボタンももちろん必要になります。このボタンは、先ほどのようにJavaファイル内で生成することができますが、Java内でボタン(や他のパーツ)を作ることはあまりよくないとされています。

その理由は、ソースコードが読みにくくなるからです。

1つのJavaファイルに、プログラムもレイアウトも両方書いていってしまうと、どの部分がプログラムでどの部分がレイアウトか、というのが混在してしまいます。その結果、あとから見たときに、どこに何が書いてあるのかがわかりづらくなってしまいます(図1の左側)。レイアウトの修正が必要になったときにも、ソースコードがプログラムとごちゃまぜになってしまっているため、修正ミスにもつながります。

そのため、各ファイルの任務を決めておくようにし、ソースコードを簡素に、見やすくしようということが推奨されているのです(図1の右側)。

- レイアウトを定義している部分
- プログラムを定義している部分
- レイアウトとプログラムを連携する部分



Javaにプログラムとレイアウトを両方書いてしまうと、それぞれが点在してわかりづらい

プログラムはJava、レイアウトはXMLに分担する
どこに何が書いてあるかがすぐにわかる

図1:左のように、Javaファイル内だけですべてを作ろうとすると、ソースコードが煩雑になるため、右のようにユーザーインターフェースは別途XMLで作るのが望ましい

実務的な観点からいうと、このようにプログラムとレイアウトを完全に分けておくことで、「プログラムをつくる人」「レイアウトをつくる人」を分けることができるため、作業効率の向上にもつながります。

Java内では、あくまでもユーザーの操作に対応するプログラムのみを書いていくということを覚えておいてください。ボタンを生成し、任意の位置に配置することは、XMLが担うのです。

8-3-2 実際に XML でレイアウトを制作してみよう

XMLレイアウトがどんなものかを知る

それではまず、XMLレイアウトがどのようなものかを知るために、あらかじめ配置されているXMLをカスタマイズしていきます。

ADTを起動し、新た「UIBasic2」というAndroidプロジェクトを作成しましょう。

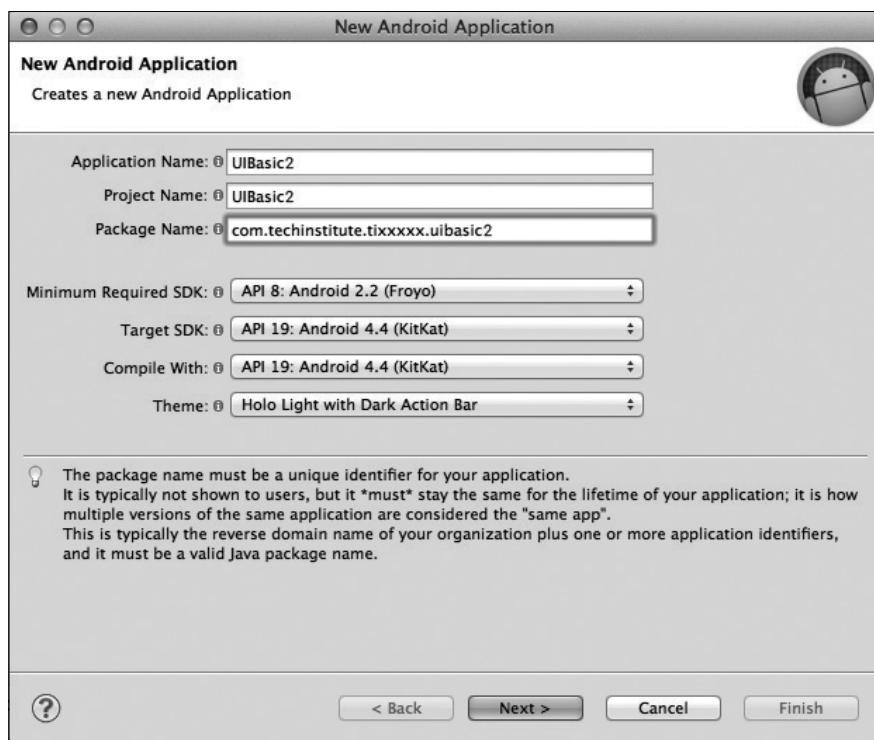


図2:「UIBasic2」という新規プロジェクトを作成する

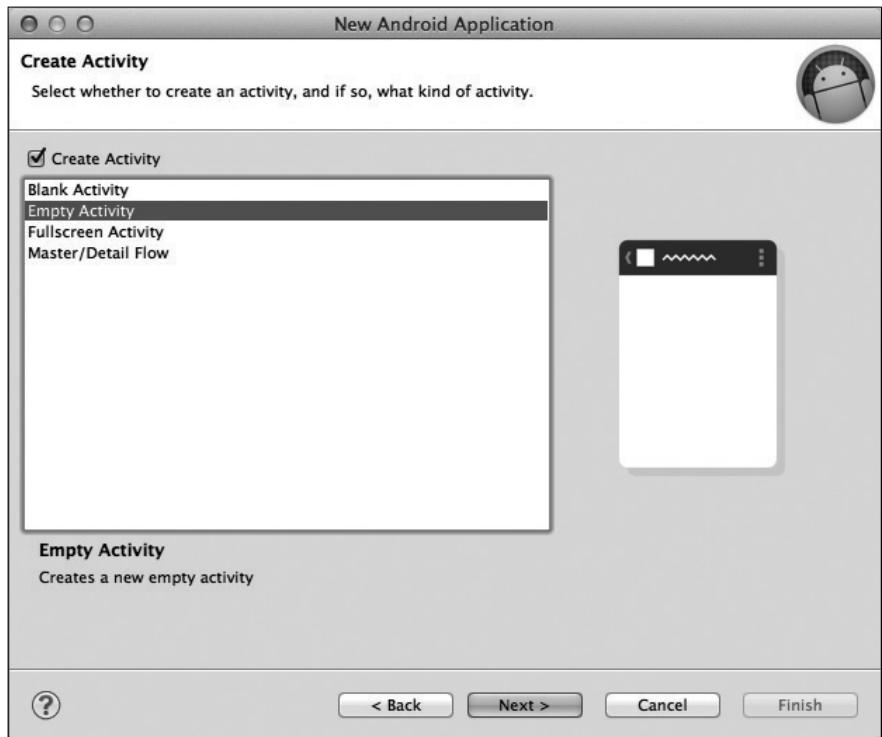


図3:「Empty Activity」を選択する

そうすると、「MainActivity.java」「activity_main.xml」のふたつのファイルが開かれた状態になります。前節と同じように「activity_main.xml」のGraphic Layoutを見ると、「Hello world!」が表示されています(図4)。

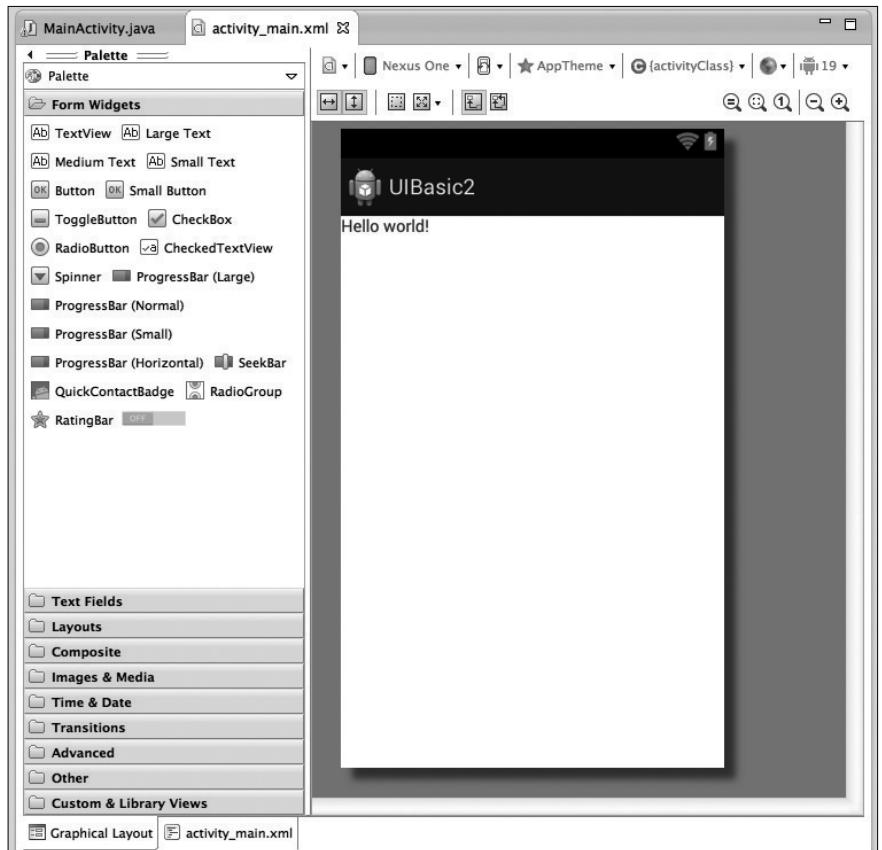


図4:Graphical Layoutを見る

以下のタブを「activity_main.xml」に切り替えてみましょう。ここに書かれているのが、レイアウトを構成するXMLです(図5)。



```

MainActivity.java activity_main.xml
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     tools:context="${packageName}.${activityClass}" >
6
7     <TextView
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:text="@string/hello_world" />
11
12 </RelativeLayout>
13

```

図5:「activity_main.xml」に、レイアウトを構成するXMLが書かれている

XML上のテキストを変更する

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

テキストを画面に表示している部分のXMLは、上のようになっています。

4行目の「`android:text=""`」の中で、表示する文字列を指定しています。ここでは、「`@string`」といって他のファイルを参照するように指定されていますが、いったん無視して、好きなテキストに変えてみます。

```
activity_main.xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="XML レイアウトの基礎" />
```

レイアウトが複雑になってきたり、プログラミングによってレイアウトを表示させたりするものは、Graphical Layoutでは確認できません。そのときはエミュレーターや実機を使って確認するようにしましょう。

確認方法は、エミュレーターでも実機でもどちらでもかまいません。簡易的な確認であれば、Graphical Layoutでもできます(図6)。



先ほど「Hello world!」だったテキストが、入力したもの(ここでは「XMLレイアウトの基礎」)に変わっていることが確認できます。

テキストの色や大きさを変更する

では次に、テキストの色や大きさを変更してみましょう。先ほどのソースコードに、次の2行を追加します。

```
activity_main.xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="XML レイアウトの基礎"
    android:textColor="#ff0000"
    android:textSize="22sp" />
```

XMLでレイアウトを作るとときには、必ず「このコードを書くとどうなるかな」というのを想像するようにしましょう。コードから、実際の表示がどうなるかというのを、自分の中でイメージできることが大切なのです

「sp」(エスピー)は、文字の大きさを示す単位です。また、領域のサイズを示すときには「dp」(ディーピー)という単位を使います。今は「22sp」がどれくらいの大きさなのかちゃんとわからなくても、「通常より大きい…かな?」という程度でかまいません。何度も試していくうちに感覚を覚えていきます。Androidでは、さまざまな画面サイズや画面密度の端末が存在するため、ウェブなどでよく使う「px」(ピクセル)という単位はほとんど使いません。

追加したコードのうち、「android:textColor」がテキストの色を指定する属性で、ここでは「#ff0000(赤色)」を指定しました。また「android:textSize」はテキストの大きさを指定する属性で、「22sp」を指定しています。これらを追加することによって、どのような表示になるかはだいたい想像が付きますよね(図7)。

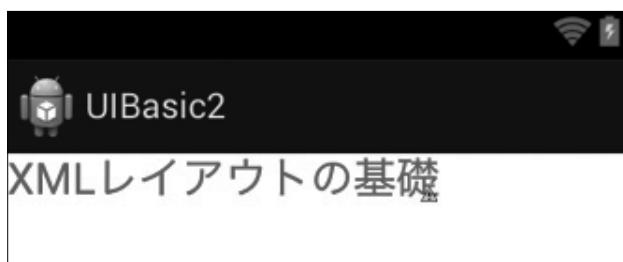


図7:出力結果。テキストが赤色になり、サイズが大きくなる

図7のように、テキストが赤色になり、フォントのサイズが大きくなったことが確認できます。このように、さまざまな属性を指定していくことで、Androidのレイアウトを作成していくのです。

色の指定方法

ここで、色の指定方法を補足しておきます。

先ほど、「#ff0000」というカラーコードを使いました。このカラーコードは16進数で構成されていて、HTMLやCSSなどで使うものと同じです。

パソコンはもちろん、テレビやスマートフォンなども含めて、液晶モニタを使って出力する色はRGBというカラー・モデルを使います。Rは赤(Red)、Gは緑(Green)、Bは青(Blue)を示しており、それぞれの色をどれぐらいずつ混ぜるかによって色を決める方法です

それぞれの値は、10進数で表すと「0～255」を使います。Illustratorなどのグラフィックツールで色を指定する場合は、これらの10進数で指定することができます。0に近付くほど黒くなり、255に近付くほど白くなりますので、「R:0、G:0、B:0」は黒、「R:255、G:255、B:255」は白を表します。



図8:黒色をRGBで表現



図9:白色をRGBで表現

これらは、光の色と考えてもかまいません。真っ黒のところに赤色の光を入れると、もちろん赤色が映ります。また、赤色と緑色の光を混ぜると、黄色になります。

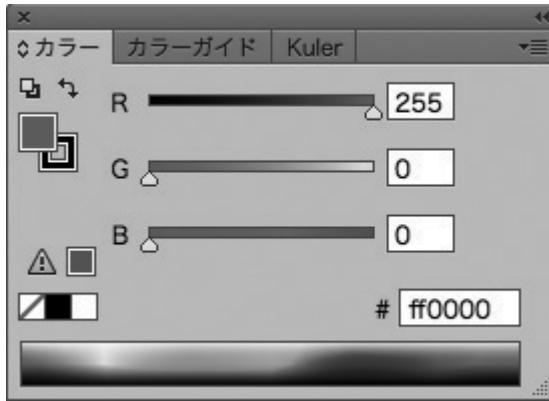


図10:赤色をRGBで表現

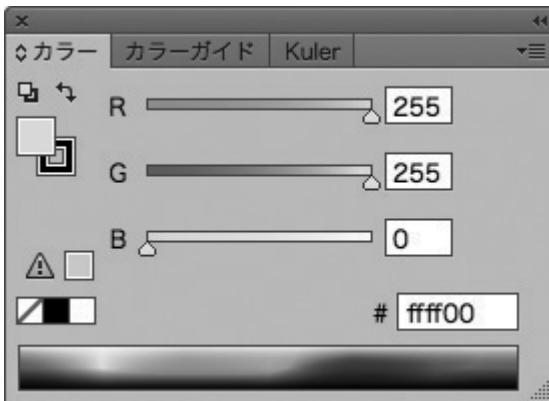


図11:黄色をRGBで表現

この10進数の値を16進数に変換し、R、G、Bの順に並べたものがカラーコードとなります。カラーコードは、図8～11の各図の右下に6桁で表記されています。

慣れてくると、カラーコードを聞くだけでだいたいどんな色なのかがわかるようになりますが、慣れるまでは、Webサイト(例：<http://www.colordic.org/>)などでカラーコードを調べて、自分の好きな色を選ぶのもよいでしょう。

Androidの場合、このRGBを表す6桁の色コードの前に、さらに2桁追加し、8桁で色を表すこともあります。RGBの前の2桁は「アルファチャンネル」(不透明度)を意味します。ただし、アルファも16進数になっていることに注意してください。

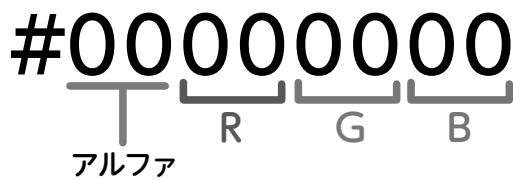


図12:アルファも入った場合のカラーコード

「#00000000」の場合、アルファが0%ですので、これは透明となって見えません。「#FF000000」の場合、アルファが100%ですので、普通の黒となります。アルファを省略する場合は、「FF」が省略されていることになります。



8-3-3 実際の XML の書き方

ユーザーインターフェースを記述するXMLは、次のソースコードのような要素で構成されています。

テキストを表示させる XML

```
<TextView> — ビュー名 属性 値  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="XML レイアウトの基礎"  
    android:textColor="#ff0000"  
    android:textSize="22sp" />  
                                         |  
                                         カッコを閉じる前に「/」が必要
```

- ・ビューネ名：何を表示させるか
- ・属性：ビューに対する詳細な設定。ほとんどの場合、先頭に「android:」の接頭辞が付く
- ・値：属性の設定値

以下、それぞれの要素について、詳しく見ていきましょう。

ビュー

Androidでは、各パーツのことを「ビュー(View)」と呼びます。よく使うビューは、次のようなものがあります。

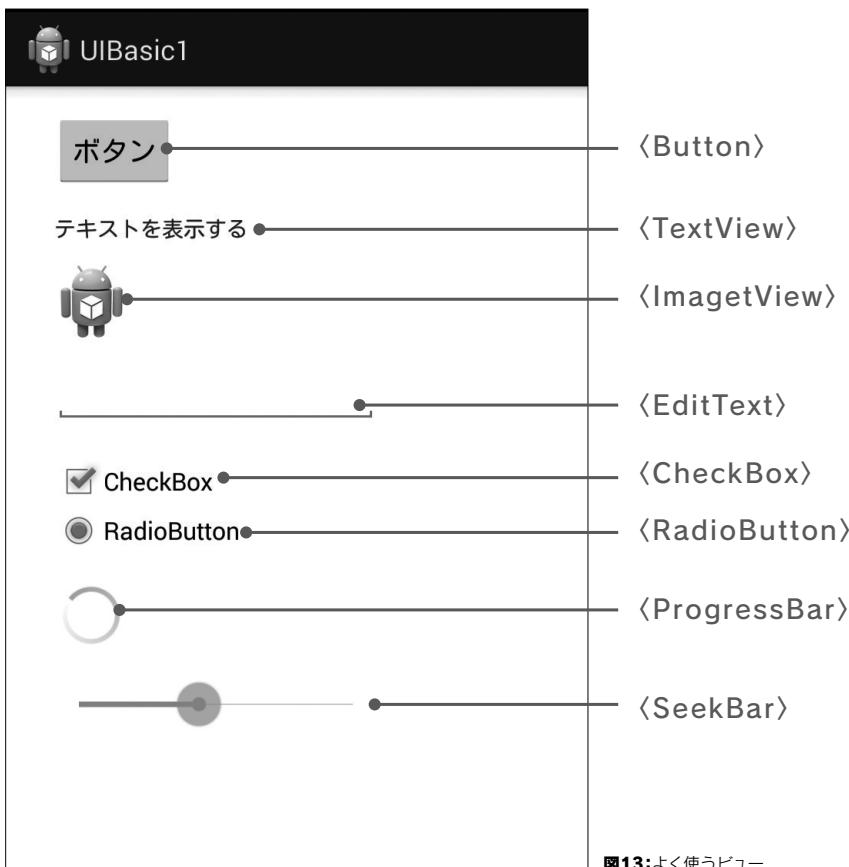


図13:よく使うビュー

ビュー名	説明
Button	ボタンを表示する
TextView	テキストを表示する
ImageView	画像を表示する
EditText	テキストフィールド(文字入力エリア)を表示する
CheckBox	チェックボックスを表示する
RadioButton	ラジオボタンを表示する
ProgressBar	プログレスバーを表示する
SeekBar	シークバーを表示する

図1:よく使うビュー

ButtonやTextViewでは、文字列を変更したり、文字の大きさや色を変えてカスタマイズすることは簡単です。また、ImageViewについても、画像リソースを変更することいろいろな画像を表示させることができます。

しかし、EditTextやCheckBox、RadioButton、SeekBarなどは、あらかじめ青色がデザインとして入っています。これらのデザインは、「コードで実装できるデフォルトのデザイン」と理解しておいてください。もちろん違うデザインに変更する(カスタマイズする)こともできますが、少しややこしくなりますのでもう少し慣れてきてから挑戦するのがいいでしょう。

属性

属性は、各ビューによって設定できるものが決まっています。

たとえば、テキストを表示するTextViewであれば、色(textColor)や大きさ(textSize)の設定ができるが、画像を表示するImageViewだと、それらの設定はありません。

各ビューに設定できる属性は、それぞれにおいてかなりの数があります。どのような属性が指定できるかは、コードを書いている際に「android:」と入力すると、オートコンプリート機能によって指定可能な候補一覧が表示されますので、それを見ておくとよいでしょう(図14)。

```

1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   tools:context="${packageName}.${activityClass}" >
6
7   <TextView
8     android:layout_width="wrap_content"
9     android:layout_height="wrap_content"
10    android:text="XMLレイアウトの基礎"
11    android:textColor="#ff0000"
12    android:textSize="22sp"
13    android: />
14
15  </RelativeLayout>
16

```

図14:「android:」まで入力すると、TextViewで指定可能な属性の候補が表示される

値

値は、各属性によって設定できるものが決まっています。

これも、設定できるものが選べる場合は、属性と同じように候補一覧が表示されますので、見ておくとよいでしょう(図15)。

数値を指定するものは、候補には出できません。自分で値を入力しないといけないので注意してください。

```

1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   tools:context="${packageName}.${activityClass}" >
6
7   <TextView
8     android:layout_width="wrap_content"
9     android:layout_height="wrap_content"
10    android:text="XMLレイアウトの基礎"
11    android:textColor="#ff0000"
12    android:textSize="22sp"
13    android:textStyle="normal" />
14
15  </RelativeLayout>
16

```

図15:TextViewのtextStyleという属性で、指定可能な値の候補が表示される

主要なパートについての説明

それでは、よく使うパートを詳しく説明していきます。これらのパートは、何も見なくてもXMLで作成できるようにしておくことが望ましいです。

- ・ボタン
- ・テキスト
- ・画像
- ・テキストフィールド

とはいっても、すべてを覚える必要はありません。ADTではオートコンプリート機能により候補一覧が表示されますので、この機能はヒントとして積極的に使っていきましょう。

■ボタン

ボタンを表示するときには、次のようなXMLを書きます。

```
Button コンポーネント
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```

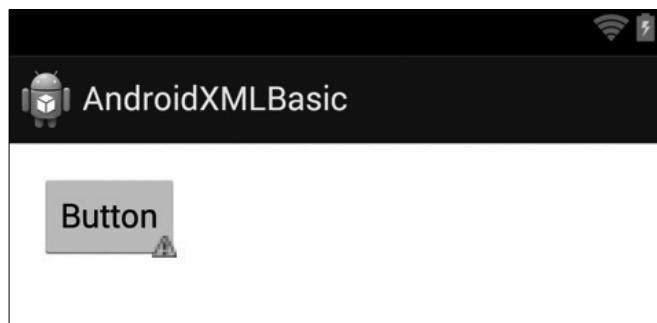


図16:ボタン表示

デザインのカスタマイズを何もしていない、デフォルトの状態のこのボタンは、タップすると色が変わり、「タップされた」という状態をユーザーにフィードバックします。



通常時



指でタップ時

図17:Android4.3まで
は、デフォルトではタップ
すると青くなる



通常時



指でタップ時

図18:Android4.4では少
し色が濃くなる

前述(P.41)のソースコード「テキストを表示させるXML」の4行目の、「`android:text`」という属性が、ボタン内に表示するテキストです。

2行目の「`android:layout_width`」と3行目の「`android:layout_height`」は、ボタンだけではなくどのビューにも共通して必要な属性です。それぞれのビューが、横方向(`layout_width`)または縦方向(`layout_height`)に対してどれくらいの領域を占めるかという設定です。

これらには「`wrap_content`」と「`match_parent`」という2つの値が存在します。

値	説明
<code>wrap_content</code>	ビューを占める領域は、そのビューがもつているサイズのみにとどまる
<code>match_parent</code>	ビューを占める領域は、縦または横の画面サイズいっぱいまで広がる

表2:ビューのレイアウト指定

説明を見るだけでは理解が追いつかないかもしれません。実際にどうなるか設定して確認してみましょう。

■■wrap_content

`wrap_content`を設定した場合、ビューを占める領域は、そのビューが持っているサイズのみにとどまります。

つまり、「Button」という文字が設定されているボタンであれば、「Button」という文字のまわりに一定の余白がとられ、それらをひっくるめたものがビューの領域となります(図19)。

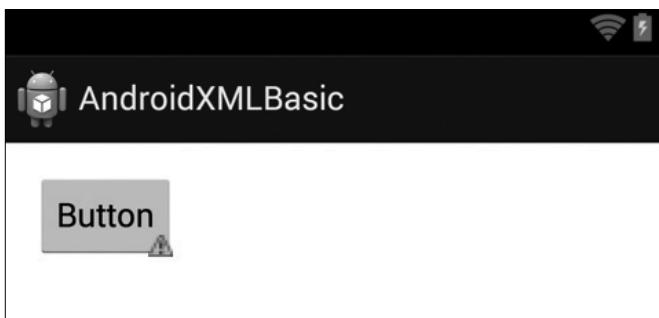


図19:ビューを幅としたボタン

確認のため、ボタン内に表示するテキストを変更してみます。

ボタンに表示する文字列

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録して送信する" />
```

4行目の文字列を変更すると、ボタン全体の横幅がテキストの文字数にあわせて伸びました(図20)。



図20:ボタンの文言を変更すると、それにあわせてボタンの横幅が伸びる

このように、ビューが持っているサイズにあわせたいときは、「wrap_content」を設定します。

■■match_parent

前述のソースコード「テキストを表示させるXML」では、`android:layout_width`も`android:layout_height`も、両方ともに「wrap_content」が設定されていました。では、以下のソースコードのように2行目の`android:layout_width`を「match_parent」に変更してみます。

android:layout_width を「match_parent」に変更

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Button" />
```

横方向に、画面サイズいっぱいまで広げる設定をしましたので、ボタンは図21のように横幅いっぱいまで広がります。

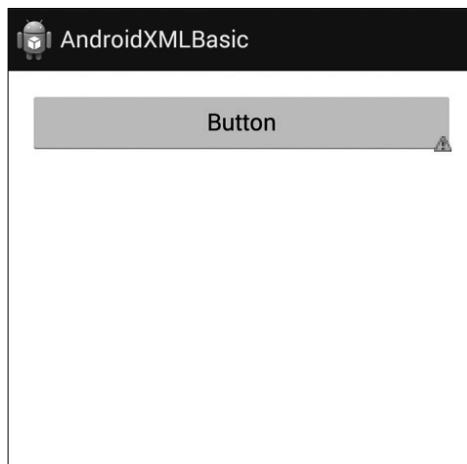


図21:ボタンが横幅いっぱいまで広がる

では今度は、`android:layout_width`を「wrap_content」に戻し、`android:layout_height`を「match_parent」に設定してみましょう。

android:layout_height を「match_parent」に変更

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"  
    android:text="Button" />
```

今度は縦方向に、画面サイズいっぱいまで広がりました(図22)。

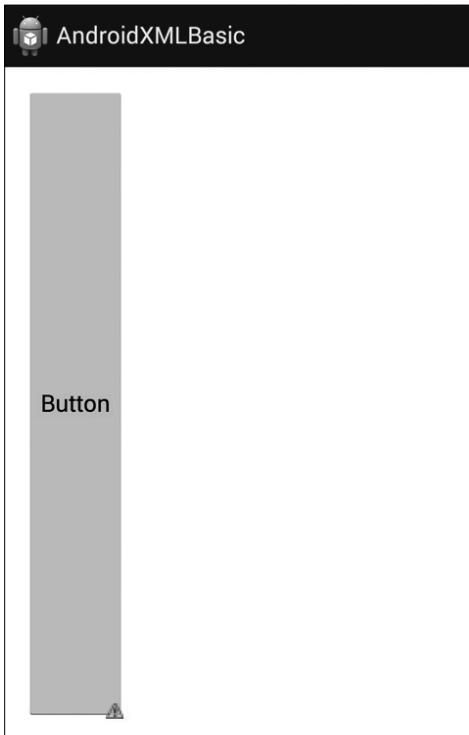


図22:ボタンが縦方向に画面サイズいっぱいまで広がる

実際のアプリケーションでは、ボタンをこのように縦方向の画面サイズいっぱいまで広げるようなレイアウトはほとんどありません。ですが、他のビューでは必要になることもありますので、覚えておきましょう。

■画像

画像を表示するときには、次のようなXMLを書きます。

ImageView コンポーネント

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher" />
```

上のコードの4行目の「`android:src`」は、resフォルダーの「`drawable-xxxxx`」フォルダー内に入っている画像リソースのファイル名を設定する属性です。画像のフォーマットは、PNGまたはJPEGが有効です。

`ic_launcher.png`は、resフォルダーの「`drawable-mdpi`」「`drawable-hdpi`」「`drawable-xhdpi`」という3つのフォルダーの中に1つずつ入っています。内容は同じですが、それぞれの画面解像度に合ったサイズのものが入っています。

プロジェクトにあらかじめ準備されている、ランチャーアイコンの画像。同じ名前、同じ大きさで上書きすれば、ランチャーアイコンを変更できる。

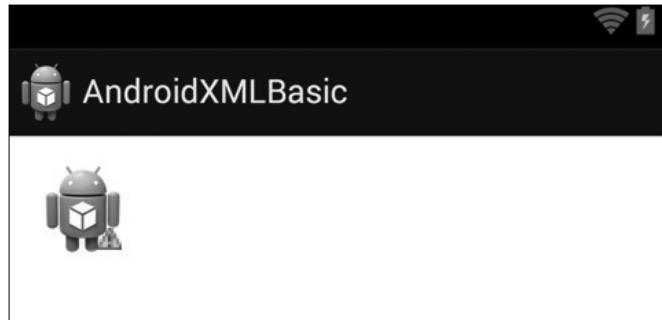


図23:drawable-xhdpiフォルダーにある「ic_launcher.png」を表示

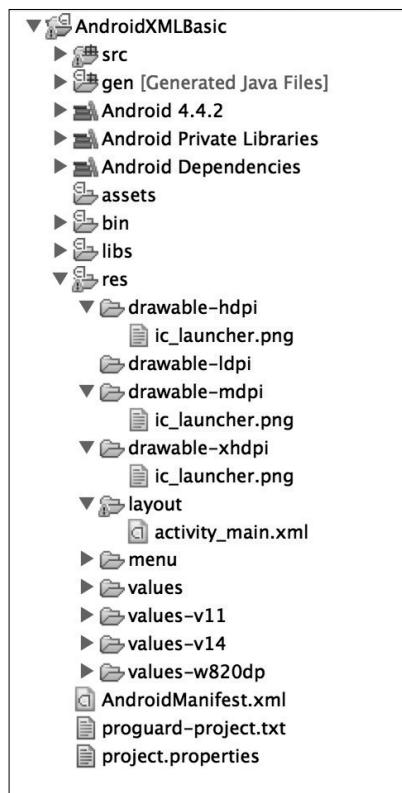


図24:resフォルダー内のdrawableフォルダー



図25:各drawableフォルダーには、それぞれの画面密度にあつた大きさの画像リソースが入っている

「`android:src`」の指定では、「`@drawable`」をつけて、`drawable`というフォルダ内を参照します。しかし、ここでは「`@drawable-xhdpi/ic_launcher`」のように画面密度は書きません。

参照フォルダに画面密度は入れない

`android:src="@drawable/ic_launcher"`

なぜなら、どの画面密度なのは端末ごとに決まっているので、Androidアプリで利用する画面密度は自動で判別されるからです。たとえば、「Galaxy S3」の端末の解像度はxhdpiに属します。このときに画像を参照するフォルダーは「drawable-xhdpi」ですが、コードで書くのは「@drawable/ic_launcher」でよいのです。

逆に、画面密度を指定してしまうと、他の画面密度の違ういろいろな端末から見たときに、表示がおかしくなってしまう恐れがあるので注意しましょう。

画像の大きさについては、たとえばランチャーアイコンやアクションバーのアイコンの大きさなど、Android全体でルールが決められているものはそれに従いましょう。それ以外の自分で作るレイアウトについては、自分たち自身で決めていかなければなりません。

Android全体のルールは、ガイドラインの「Iconography」の項に書かれているので、ぜひ見ておきましょう。

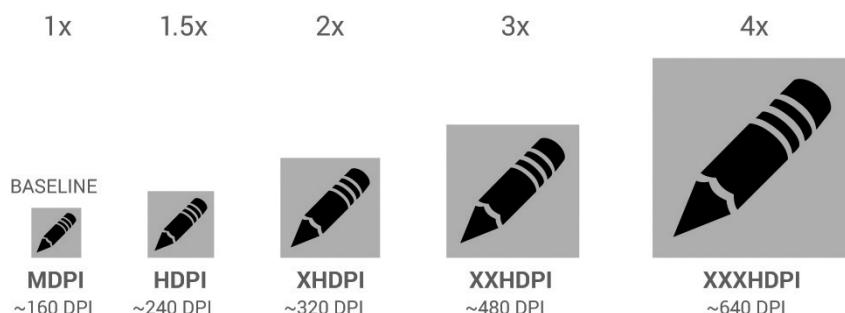


図26:各画面密度に最適化した大きさの画像が必要になる

■テキストフィールド

入力エリアを表示するときには、次のようなXMLを書きます。

```
EditText コンポーネント
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

これだけで、最低限の入力エリアを作成することができます。

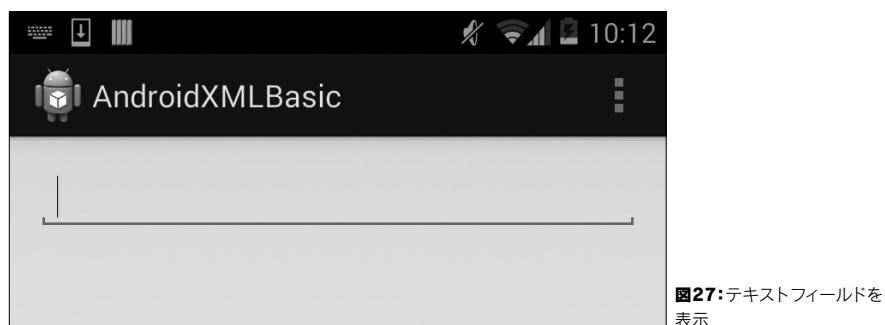


図27:テキストフィールドを表示

前ページのコードでは、「`android:layout_width="match_parent"`」となっているので、入力エリアが横幅いっぱいに配置されます。

一方、文字数によって横幅を指定したいときは、次のコードのように`android:layout_width`を「`wrap_content`」にして、「`android:ems`」を追加します。「`android:ems`」には、文字にして何文字分か? という数値を設定します。

文字数による横幅指定

```
<EditText  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:ems="10" />
```

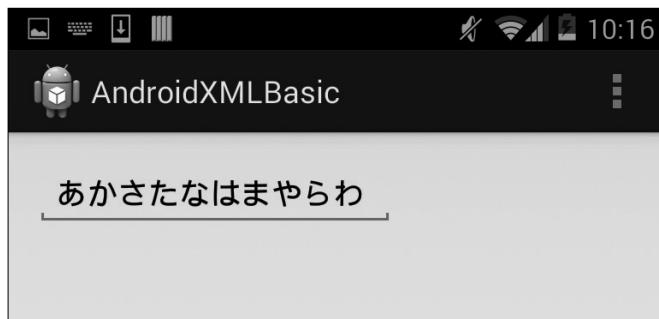


図28: 文字数による横幅指定

図28の場合は10文字分の横幅になります。

しかし、実はこれだけでは実用には不十分です。10文字以上文字を入力すると、このままでは改行されてしまうからです(図29)。

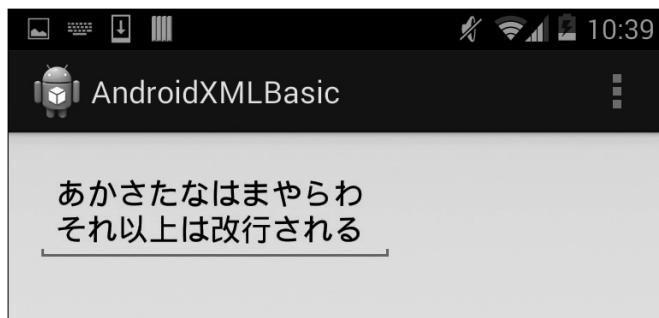


図29: 10文字以上入力すると、改行されてしまう

複数行入力させるような内容の場合ではこれでもよいのですが、通常の入力フォームでは、名前やメールアドレス、パスワードなどを入力することが多いので、改行してしまうのはあまり使い勝手がよくありません。

こういったときには、入力制限を設定する必要で、改行できないようにする場合は、「`android:inputType="text"`」を追加します。

EditText コンポーネント

```
<EditText  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:ems="10"  
    android:inputType="text" />
```

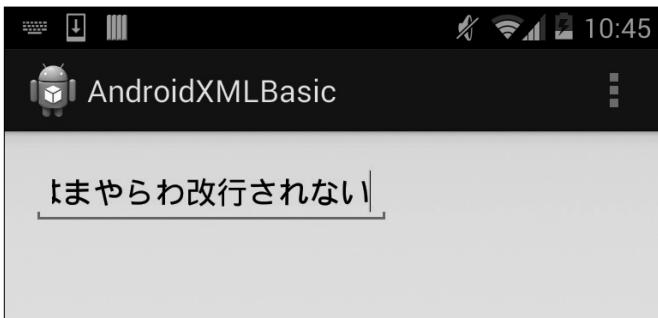


図30:10文字以上入力しても、改行されずに左に押し込まれるようになった

「`android:inputType`」という属性は、その他にもいろいろな入力制限を設定することができます。

たとえば、「`android:inputType="textPassword"`」を追加すると、前章にも出てきたパスワード入力用のテキストフィールドになります。

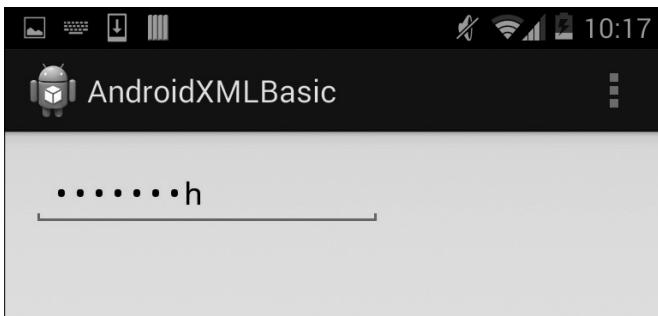


図31:パスワード用の入力エリアになる

「`android:inputType`」については、場合に応じて必要な入力制限を設定するようにしましょう。表3に、よく使われるものを挙げておきます。すべての値は、ガイドラインのTextView > `android:inputType`の項を参照してください。

値	説明
<code>None</code>	入力不可
<code>Text</code>	文字を入力
<code>textAutoCorrect</code>	文字のスペルミスを自動で修正する
<code>textAutoComplete</code>	文字の補完入力をする
<code>textMultiLine</code>	文字を複数行入力する
<code>textUri</code>	URLを入力する
<code>textEmailAddress</code>	メールアドレスを入力する
<code>textPassword</code>	パスワード入力する
<code>textVisiblePassword</code>	パスワードを隠さずに入力する
<code>number</code>	数値を入力する
<code>phone</code>	電話番号を入力する

表3:`inputType`でよく使う値

ユーザーにわかりやすい入力フォーム

これらの入力フォームをいちばんよく使うのは、アプリを開いて、サービスにログインする部分でしょう。メールアドレスとパスワードを入力させることが圧倒的に多いです。以前は、図32のように、「Eメール」や「パスワード」というTextViewを置き、その横にEditTextで入力フォームを並べていることがほとんどでした。

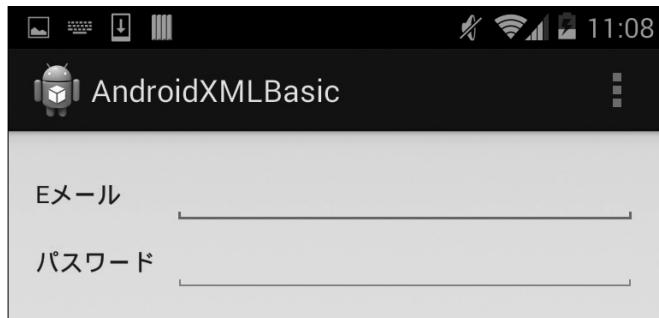


図32: TextViewとEditTextが横並びで配置

しかし、最近では図33のように、入力エリアの中に、うすい文字で「Eメール」や「パスワード」説明を書いておくことが多くなりました。

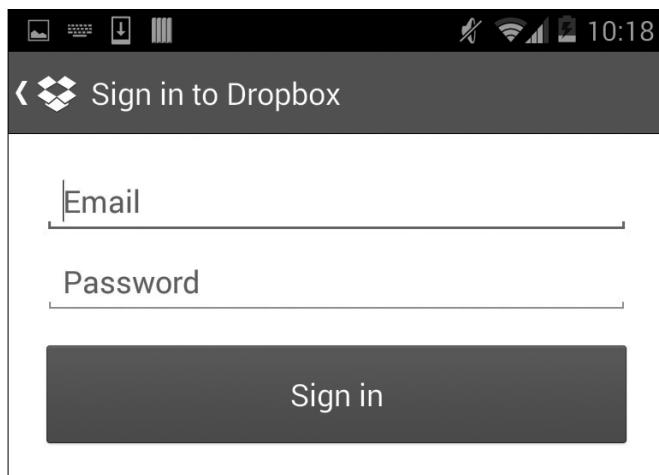


図33: Dropboxアプリのログイン画面

HTMLでもテキストフィールドに説明テキストを入れることができます。HTMLの場合は、placeholderという属性になります。<input type="text" placeholder="Eメールアドレスを入力">

android:hintを利用する

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Eメールアドレスを入力"  
    android:inputType="textEmailAddress" />
```

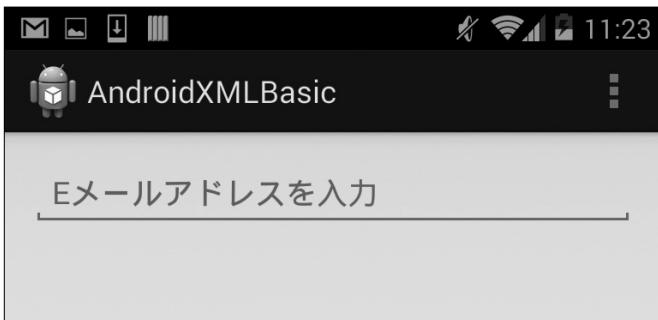


図34: android:hintを利用する

図33や図34のように、入力エリア内に説明のテキストラベルを入れてもいいのですが、ユーザーの操作性からすると、少し問題もあります。入力前の状態を比べてもあまり差がわからないかもしれませんので、入力後の状態を見てみましょう。

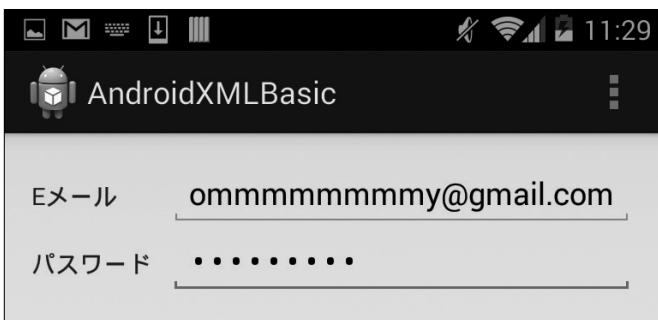


図35: テキストラベルが入力エリアの外にある場合

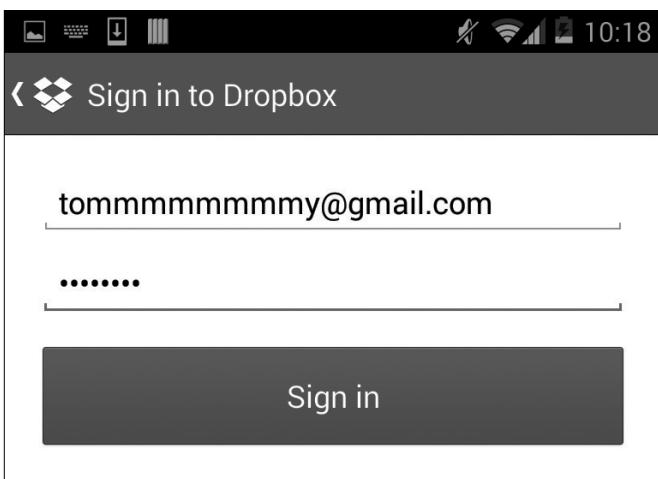


図36: テキストラベルが入力エリアの中にある場合、入力するとラベルが消える

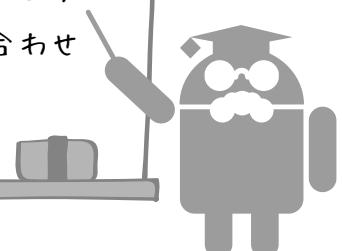
図36のほうは、説明のテキストラベルが消えてしまいました。これは、今回のようなEメールとパスワードしかないような場合は問題ないのですが、入力項目が多くなった場合、あとで見返したときに何の項目を入力していたかがわからなくなるので、ユーザーにとってはあまり便利ではないものになってしまいます。

単なるテキストフィールドとはいえ、ユーザーが実際に入力をしていくパートです。やみくもにEditTextを置いていくだけではなく、inputTypeをそれぞれについて最適なものを設定したりして、操作性と認識性には常に心がけるようにしましょう。

8-4 UIの基礎知識（4）

著：秋葉ちひろ

ここまでこの節で、XMLでだいたいのパーツがつくれるようになったと思います。ですが、それはまだ1つのパーツを学んだだけにすぎません。ここでは複数のパーツを組み合わせて、画面のレイアウトを作成していく方法を学びます。



この節で学ぶこと

- 1 RelativeLayoutとLinearLayout
- 2 Gravityについて
- 3 Androidアプリのレイアウトを作成できるようになる

この節で出てくるキーワード一覧

RelativeLayout
LinearLayout
android:layout_weight
android:orientation
android:layout_above
android:layout_below
android:layout_centerHorizontal
android:layout_centerVertical
android:layout_alignParentLeft
android:layout_alignParentRight
android:layout_alignParentTop
android:layout_alignParentBottom

8-4-1 レイアウトを作成する

Androidにおいて、XMLでレイアウトを作成し、パーツを配置していくためには、主に次の2つの方法があります。

レイアウト名	説明
RelativeLayout	相対レイアウト。それぞれのパーツにおいて基準となるビューを決めながら配置していく。
LinearLayout	線形レイアウト。画面の左上を中心にして、ビューを右方向または下方向へ順番に配置していく。

表1: パーツを配置していくための方法

これら2つの方法を、適宜組み合わせながら配置していきます。以下で、詳しく見ていきましょう。

RelativeLayout

「RelativeLayout」は、相対レイアウトともいわれます。それぞれのパーツを配置していくときに、基準となるビューを決めながら配置をしていきます。それでは実際に作成しながら進めていきましょう。

ADTを起動し、新たにAndroidプロジェクトを作成します。

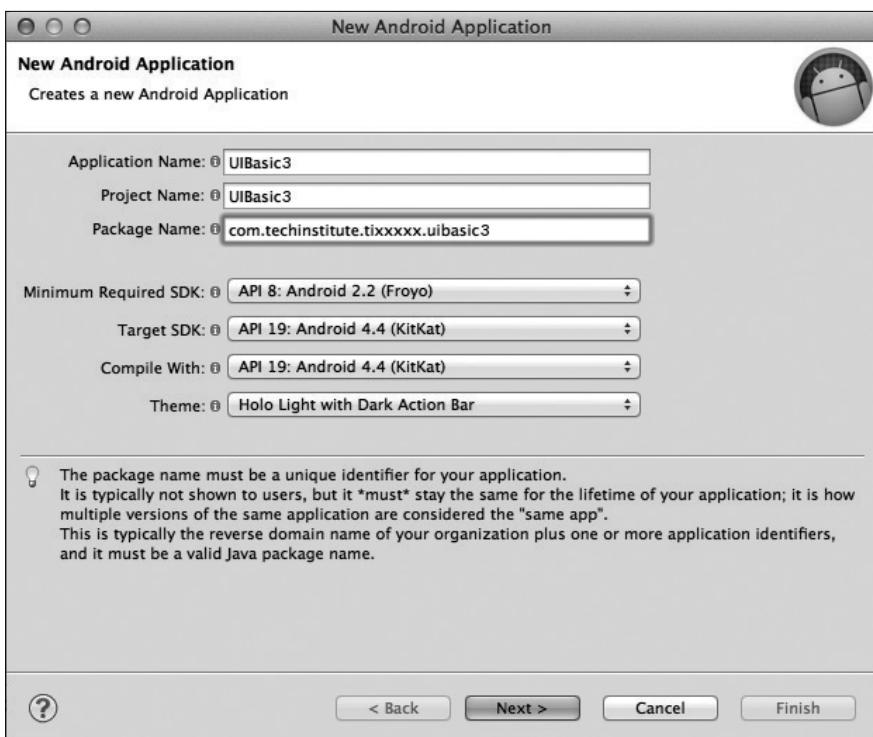


図1: UIBasic3という新規プロジェクトを作成する

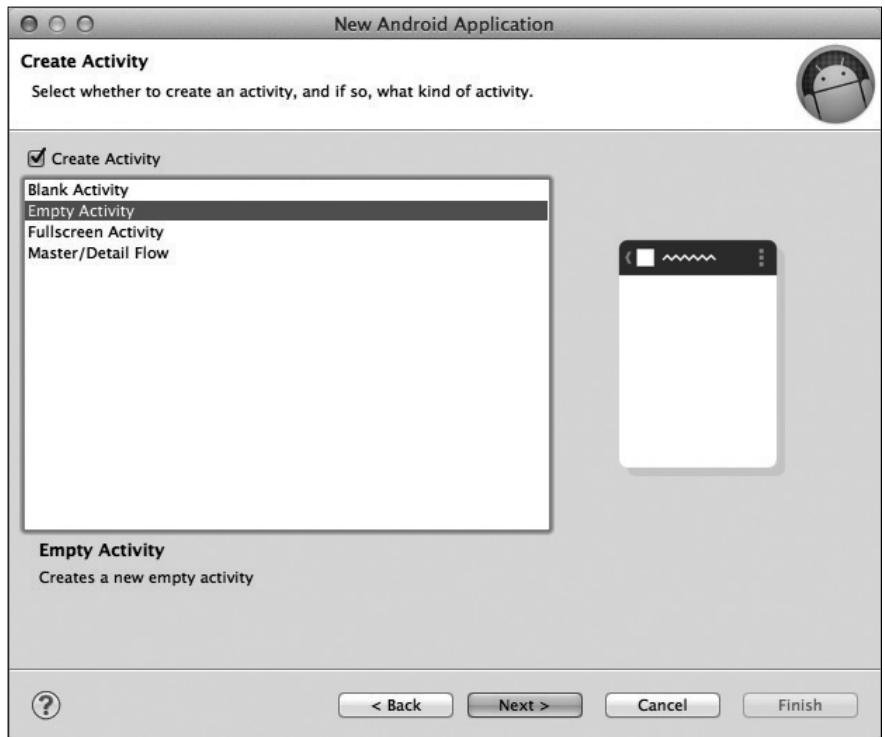


図2:Empty Activityを選択する

新規に作成したAndroidプロジェクトのXMLレイアウトは「activity_main.xml」ですが、ここにはあらかじめRelativeLayoutが記述されていますので、このまま使っていきましょう(図3)。

The screenshot shows the Android Studio code editor with two tabs: 'MainActivity.java' and 'activity_main.xml'. The 'activity_main.xml' tab is active, displaying the following XML code:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${packageName}.${activityClass}" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

図3:activity_main.xml



8-4-2 親ビューを基準としたレイアウトの例

RelativeLayoutは相対レイアウトですから、基準となるビューが必要です。まずは、基準を親ビューにすることを考えます。

図3にある、すでに書かれているRelativeLayoutを使い、その中に「ImageView」を入れます。この場合の確認は、エミュレーターや実機でなくても、Graphical Layoutでもかまいません。

`activity_main.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${packageName}.${activityClass}" >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />
</RelativeLayout>
```

1~5行目と、10行目は、あらかじめ書かれているコードです。中に書かれていた TextViewを一旦削除し、ImageViewを追加しましょう(図4)。

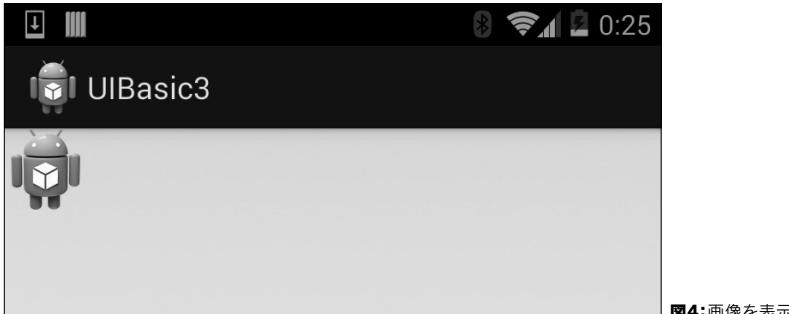


図4:画像を表示

親ビューの中央に配置する

次に、図5のように画像を中央に配置するために、activity_main.xmlのImageViewを次のようにします。

`activity_main.xml`

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:layout_centerInParent="true" />
```

ソースコードの5行目に、「`android:layout_centerInParent`」という属性を追加しました。値は「`true`」です。これは、「親ビューに対して中央に配置する」という

`layout_centerInParent`のデフォルト値は`false`ですので、通常は`true`を指定する場合のみ書きます。

指示になります。

図5のように、ドロイドくんが中央に配置されたら成功です。

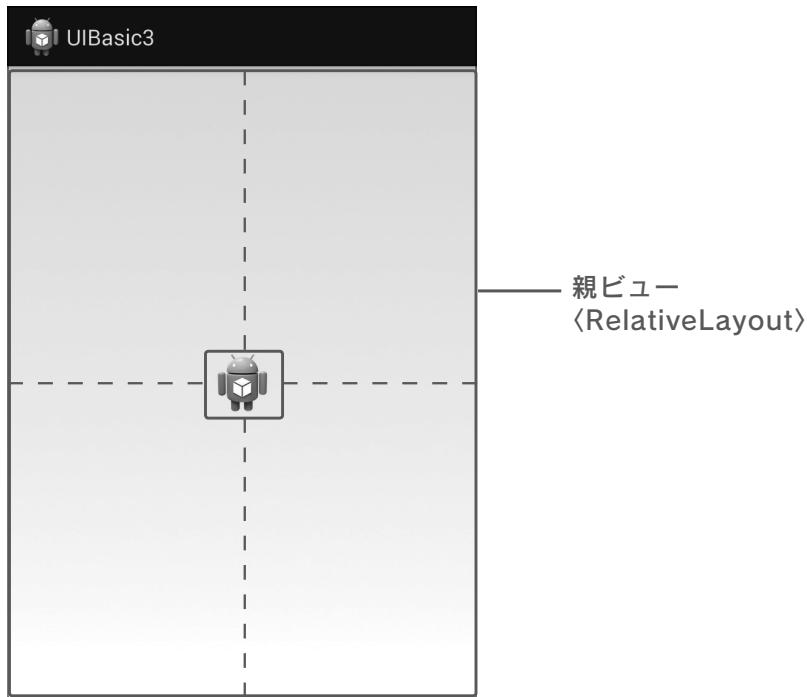


図5: ドロイドくんを配置。親ビューに対して中央に配置されている

親ビューの端にそろえて配置する

次に、中央だけではなく、親ビューの端にそろえてドロイドくんを配置していきます。

具体的には図6のように、上端、下端、左端、右端、またそれらの中央も含めて配置する方法です。

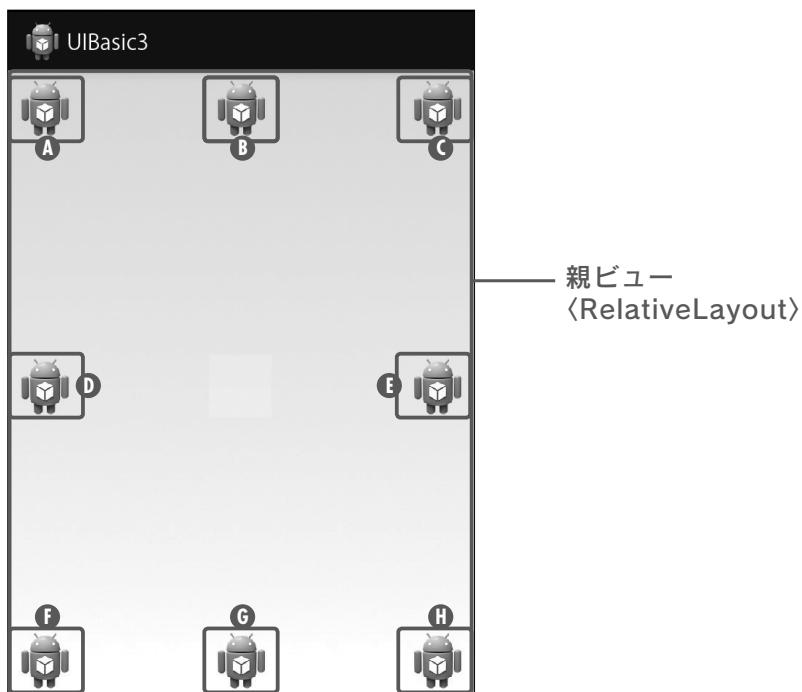


図6: 親ビューの端にそろえてさまざまな場所に配置

前述のソースコードで5行目に追加した「`android:layout_centerInParent=true`」を、次の表2の属性を参考にしながら書き換えていきます。

属性	説明
<code>android:layout_centerInParent</code>	親ビューの中央に配置する
<code>android:layout_centerHorizontal</code>	親ビューの水平方向中央に配置する
<code>android:layout_centerVertical</code>	親ビューの垂直方向中央に配置する
<code>android:layout_alignParentLeft</code>	親ビューの左端にそろえて配置する
<code>android:layout_alignParentRight</code>	親ビューの右端にそろえて配置する
<code>android:layout_alignParentTop</code>	親ビューの上側にそろえて配置する
<code>android:layout_alignParentBottom</code>	親ビューの下側にそろえて配置する

表2:親ビューに対する配置を決める属性一覧

図6のA～Hの場合を実際に作ってみて確認してください。

8-4-3 独自の情報収集

親ビューを基準にして、いろいろな場所に配置してみましたが、今度は親ビューではなく任意のビューを基準にして配置してみます。

図7のような配置を考えてみましょう。画像(ImageView)の下にテキスト(Text View)があるような配置です。



図7:画像の下にテキストがある配置

先ほどと同じように、すでに書かれているRelativeLayoutを使って、その中にImageViewとTextViewを入れて確認します。

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context="${packageName}.${activityClass}">  
    <ImageView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:src="@drawable/ic_launcher" />  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="このイラストはドロイドくんです" />  
</RelativeLayout>
```

ただし、このソースコードのとおりに記述すると、図8のように画像とテキストが重なってしまいます。

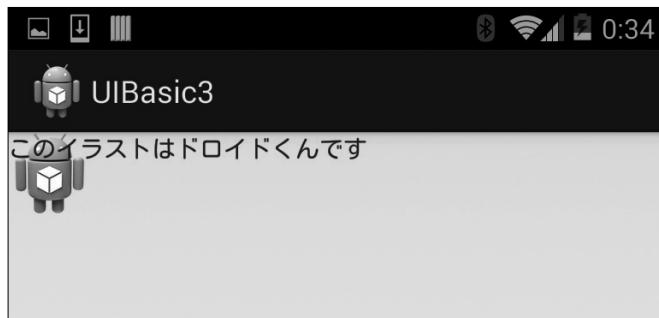


図8:画像とテキストが重つてしまふ

RelativeLayoutで囲んだものは、配置に関する設定をしなければ、中のビューは常に同じ場所に配置され、重なって表示されてしまうという特性があるということを覚えておきましょう。

RelativeLayoutにおいて今回のような配置にする場合は、「画像を基準として、その下にテキストを配置する」という考え方をします。

基準とするものにid名をつける

基準とするものについては、目印として名前をつけます。activity_main.xmlの ImageViewを次のように書き換えます。

activity_main.xml

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher"  
    android:id="@+id/droid" />
```

ソースコードの5行目に、「`android:id`」という属性を追加しました。値は、「`@+id/droid`」と書きましたので、つまりid名は「`droid`」になります。

これで、このImageViewに「droid」という名前をつけた、ということになります。

基準となるものに対して、どこに配置するかを指定する

引き続き、画像の下にテキストを配置しましょう。これは表示するテキストに対して、「droid」という名前のついたビューの下にテキストを配置するという指示をします。先ほどのコードのTextViewを、次のように変更します。

activity_main.xml

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="このイラストはドロイドくんです"  
    android:layout_below="@+id/droid" />
```

「@」と「id」の間に「+」があるかどうかに注意してください。「+」がある場合は「id名をつける」という意味になりますし、ない場合は「id名を参照する」という意味になります

5行目に、「android:layout_below」という属性を追加しました。値は、「@id/droid」と書き、「droid」というid名を参照するものになります。

「layout_below」は、基準となるパーツのどこに配置するかを指示するもので、この場合は「下に配置する」という意味になります。このような指定をすることで、図7のようなレイアウトを作成することができます。

ここで付けたidは、プログラムから参照するときにも使います

ドロイドくんを中央に置き、そのまわりにテキストを配置

次は、ドロイドくんの画像を画面の中央に置き、その周りにテキストを配置してみます。前述したImageViewについて、画面の中央に配置するように、6行目に「android:layout_centerInParent="true"」を追加します。

activity_main.xml

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher"  
    android:id="@+id/droid"  
    android:layout_centerInParent="true" />
```

また、TextViewにテキストも追加しましょう。

```

activity_main.xml
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:id="@+id/droid"
    android:layout_centerInParent="true" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="このイラストはドロイドくんです"
    android:layout_below="@+id/droid" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="上側に配置します" />

```

ソースコードの11行目の「`android:layout_below="@+id/droid"`」は、「`droid`」というid名がついたビューの下に配置するという意味です。それ以外にも、基準となるビューに対してどこに配置するかを指定する属性があります。

属性	説明
<code>android:layout_above</code>	上側に配置する
<code>android:layout_below</code>	下側に配置する
<code>android:layout_toLeftOf</code>	左側に配置する
<code>android:layout_toRightOf</code>	右側に配置する
<code>android:layout_alignLeft</code>	左端をそろえて配置する
<code>android:layout_alignRight</code>	右端をそろえて配置する
<code>android:layout_alignTop</code>	上端をそろえて配置する
<code>android:layout_alignBottom</code>	下端をそろえて配置する

表2:基準となるビューに対してどこに配置するかを決める属性一覧

それでは、先ほど追加した2つめのテキスト(TextView)を、画像の上側に配置するように変更します。

```
activity_main.xml
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:id="@+id/droid"
    android:layout_centerInParent="true" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="このイラストはドロイドくんです"
    android:layout_below="@+id/droid" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="上側に配置します"
    android:layout_above="@+id/droid" />
```

ただし、このまでは、図9のように、テキストは画像の上下に配置されますが、左端に寄ってしまっている状態になります。

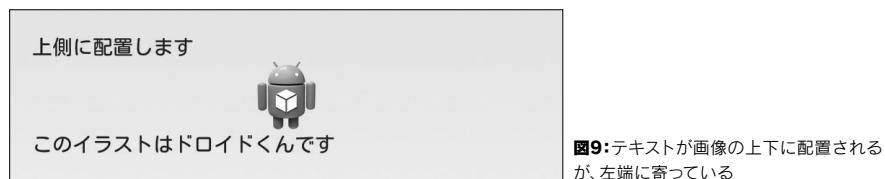


図9:テキストが画像の上下に配置されるが、左端に寄っている

テキストをこの位置で中央にそろえたいときは、以下のコードのように、6行目と12行目、18行目に「`android:layout_centerHorizontal="true"`」を追加します。

```
activity_main.xml
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:id="@+id/droid"
    android:layout_centerInParent="true" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="このイラストはドロイドくんです"
    android:layout_below="@+id/droid"
    android:layout_centerHorizontal="true" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="上側に配置します"
    android:layout_above="@+id/droid"
    android:layout_centerHorizontal="true" />
```

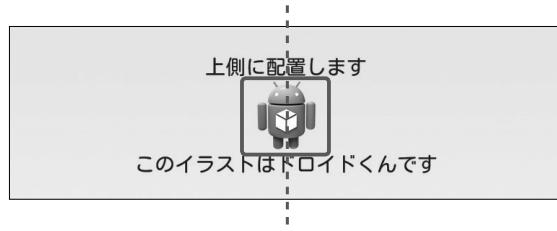


図10:テキストが中央にそろえられる

全体に対して中央ぞろえ

同様に、テキストの左端を画像の左端にそろえたいときは、12行目と18行目を「`android:layout_toLeftOf="@+id/droid"`」に変更します。

```
activity_main.xml
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:id="@+id/droid"
    android:layout_centerInParent="true" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=" このイラストはドロイドくんです "
    android:layout_below="@+id/droid"
    android:layout_toLeftOf="@+id/droid" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=" 上側に配置します "
    android:layout_above="@+id/droid"
    android:layout_toLeftOf="@+id/droid" />
```



図11:画像の左端にテキストがそろえる

画像に対して左端をそろえる



演習問題

ここまで、RelativeLayoutは何かを基準としてビューを配置することを学びました。基準とするものは、親ビューでも、任意のビューでもかまいません。自分で決めることができます。

それでは、RelativeLayoutを使って図12のようなレイアウトを作成してみましょう。言わずと知れた、日本地図です。
どのようなやり方でもかまいません。素材ファイルにあるバラバラになった画像(図13)を使い、ImageViewを組み合わせてこのレイアウトを完成させましょう。

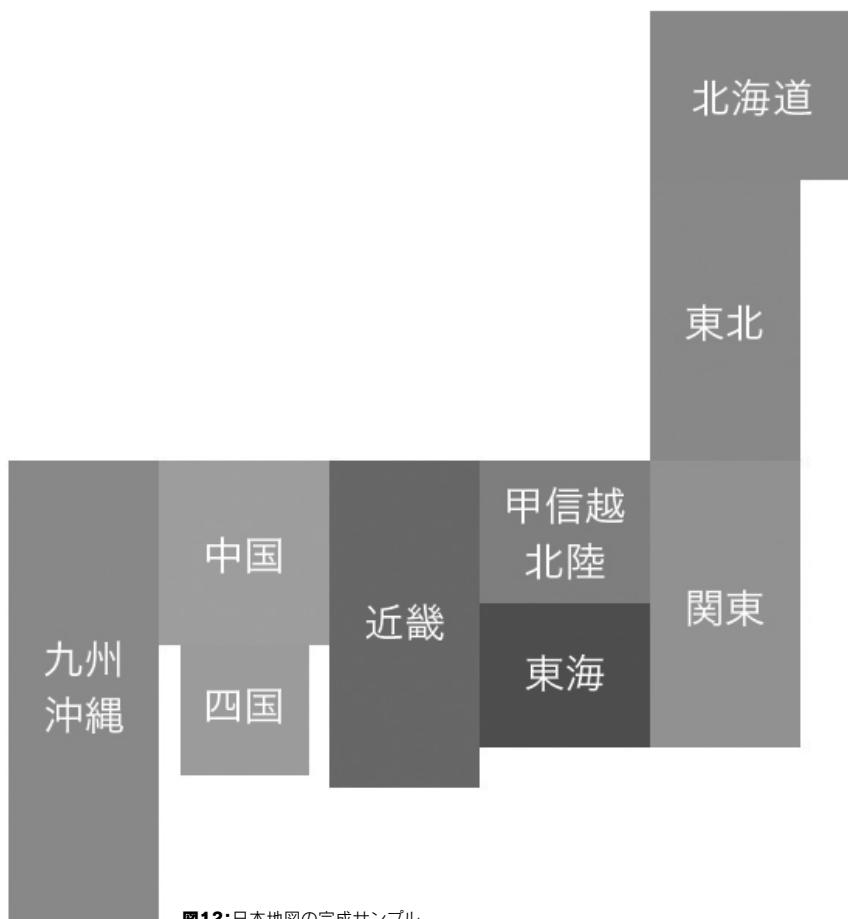


図12:日本地図の完成サンプル

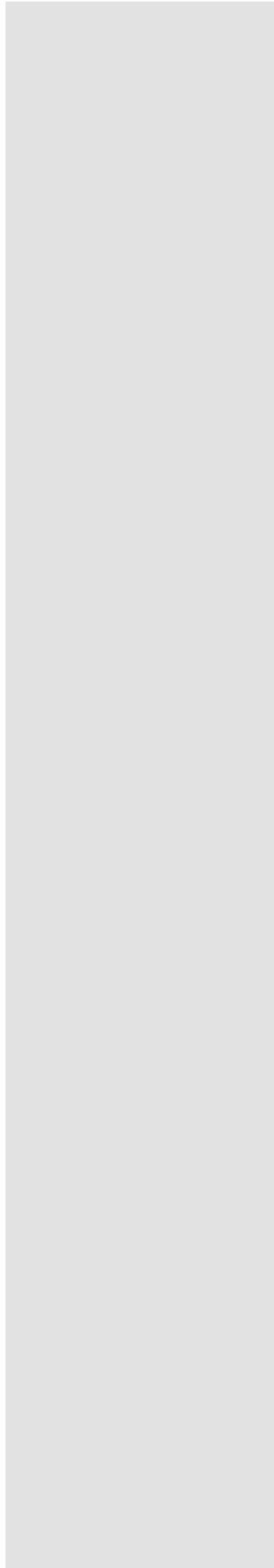


図13:この演習問題で使う素材。地方ごとにバラバラの画像になっている

演習用に、まず「UIBasicMap」という新規プロジェクトを作成します。

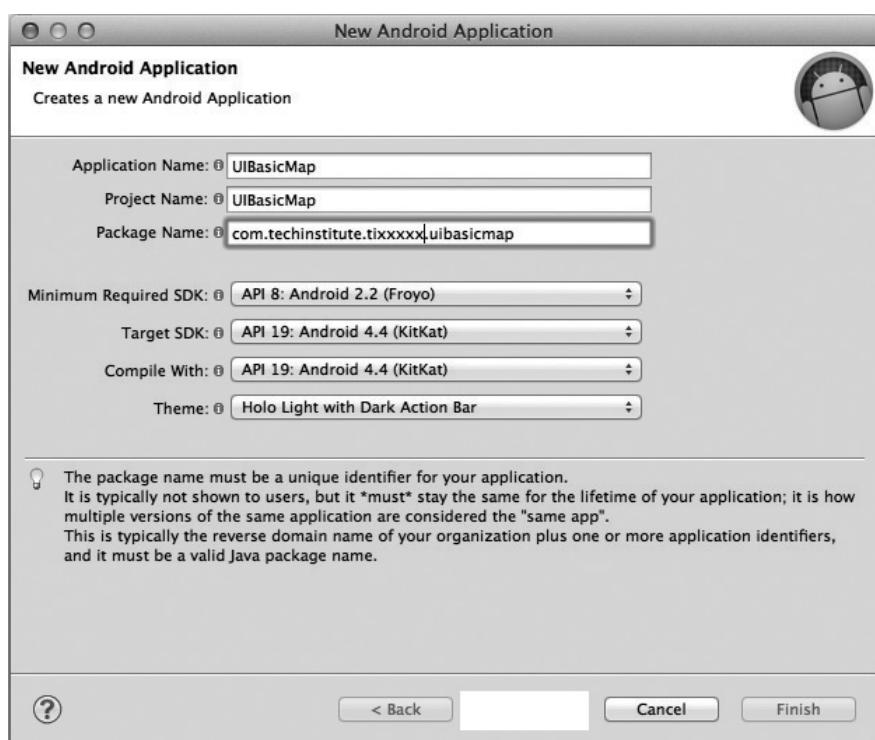


図14:UIBasicMapという新規プロジェクトを作成する

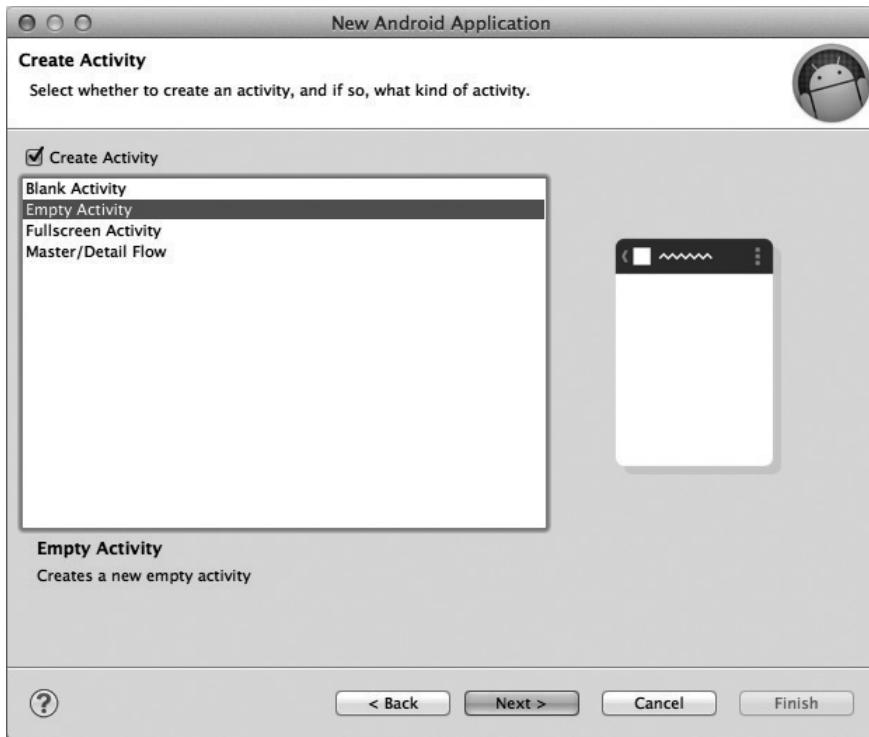


図15:Empty Activityを選択する

素材ファイルの中身を、「res -> drawable-xhdpi」フォルダー中にコピーします。コピー時に、「コピーをしますか？ それともリンクをしますか？」と聞かれるので、「Copy files」を選択し、「OK」をクリックします。

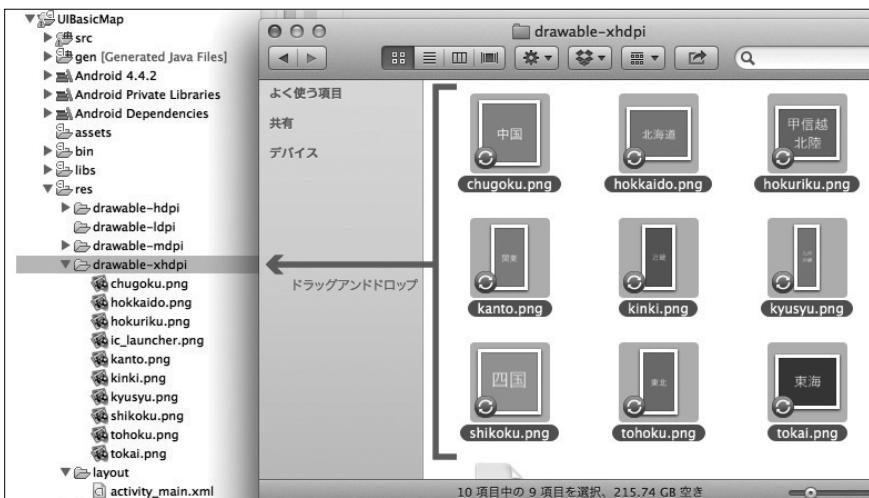


図16:素材をコピーする

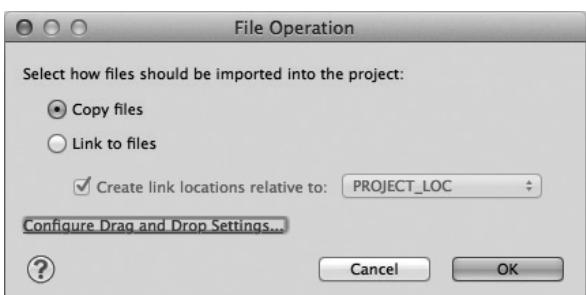


図17:「Copy files」を選択し、「OK」を押す

ここまでできたら、<ImageView>を作り配置していきましょう。



8-4-4 LinearLayoutによるレイアウト作成

LinearLayoutは、線形レイアウトの意味で、配置したビューが単純に直線的に並びます。配置したビューはRelativeLayoutのときのように重なることなく、横方向に、または縦方向に並びます。

LinearLayoutに必要な属性は、「`android:layout_width`」「`android:layout_height`」の他に、「`android:orientation`」という属性が必要になります。これは、どの方向にビューを並べていくかを決める属性で、次の2つの値があります。

値	説明
<code>horizontal</code>	中のコンテンツを水平方向(横方向)に並べる
<code>vertical</code>	中のコンテンツを垂直方向(縦方向)に並べる

図4:「`android:orientation`」に設定する値

ADTを起動し、新たにAndroidプロジェクトを作成します。

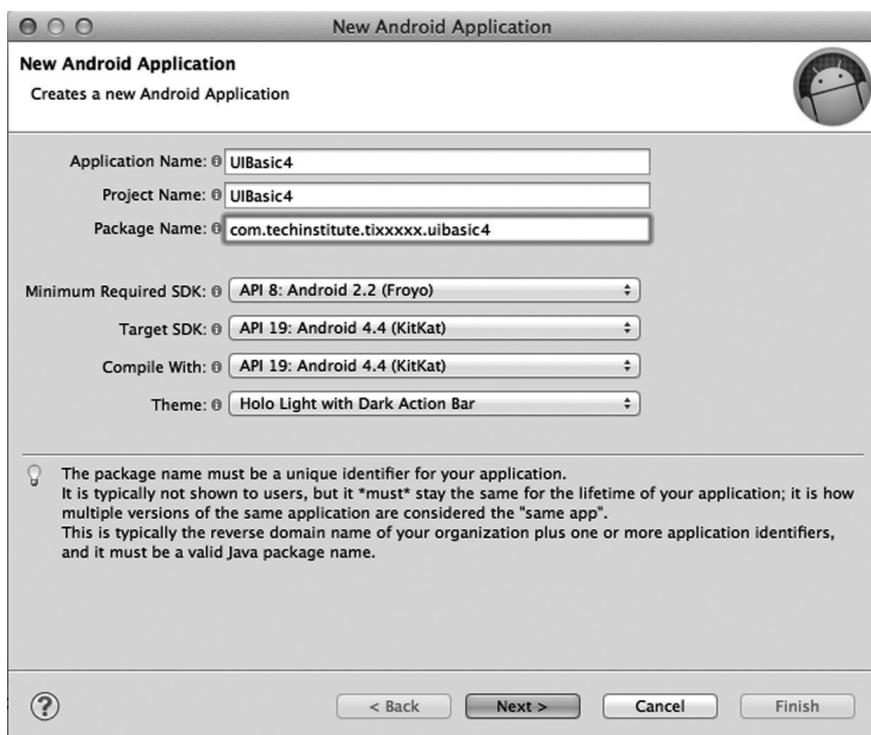


図18:UIBasic4という新規プロジェクトを作成する

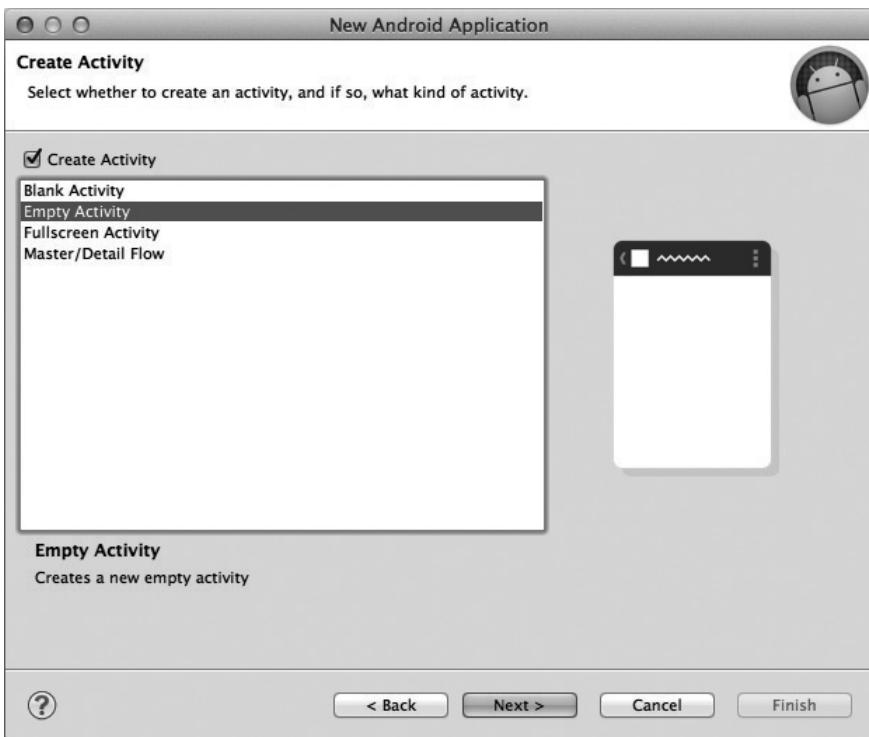


図19:Empty Activityを選択する

activity_main.xmlの中身を、以下のように書き換えます。

```
activity_main.xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="この内容で登録します。よろしいですか？" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="登録する" />
</LinearLayout>
```

「`android:orientation="horizontal"`」を指定することで、テキストとボタンは横並びに配置されます(図20)。

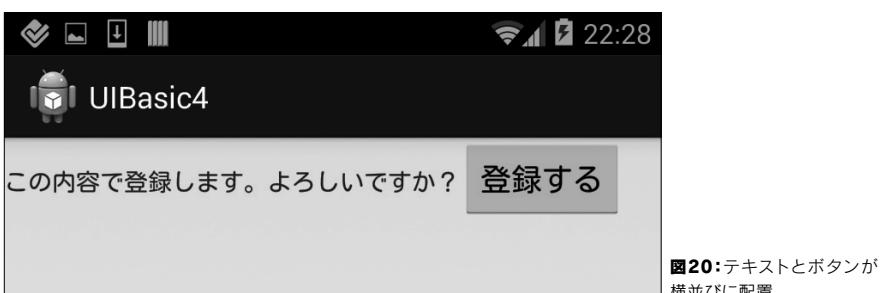


図20:テキストとボタンが横並びに配置

この場合、LinearLayout内にたくさんのビューを配置してしまうと、画面の横幅内に入りきらないものは表示されなくなってしまうので、注意してください。

試しに、次のコードのように、LinearLayout内に5つのボタンを入れてみます。

```
activity_main.xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録する 1" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録する 2" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録する 3" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録する 4" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登録する 5" />
```

ボタンが多すぎるとスマートフォンの横幅には入りきらないので、実際には図21のように表示されます。

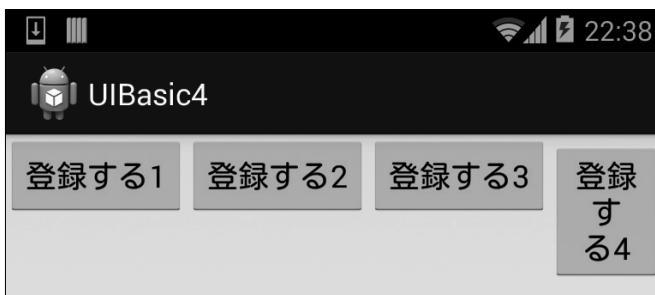


図21:「登録する4」はぎり入っているが改行されて崩れる。「登録する5」は表示されない

このようにレイアウトを崩さないために、LinearLayoutを使うときには、レイアウトに含まれるコンテンツの量に注意しましょう。

次に、縦並びを試してみましょう。先ほどのコードの5行目を以下のように変えると、コンテンツが縦方向に並ぶため、テキストとボタンは縦並びに配置されます。

```
activity_main.xml の 5 行目を変更
android:orientation="vertical"
```

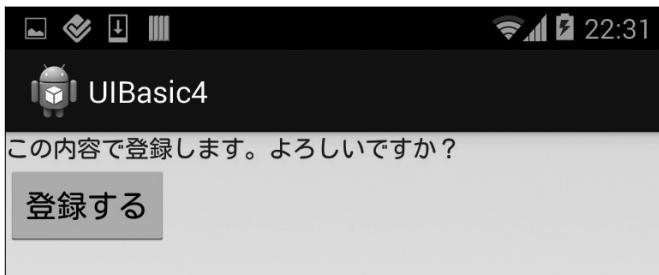


図22:テキストとボタンが
縦並びに配置

図22で確認できるように、たくさんのビューを配置すると、縦方向にどんどん並んでいきます。

領域の大きさを比率で指定する

LinearLayoutの大きな特徴は、領域の大きさを比率で指定できることです。たとえば、3つのボタンを横方向に並べます。

`activity_main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="A" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="C" />
</LinearLayout>
```

このソースコードの場合、「`android:layout_width`」が`wrap_content`ですので、指定されたテキストの文字数分しか横幅を持ちません(図23)。

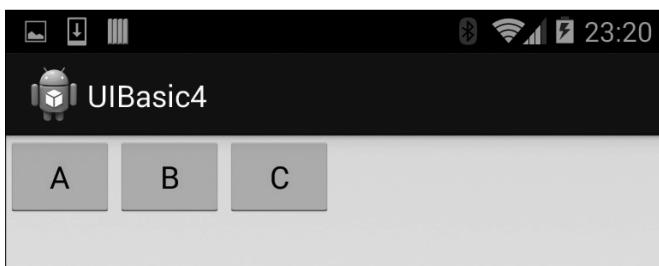


図23:ボタンの横幅は指定
されたテキストの文字数分

それぞれのボタンを横幅いっぱいに均等配列(3等分に配置)したい場合、ボタン要素を次のコードのように書き換えます。

Button 要素

```
<Button  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="A" />
```

各Button要素の、2行目の「`android:layout_width`」を「`0dp`」にし、4行目に「`android:layout_weight="1"`」を追加しました。3つのボタンすべてにこの変更を適用すると、横幅いっぱいに3等分されて配置されます(図24)。

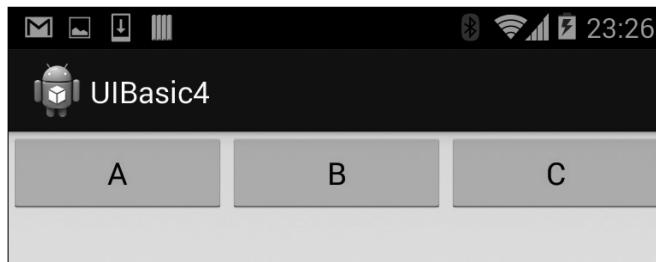


図24:ボタンが横幅いっぱいに3等分されて配置

ここで利用した「`android:layout_weight="1"`」とは、割合を決める属性です。すべてに1を設定している場合、A:B:Cの割合が、1:1:1であることになります。

また「`android:layout_width`」を「`0dp`」にするのは、いってみれば、「横幅における、ビューを占める領域の設定を無効にする」ということに近い意味になります。

では次に、A:B:Cの割合を、1:2:4にしてみましょう。各Button要素のそれぞれの「`android:layout_weight`」の値を、順に「1」「2」「4」に変えればいいですね。

A : B : C の割合を 1 : 2 : 4 にする

```
<Button  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="A" />  
  
<Button  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_weight="2"  
    android:text="B" />  
  
<Button  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_weight="4"  
    android:text="C" />
```

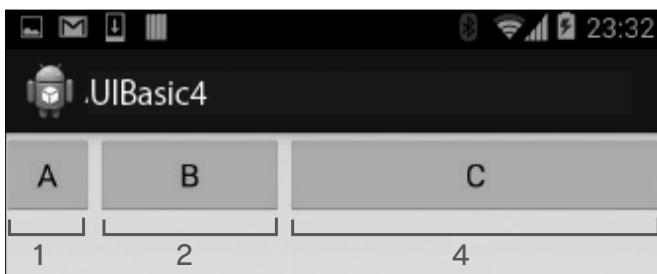


図25:A:B:Cの割合を1:2:4に変更

図25では横方向について設定をしましたが、これは縦方向についても同じことができます。

次は画面全体を使って、図26のように、縦方向にボタンを並べ、ボタンの高さがA:B:C:D=1:2:2:4になるようにしてみましょう。

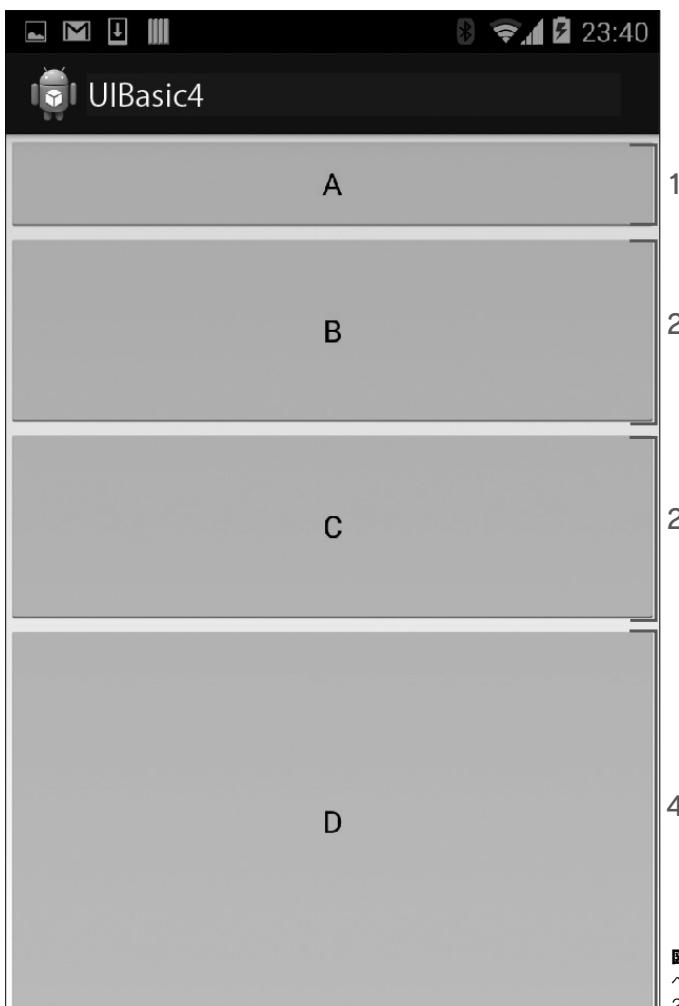


図26:ボタンを縦方向に並べ、高さがA:B:C:D=1:2:2:4になるように配置

ここまで例では、ビューのすべてに「`android:layout_weight`」を設定しましたが、実はすべてに設定をする必要はありません。一部の高さがすでに決まっている場合は、残った領域を比率によって配分することになります。

たとえば、図26のAとDを、コンテンツの成り行きの高さに合わせたとします(`android:height="wrap_content"`)。そうすると、BとCで残った領域を比率によって配分することになります。

次のコードのように、BとCの比率をB:C=1:3に変更すると、図27のように表示されます。

AとCはコンテンツの成り行きの高さになり、残りをB:C = 1 : 3となるように配分

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="A" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:text="B" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="3"  
    android:text="C" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="D" />
```

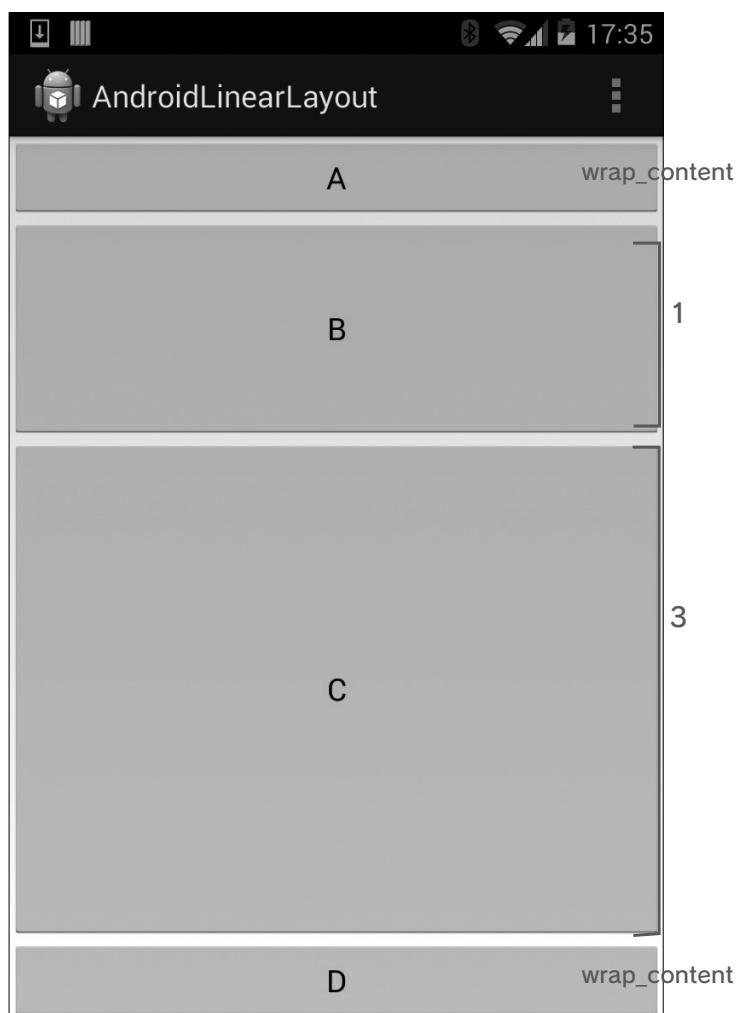


図27:AとCはコンテンツの成り行きの高さになり、残りをB:C = 1:3となるように配分]した結果



8-4-5 いろいろな行ぞろえ／列ぞろえを決める Gravity

Gravityとは、重力の意味です。Androidでは、さまざまなページを内包したLinearLayoutや、個々のビューに「揃え(そろえ)」を指定することができる要素です。「そろえ」という説明をしましたが、ビューに重力をかけて、配置する位置を決めるというと、イメージしやすいかもしれません。

実際に利用できるGravityには2種類あります。

属性	説明
android:gravity	親となるビューに指定することで、中に入るコンテンツのそろえを設定する
android:layout_gravity	それぞれのビューに指定することで、そのビューのそろえを設定する

表5:Gravityの種類

android:gravity

android:gravityは、親となるビューに指定する属性です。これを指定すると、中に入るコンテンツのそろえを設定することができます。

まず、次のコードのXMLで考えてみます。

縦方向にコンテンツが並んでいくLinearLayoutの中に、ボタンが4つ入っているシンプルなものです。

ボタン4つを並べる

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="xxxxxx" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="A" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="C" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="D" />
</LinearLayout>
```

上記のコードの6行目の、「`android:gravity="xxxxxx"`」の「`xxxxxx`」に入るもののうち、よく使う要素は次のとおりです。

値	説明
<code>top</code>	上に寄せる
<code>bottom</code>	下に寄せる
<code>left</code>	左に寄せる
<code>right</code>	右に寄せる
<code>center_vertical</code>	垂直方向中央にそろえる
<code>center_horizontal</code>	水平方向中央にそろえる
<code>center</code>	画面の中央にそろえる

表6: `android:gravity`の値

指定なし、`bottom`、`right`をそれぞれ指定したものが図28です。



図28: 親の`LinearLayout`に対する`android:gravity`の指定

右下に配置したいときには、「`android:gravity="right|bottom"`」といった具合に、値を「!(ビットOR)」でつなぐと、複数指定することができます。

コードで覚えるにときには、なるべく自分の中にイメージ持てるよう工夫するといいでしょう。たとえば、Gravityは、先ほども述べたとおり、重力の意味があります。親となるLinearLayoutにどのような重力をかけるか、ということを考えるとイメージしやすいかもしれません。

また、Gravityはレイアウトだけでなくボタン(Button)やテキスト(Text View)といったビューにも指定できます。

たとえば、先ほどのソースコードのBのボタンをテキスト(Text View)に変えてみます。そしてこのテキストについて、横方向にmatch_parentにします。

Bのボタンを TextView に変え、「`android:layout_width="match_parent"`」に変更

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="right" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="A" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=" テキストを設定します " />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="C" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="D" />
</LinearLayout>
```

そうすると、横方向にwrap_contentであるA、C、Dのボタンは、親のLinearLayoutの「右ぞろえ(6行目の「`android:gravity="right"`」)」を受けてすべて右端に寄っていますが、テキストだけmatch_parentなために、横幅いっぱいに領域が伸びています(図29)。

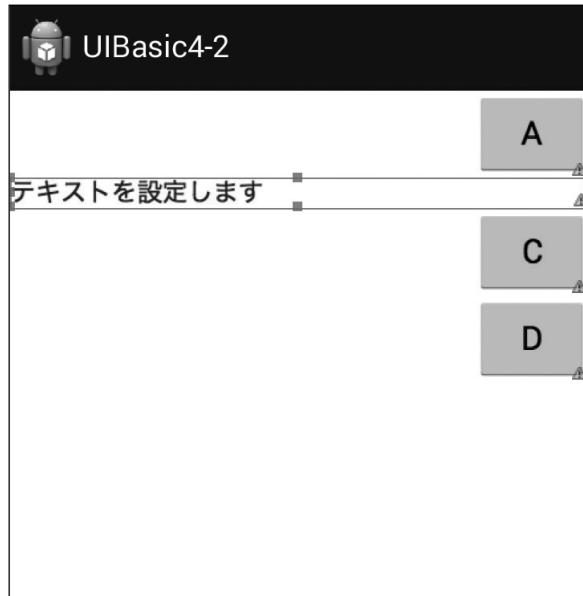


図29:テキストの領域は横幅いっぱいに伸びる

ここで、`TextView`に「`android:gravity="right"`」を付け加えると、テキストが右寄せになります(図30)。

TextViewに「`android:gravity="right"`」を追加

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:gravity="right"  
    android:text="テキストを設定します" />
```

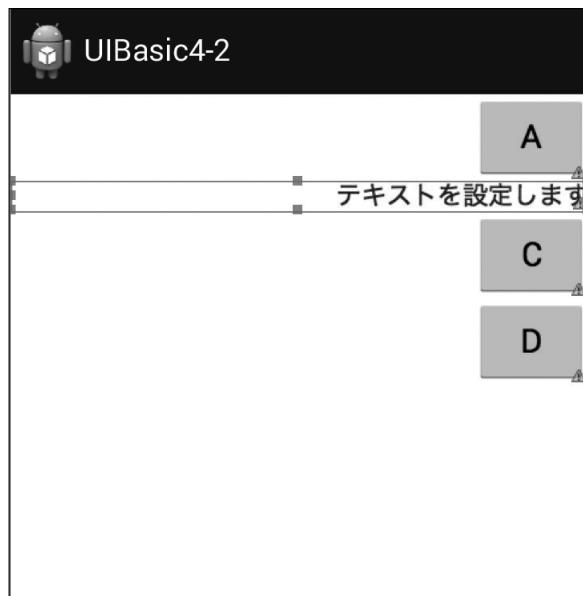


図30:テキストのみ右寄せになる

「`android:gravity`」は、どの要素にも設定することができます。「箱全体に重力をかける」というイメージをもっておくとよいでしょう。箱全体に特定の方向の重力をかけると、中の要素が指定した方向に寄っていくのは、自然界でも容易に想像できますよね。

android:layout_gravity

android:layout_gravityは、それぞれのビューに指定することで、そのビューの配置を設定する属性です。これらのビューには、配置する方向にwrap_contentが指定されている必要があります。

たとえば、図31には「A」～「D」の4つのボタンがありますが、すべて横方向にwrap_contentが指定されています。そして、デフォルトの左寄せになっています。

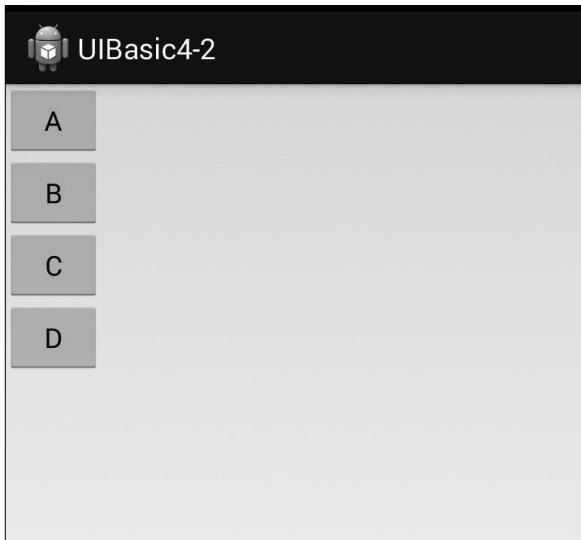


図31:4つのボタンを配置

4つのボタンを配置

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="A" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="C" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="D" />
</LinearLayout>
```

ここでBのボタンのみを右寄せに変更したい場合、ボタン「B」のコード(属性)に「`android:layout_gravity="right"`」を追加します。

android:layout_gravity を追加する

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="right"  
    android:text="B" />
```

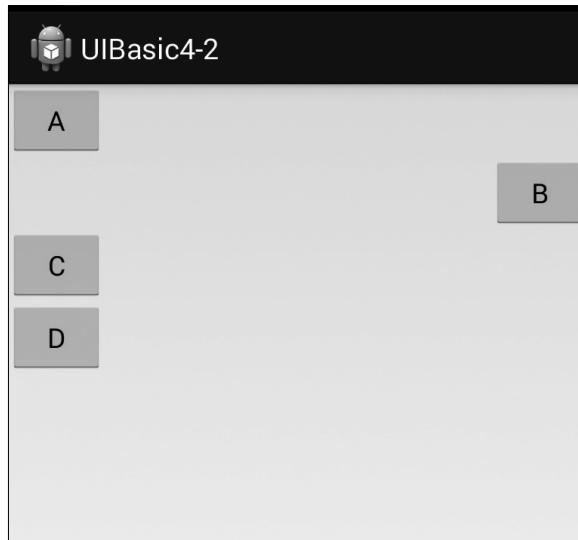


図32:Bのボタンのみ右寄せに配置

「android:layout_gravity」に指定できる値のうち、よく使うものは次のとおりです。

値	説明
top	上に寄せる
bottom	下に寄せる
left	左に寄せる
right	右に寄せる
center_vertical	垂直方向中央にそろえる
center_horizontal	水平方向中央にそろえる
center	画面の中央にそろえる

表7:android:layout_gravityの値

「android:gravity」と「android:layout_gravity」とで矛盾するような指定をした場合には、内側のビューの「android:layout_gravity」の指定が優先されます。

ほとんどが、「android:gravity」と同じと考えてもよいでしょう。

違うのは、箱全体に重力をかけるのか、または箱の中の各要素に個別に重力をかけるのか、の違いです。

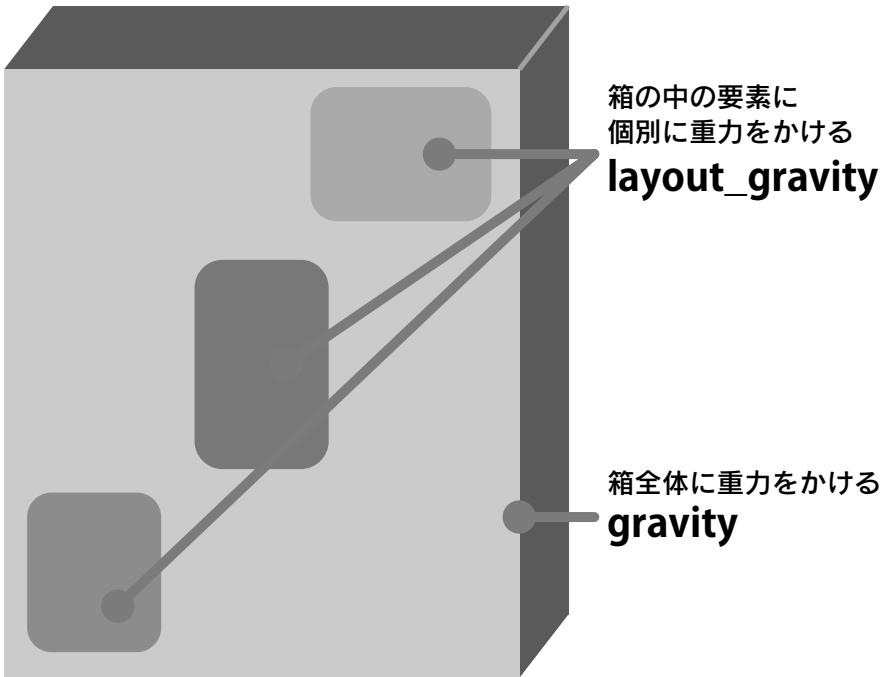


図33:gravityとlayout_gravityの違い

Androidのレイアウト方法

ここまでで、Androidの基本的なレイアウト方法を学びました。

RelativeLayout／LinearLayoutは、それぞれの考え方方が違いますが、さまざまな画面サイズの端末に簡単に対応できる、という点では共通しています。

RelativeLayoutであれば、親ビューを基準にしたり、もしくは任意のビューを基準にして配置します。

LinearLayoutであれば、ビューの占める領域を比率によって指定することができます。

これらは、HTML／CSSによるWebサイトのレイアウトや、iPhoneアプリの画面レイアウトとはまったく違った考え方を持っていて、画面サイズがどんなに変わってもそのときに応じて配置できるというのが特徴です。これは、AndroidというOSがオープンソースであり、多くのキャリア（携帯事業者）や端末メーカーが利用していて、どのような画面サイズの端末でも動く、ということにもつながります。

極端な例では、縦横比すらまったく違うような、正方形のような画面サイズのものもあるでしょう。

多種多様な画面サイズがある中で、すべてで完璧に動作させることは難しいですが、レイアウトを作成する段階で、画面サイズの変化に柔軟に対応できるように心がけましょう。



演習問題

今までの内容の復習として次のレイアウトを作成してみましょう。



図34:完成サンプル

演習用に、まず「UIBasicPhone」という新規プロジェクトを作成します。



図35:UIBasicPhoneという新規プロジェクトを作成する

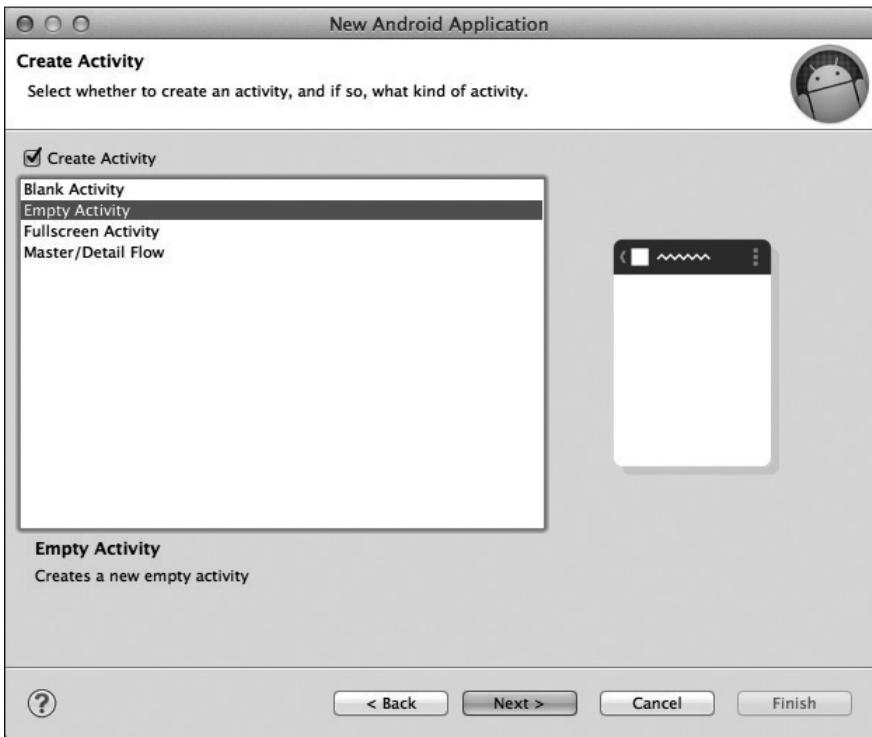


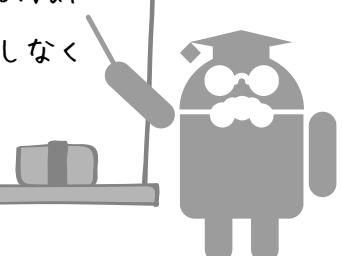
図36:Empty Activityを選択する

ここまでできたら、ボタン(Button)を配置してレイアウトを作成していきましょう。

8-5 UIの基礎知識（5）

著：秋葉ちひろ

Androidプロジェクトのレイアウトを構成する「res」フォルダーや中に、「values」というフォルダーがあります。このvaluesフォルダー内のXMLを使えば、1つのXMLを変更しなくても、さまざまな値を一括して管理・変更できます。



この節で学ぶこと

- 1 valuesフォルダーのstrings.xmlを活用する
- 2 valuesフォルダーのcolors.xmlを活用する
- 3 valuesフォルダーのstyles.xmlを活用する
- 4 marginとpaddingの違いを学び、余白の調整をする

この節で出てくるキーワード一覧

values
strings.xml
colors.xml
styles.xml
他言語化
ローカライズ
android:layout_margin
android:padding



8-5-1 「values フォルダー」を活用しよう

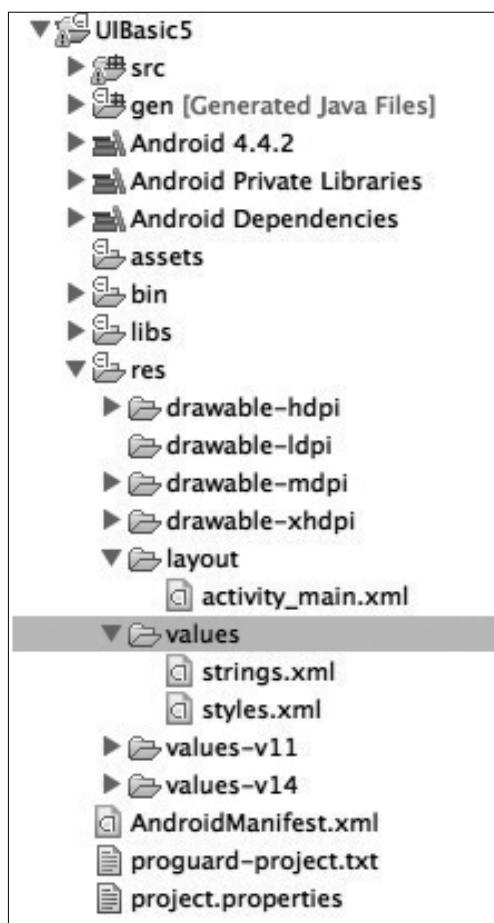


図1:valuesフォルダー

valuesフォルダーに入っているXMLは、さまざまな値を一括して管理するデータシートのような役割をします。たとえば、アプリ内で使う色を例に挙げてみましょう。やみくもにいろいろな色を使うのではなく、最低でも

- ・メインカラー
- ・サブカラー
- ・アクセントカラー
- ・ベースとなるバックグラウンドカラー

の4つは、あらかじめ決めて設計をしていくことが望ましいです。

アプリを作っていく際には、いろいろなところにこれらの色を使っていくでしょう。

TextViewでメインカラーを使う、背景でバックグラウンドカラーを使う、また他の画面でもメインカラーを使う等々、レイアウトファイルでは、これらの色の指定をする部分がかなり多く出てきます。

レイアウトファイルだけでなく、Javaファイルでも色の指定をするときは、さらに指定箇所は増えます。

カラーコードは、今回のように3行で表示することもあります。
3行の場合、たとえば「#c00」だと、6行であらわすと「#cc0000」となり、2桁ごとで同じものが続く場合のみ、3行に省略することができます。

アルファも、同じく16進数で2桁同じものが続く場合、1桁に省略することができます。
「#5fc0」←「#55ffcc00」の省略形

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00"  
    android:text="Eメール" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00"  
    android:text="パスワード" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00"  
    android:text="パスワードの確認" />
```

それぞれの TextView で同じ色を指定している

図2:通常のレイアウトXMLで色を指定

図2の例では、TextViewで同じ色(#c00:濃い赤色)を指定しているところが3カ所あります。#c00をメインカラーとしたとき、この3つのテキストは常に同じ色であることを求められます。しかし、いろいろな事情で、「やっぱりメインカラーをちょっと変えよう」ということも、アプリを開発している途中でよく起こることです。

ですが、「ちょっと変えよう」というときに、図3の例では、3カ所所すべてのカラーコードを変更しなければなりません。

3カ所だけならまだしも、実際には他の画面でもメインカラーはあちこちで使われています。また、プログラムのJavaでもこのカラーコードが書かれているとしたら、すべての箇所を変更しなければなりません。

これは、大変膨大な作業で、またミスを引き起こしやすい状態となってしまいます。

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00" → #f00  
    android:text="Eメール" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00" → #f00  
    android:text="パスワード" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textColor="#c00" → #f00  
    android:text="パスワードの確認" />
```

すべての箇所のカラーコードを変更しなければならない

図3:変更箇所が点在しているので、ミスを引き起こしやすい

こういったときに、valuesフォルダーのXMLを使って、「メインカラーは#c00である」ということをあらかじめ定義しておくと、「ちょっと変えよう」ということが起こっても、定義している部分の1カ所だけを修正することで、すべてが反映されます。

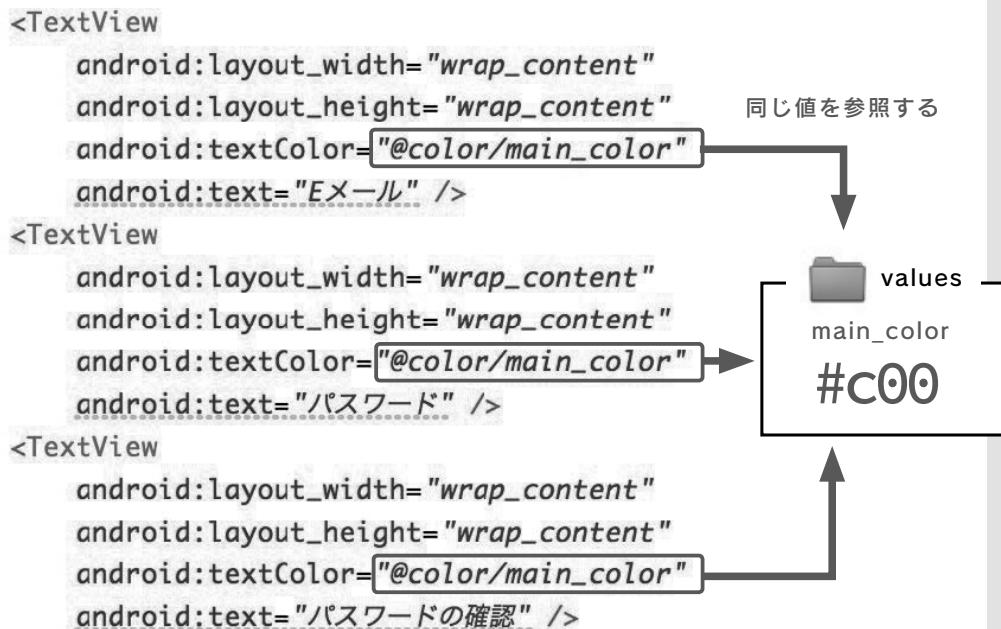


図4:別ファイルで色を一括管理

言い換えると、valuesフォルダーのXMLを使ってさまざまな値を一括して管理することは、メンテナンス性の向上にもつながります。

valuesフォルダー内で管理できるもの

valuesフォルダー内で管理できるものは、ガイドラインの「More Resource Types」を参照してください。その中でもよく使うものは、次のとおりです。

ファイル名	説明
strings.xml	アプリ内の文字列
colors.xml	アプリ内の色
styles.xml	レイアウトの属性をまとめる

表1:XMLファイルとその役割

何の設定をするかによって、ファイルが分かれています。

それでは、実際に新規プロジェクトを作成して、詳しく見てていきましょう。

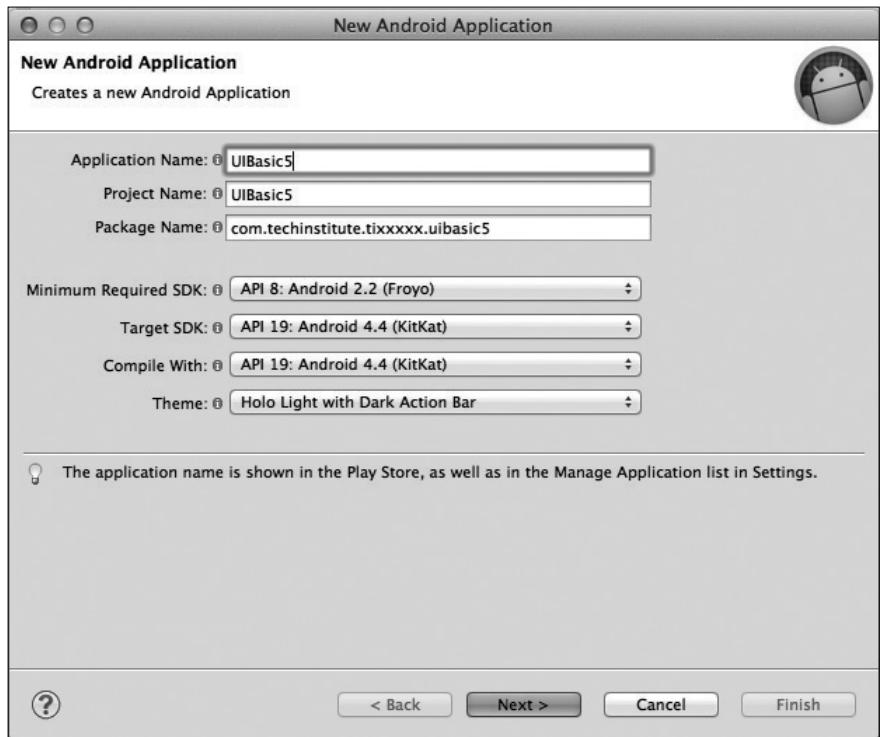


図5:「UIBasic5」という新規プロジェクトを作成する

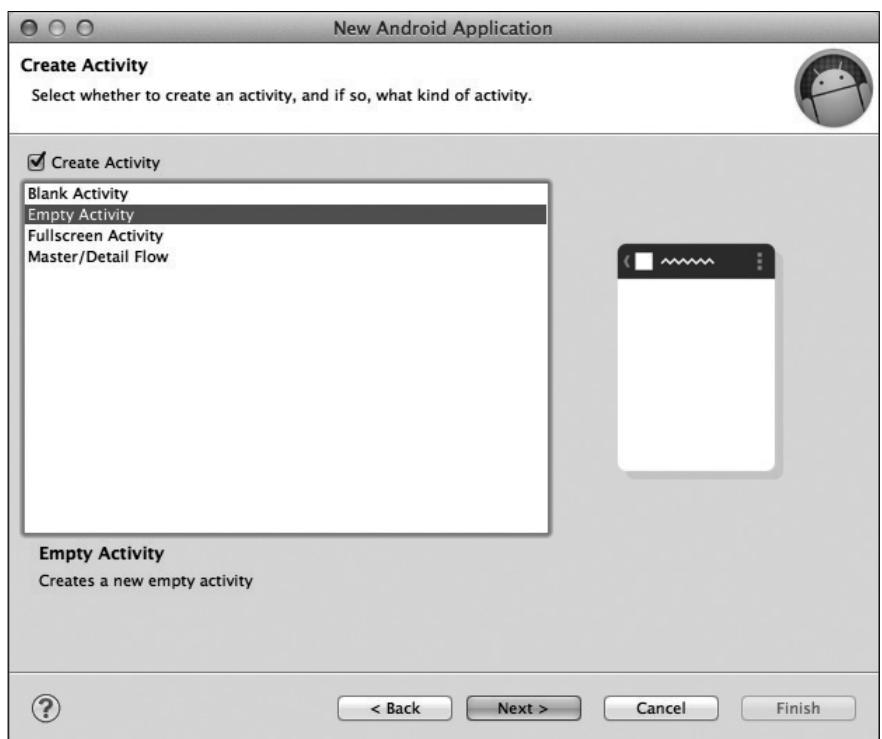


図6:Empty Activityを選択する



8-5-2 strings.xml で、文字の装飾や多言語対応も一発

「strings.xml」は、文字列を管理するファイルです。

新規Androidプロジェクトを作成すると、このファイルはあらかじめ存在していて、最初から次のように定義されています。

```
string.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">UIBasic5</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

この文字列は、実際のレイアウトと照らし合わせると、次の部分の文字列になります。

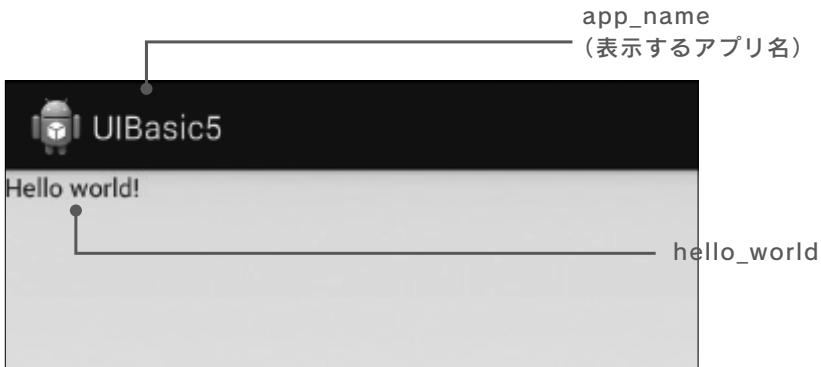


図7: 文字列の照合

この中で、レイアウトXMLから参照されているのは「Hello world!」で、アプリ名は、AndroidManifest.xmlから参照されています。

レイアウトXMLから参照する場合、次のように書きます。

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

「android:text」の部分で、参照するファイルの名前(ここでは「string」、単数形にする)の前に「@」をつけ、その後に「/(スラッシュ)」と、設定したnameを入れます。

多言語対応をする

文字列をvaluesフォルダーで管理することのメリットは、多言語対応です。

Androidアプリをひとたびリリースすれば、それは全世界で公開されます。より多くの人に使ってもらおうとするなら、日本語だけではなく、最低でも英語対応は必須でしょう。

もしも、文字列をそのままXMLに直接表記してしまっていたら、日本語版は日本語用のレイアウト、英語版は英語用のレイアウトなどのように、何通りも同じようなレイアウトXMLを作らなければなりません。これは、とても骨の折れる(無駄な)作業です。

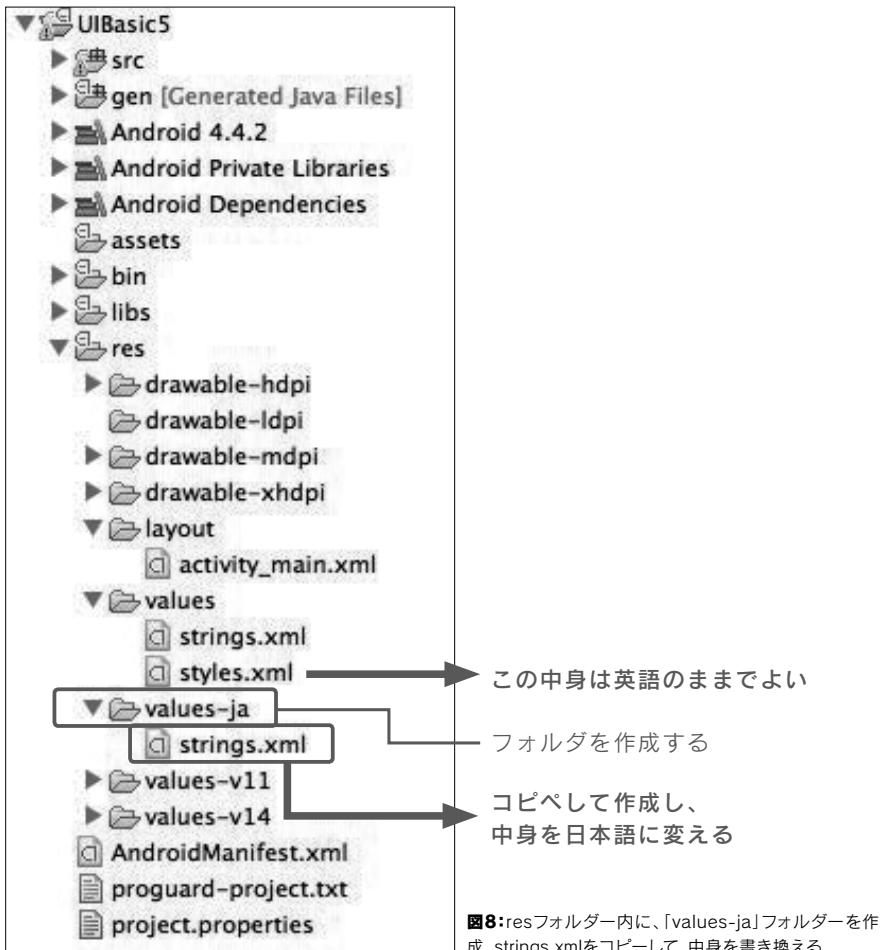
また、修正が入ってしまった場合に、すべてのレイアウトXMLを更新するのも現実的ではありません。

ですが、Androidアプリは、多言語化しやすいように構成されています。今回のようにstrings.xmlを参照することで、文字列の部分の役割を、レイアウトXMLから外へ出すことができます。そして、日本語用のstrings.xml、英語用のstrings.xmlを準備することで、簡単にAndroid端末の言語環境にあわせることができます。

新規に立ち上げたサンプルアプリで、英語版にも対応するように作り変えてみましょう。

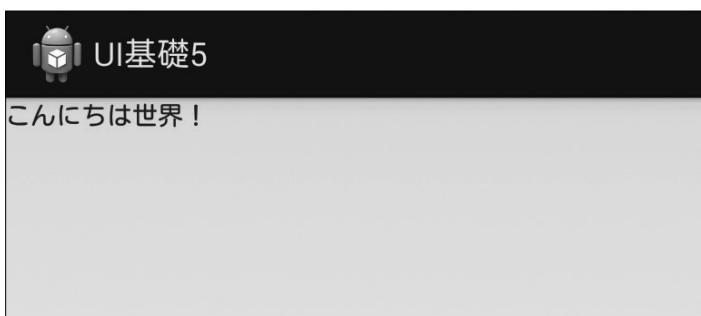
- 1 「values」フォルダーと同じ階層に「values-ja」というフォルダーを作成する(図8参照)
- 2 「values-ja」フォルダー内に、「values」フォルダー内の「strings.xml」をコピーする(同じく図8参照)
- 3 新しくコピーしてきた「values-ja」フォルダー内の「strings.xml」の文字列を、以下のコードのように日本語にする

```
values-ja/string.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">UI 基礎 5</string>
    <string name="hello_world"> こんにちは世界！ </string>
</resources>
```



完成したら、エミュレーターまたは実機で確認をしてみましょう。

アプリを立ち上げてみて、「values-ja」で定義した日本語の文字列になっていれば、成功です。



■9: 文字列が日本語で表示される

どの言語を参照しているかは、使っているAndroid端末の言語設定に依存します。設定を日本語にしている場合は、「values-ja」フォルダー内のstrings.xmlが参照されます。もしフランス語の設定にしている場合は、「values-fr」フォルダー内のstrings.xmlが参照されます。しかし、フランス語に設定しているときに「values-fr」というフォルダーがない場合は、デフォルト（「values」フォルダー）のstrings.xmlを参照します。

「values-xx」の「xx」にあたるものは、ISO 639に定められている言語コードです。言語によっては、複数の地域で使われているものもありますが、それらは「values-xx_XX」となり、ISO 3166に定められている言語コード（リージョンコー

言語設定は、「設定」メニュー→「言語と入力」で変更できます。

ド)となります。

このように、特定の言語用のリソース(文字列や画像)を用意して、その言語に対応させることを「ローカライズ」といいます。

Androidアプリのローカライズでよく使う言語と言語コードを挙げておきます。

言語	言語コード
日本語	values-ja
英語	values-en
フランス語	values-fr
スペイン語	values-es
中国語(簡体字)	values-zh_CN
中国語(繁体字)	values-zh_TW
韓国語	values-ko

表2:ローカライズでよく使う言語／言語コード

図9のように日本語が表示されたら、今度はAndroid端末の言語設定を英語(English(United States))にして、再度アプリを確認してみましょう。

アプリ内の文字列が英語で表示されていたら、うまく多言語対応ができていることになります。

このように、strings.xmlをうまく使えば、レイアウトXMLやJavaを書き換えることなく、効率よく多言語展開ができることがわかります。もし他言語化展開をする必要がなかったとしても、文字列はstrings.xmlファイルで管理することが推奨されていますので、できるだけそうするようにしましょう。



8-5-3 colors.xmlで、色をまとめて管理

「colors.xml」は、色を管理するファイルです。

このファイルは、新規にAndroidプロジェクトを作成したときにはありませんので、必要なときには自分で作ります。先の例でも挙げましたが、配色を設計するときに指定した色を書いておくとよいでしょう。

まず、色を管理するサンプルXMLを作成します。ここで作成するサンプルは、図10のような簡単なリストです。

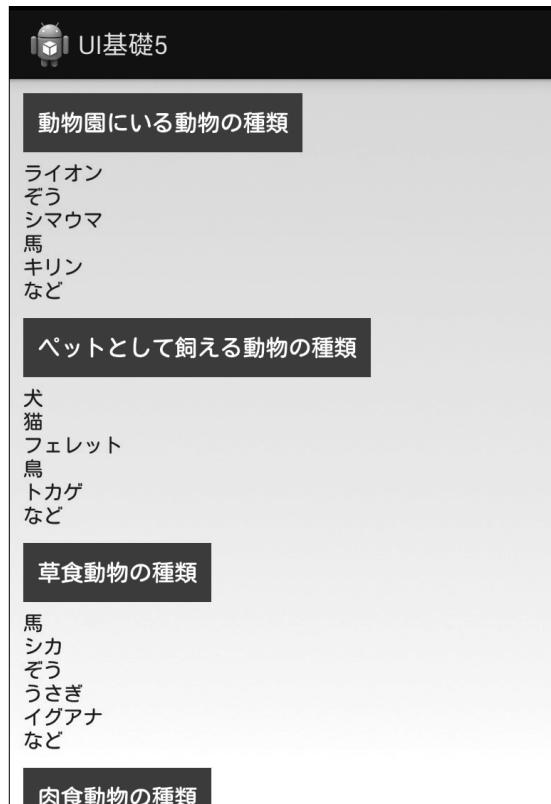


図10: /v見出しに色の付いたリストのサンプ

このサンプルのキーカラーは赤色で、見出しどなるビューの背景に適用されています。複数箇所に使われていますが、colors.xmlで一括管理されていますので、レイアウトXMLを修正することなく、簡単に色の変更ができるサンプルになっています。

また、少しデザイン的な要素も含んでいますので、デザインの勉強もあわせてていきましょう。

1 ビューを作成し、文字列を入れる

まず、LinearLayoutを作成し、中の要素が縦向きに並ぶように「android:orientation="vertical"」を指定します。その中にテキスト(TextView)を合計8つ(それぞれが見出しと本文を持ち、4x2となるように)作成します。

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/animal" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/animal_text" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pet" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pet_text" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/plant" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/plant_text" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat_text" />
</LinearLayout>
```

values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">XML基礎 - 色の管理 </string>
    <string name="animal">動物園にいる動物の種類 </string>
    <string name="animal_text">ライオン\nぞう\nシマウマ\n馬\nキリンなど </string>
    <string name="pet">ペットとして飼える動物の種類 </string>
    <string name="pet_text">犬\n猫\nフェレット\n鳥\nトカゲ\nなど </string>
    <string name="plant">草食動物の種類 </string>
    <string name="plant_text">馬\nシカ\nぞう\nうさぎ\nイグアナ\nなど </string>
    <string name="meat">肉食動物の種類 </string>
    <string name="meat_text">ライオン\nアライグマ\nクマ\nピューマ\nミーアキャット\nなど </string>
</resources>
```

上のソースコード内の「\n」という記号は、改行を意味するものです(HTMLでは「
」に該当するものです)。この2つのソースコードにより、図11のような状態を作成します。

しかし、これだけでは見出しと本文の違いがまったくわからず、見やすい状態ではありませんので少しデザイン要素を追加していきます。

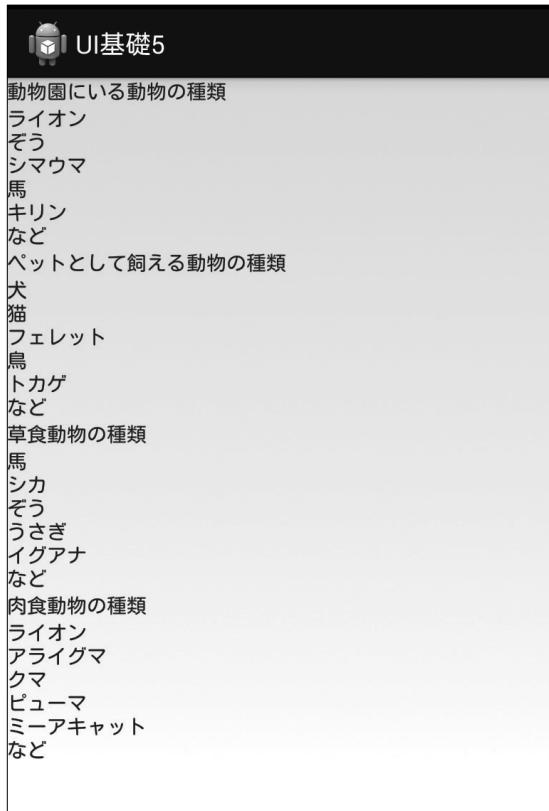


図11:とりあえず必要な要素を並べる

2 見出しの文字を少し大きくする

見出しと本文の違いをデザインによって表現するため、見出しの文字を大きくしてみましょう。

見出しどころの TextView (前述のactivity_main.xml の6~9行目、14~17行目、22~25行目、30~33行目の4ヶ所) に、次の2行を追加します。

「デザイン」と聞くと、絵を描くことを思い浮かべる人がいるかもしれません。それは少し偏った考え方です。「デザイン」には本来「設計」という意味が含まれており、絵やイラストなどを入れなくとも文字の大きさや余白を調整することで、デザインとなりうるのです。

見出しどなる TextView に追加

```
android:textSize="16sp"  
android:textStyle="bold"
```

テキストの大きさが16spになり、太字になりました。

このように、見出しの文字サイズを大きくしたり、太字にしたりすることはよくありますので覚えておきましょう。

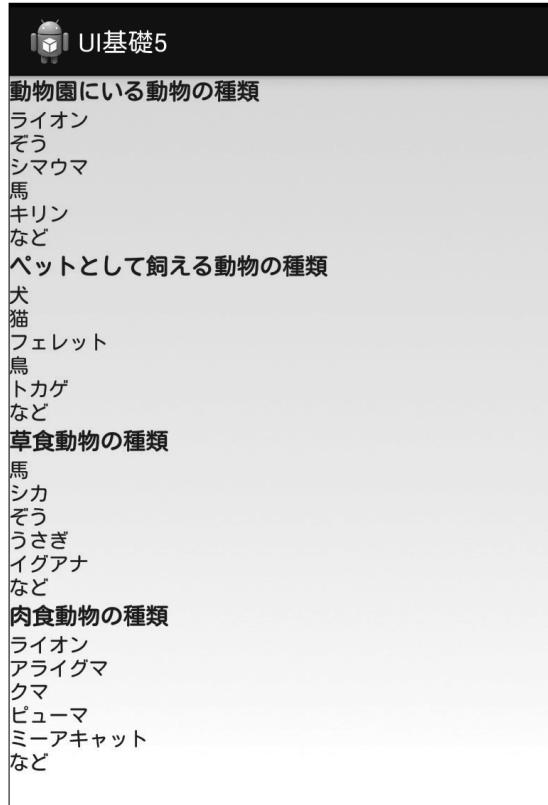


図12:見出しの文字サイズとスタイルを変更

3 見出しの文字を装飾する

次に、見出しの文字をもう少し目立たせるように装飾を加えましょう。

見出しの背景をキーカラー（少し濃い色がいいでしょう）にして、文字を白色にしてみます。先ほどと同じく、見出しどなるテキスト（TextView、4ヶ所）に次の2行を追加します。

見出しどなる TextView に追加

```
android:textColor="#fff"  
android:background="#b81c22"
```

この時点では、見出しどなる TextView は次のソースコードのようになっています。

見出しとなる TextView

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textStyle="bold"
    android:textColor="#fff"
    android:background="#b81c22"
    android:text="@string/animal" />
```

「`android:textColor`」は文字の色を決める属性、「`android:background`」はその要素を占める領域の背景色を決める属性です。それぞれ`#fff`(白)と`#b81c22`(濃い赤:キーカラー)に設定しました。

キーカラーは自分の好きな色に変えてもかまいません。

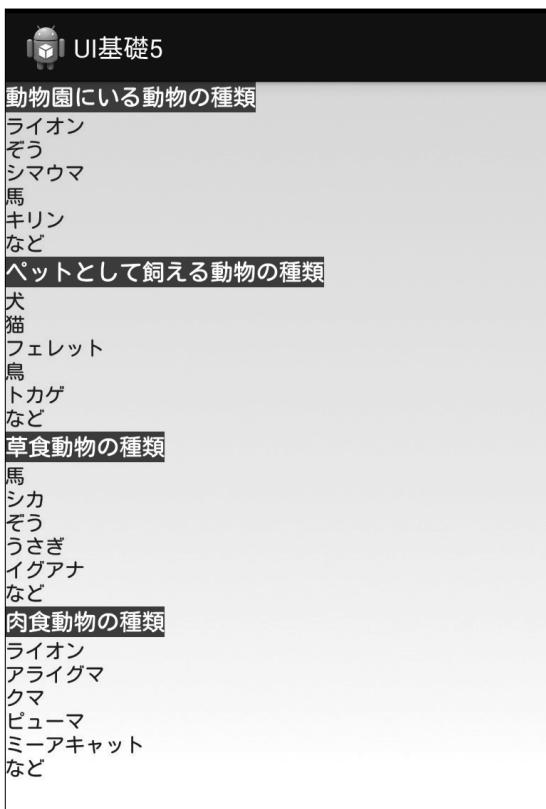


図13: 見出しとなるTextViewの背景色と文字色を変更

4 余白を調整する

さて、ここまで装飾をしてきて、すっかりデザインをした気になっていませんか？

よくある話なのですが、プログラマーは、ここまで見栄えができたら、アプリを公開してしまいます。しかし、この状態では「デザインされている」とは、まったくもっていません。情報としてはまちがっていないですし、問題ないのですが、読みやすさに関してまったく配慮がなされていないのです。

読みやすさを左右する重要な要素に、余白があります。

AndroidアプリのレイアウトXMLでは、余白を自動的にしてくれるようなシステム

は残念ながらありませんので、開発者ひとりひとりが気をつけながら作っていかなければなりません。

開発に関するプログラムを学ぶことは最低限必要なことですが、その後良いアプリが作れるかどうかは別問題。デザイン力も必要になってきます。プログラムといっしょに、デザインに関する知識もつけていくようにしましょう。

この図13において何がよくないのかというと、基本的に要素と要素がくっついているのが不自然なのです。

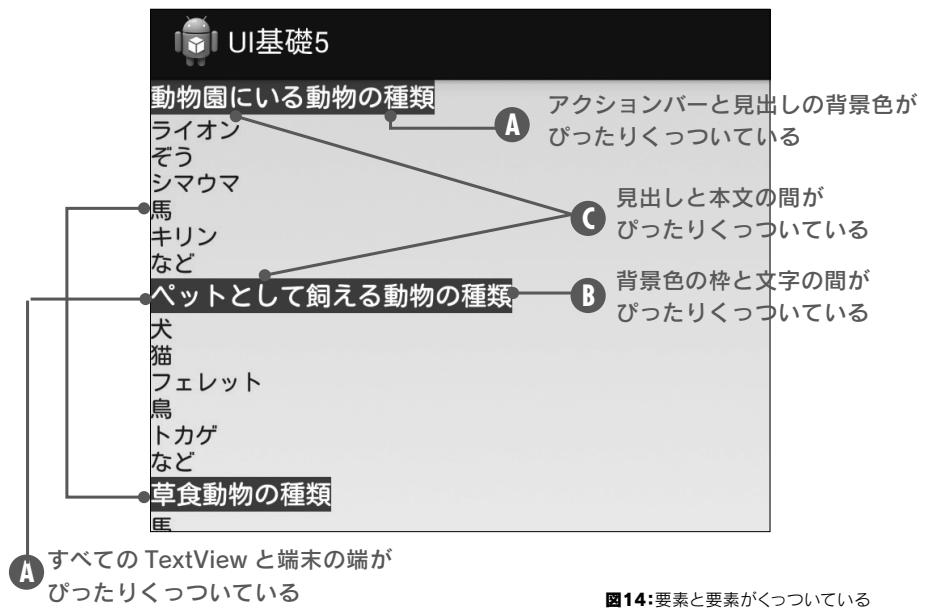


図14:要素と要素がくっついている

よほどのことがない限り(特にテキストの場合)、要素と要素がぴったりくっつくことはありません。図14の指摘箇所に、余白を追加していきます。

(A)については、全体的なViewに余白がないため、親となるView(ここではLinearLayout)に余白を追加します。

```
activity_main.xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:orientation="vertical">
    ~中略~
</LinearLayout>
```

(B)については、見出しとなるTextViewの内側に余白を追加します。

activity_main.xml

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="16sp"  
    android:textStyle="bold"  
    android:textColor="#fff"  
    android:background="#b81c22"  
    android:padding="10dp"  
    android:text="@string/animal" />
```

最後に(C)については、見出しと本文の両方のTextViewの下側に余白を追加します。

activity_main.xml

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="16sp"  
    android:textStyle="bold"  
    android:textColor="#fff"  
    android:background="#b81c22"  
    android:padding="10dp"  
    android:layout_marginBottom="5dp"  
    android:text="@string/animal" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginBottom="10dp"  
    android:text="@string/animal_text" />
```

最終的には、図15のようなデザインになります。

この状態では、コンテンツが長くなつてもスクロールがまだできない状態です

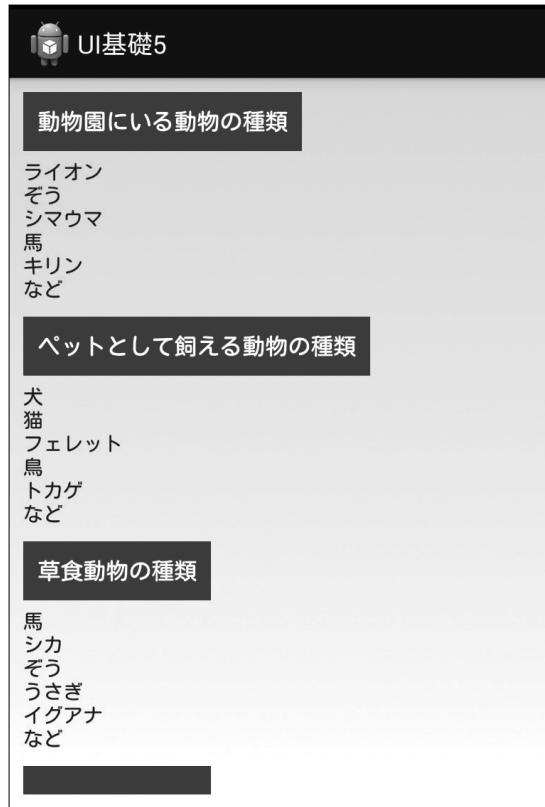


図15:余白を追加した状態

ここまで、余白を追加するのにそれぞれ違う種類の属性を使いました。

- (A) android:layout_margin
- (B) android:padding
- (C) android:layout_marginBottom

(C)については、android:layout_marginの下側に限った属性です。すでに想像できると思いますが、同じように

- android:layout_marginTop
- android:layout_marginBottom
- android:layout_marginLeft
- android:layout_marginRight

があり、基本的な考え方は「android:layout_margin」と同じものです。

それでは、marginとpaddingの違いは何でしょうか。同じ余白ではありますが、marginは外側の余白、paddingは内側の余白と解釈されます。

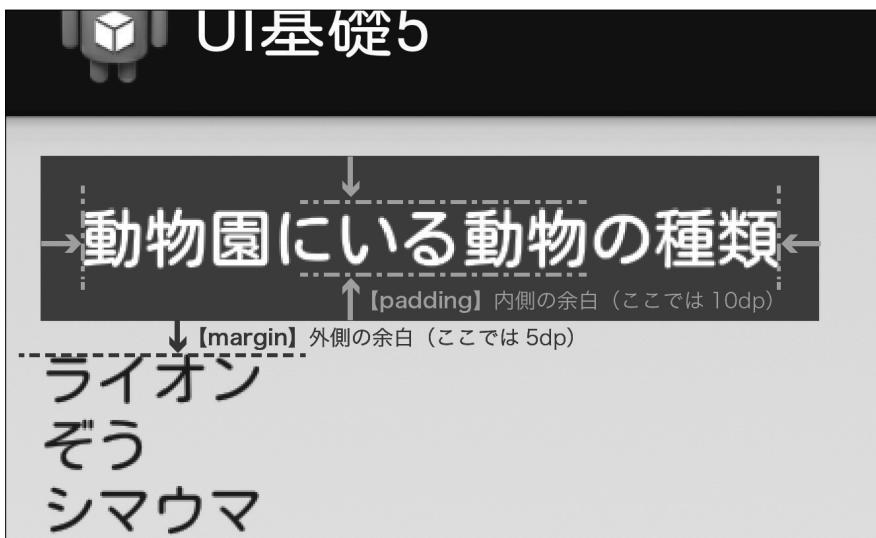


図16:見出しを元にしたときの、marginとpaddingの違い

背景色がついていないものについては、marginにしてもpaddingにしても結果は変わらない場合があります。

しかし、背景色があるものについては、背景色の部分が領域の境界となりますので、marginとpaddingは正しく使い分けるようにしましょう。

5 應用

今回作成したサンプルはコンテンツの量が多く、しかもスクロールができないために下のほうが見切れてしまいました。全体をスクロールさせるためには、いちばん大きな親のビューとして、ScrollViewを追加し、すでにあるLinearLayoutを少し修正します。

```
activity_main.xml
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="vertical">
        ~中略~
    </LinearLayout>
</ScrollView>
```

ScrollViewを追加し、LinearLayoutの「`android:layout_height`」を「`wrap_content`」に変更した結果、全体が縦方向にスクロールするようになりました。図17の右端の細い縦棒のようなものは、スクロールバーです。スクロールバーは、スクロールしているときのみ出現します。

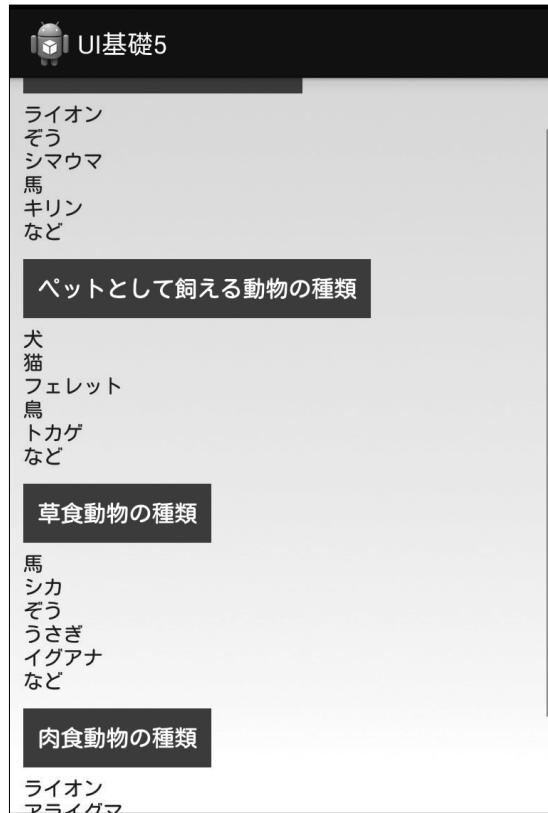


図17:全体がスクロールする

6 colors.xmlに色を定義する

ここまでがサンプルファイルの作成でした。ここからはcolors.xmlを作成していきます。「values」フォルダー内に「colors.xml」を作成し、次のように書いてください。

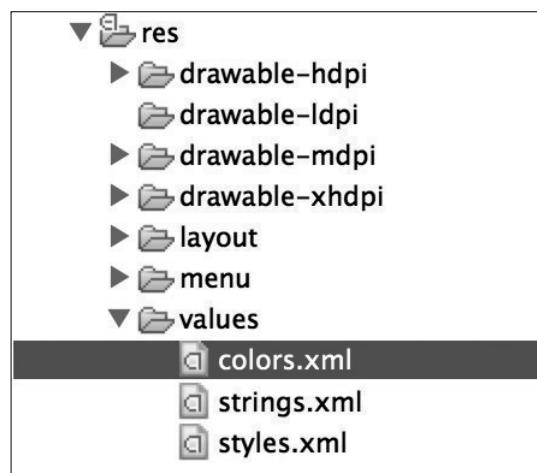


図18:valuesフォルダーにcolors.xmlを作成

values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="keycolor">#b81c22</color>
    <color name="keycolor_text">#fff</color>
</resources>
```

「keycolor」という名前で「#b81c22」(濃い赤)を定義しました。また、「keycolor_text」という名前で「#fff」(白)を定義しました。レイアウトXMLもこれにあわせて変更します。

activity_main.xmlの見出しの部分、「android:background」と「android:textColor」の指定を、以下のソースコードのように変更します(4カ所)。

activity_main.xml

```
android:background="@color/keycolor"
android:textColor="@color/keycolor_text"
```

colors.xmlのkeycodeという名前の色を指定するように変更しました。これにより、4カ所あった見出し部分の背景色(キーカラー)と文字色を、colors.xmlで指定して一元管理できるようになりました。

colors.xmlを使うメリットも、strings.xmlを使うときと同じで、レイアウトXMLやJavaを編集することなく、効率よく修正ができることがあります。

修正が発生した際に、数十カ所の値を直さなければならないのと、1カ所のみの修正でよいことを比べると、圧倒的に後者のほうが効率がよいでしょう。

今のうちからそういったことに気をつけておくと、複雑なソースコードになったときでも、ミスなく対応することができるようになります。



8-5-4 styles.xmlで、複数の装飾を一括管理

「styles.xml」は、ビューの属性を一括管理できるファイルです。colors.xmlで作成したサンプルファイルをそのまま使って見ていきましょう。

今、レイアウトXMLの見出しと本文は、見出しの装飾も含めて以下のようなソースコードになっています。

```
activity_main.xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textStyle="bold"
    android:textColor="@color/keycolor_text"
    android:background="@color/keycolor"
    android:padding="10dp"
    android:layout_marginBottom="5dp"
    android:text="@string/animal" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    android:text="@string/animal_text" />
```

しかしこれはほんの一部であり、同じようなコードが4回繰り返されるのです。

色についてはcolors.xmlを使って一元管理しましたが、その他の部分に少しでも修正が入ったら……4カ所について同じように修正をしていかなければならず、ミスが発生しやすくなってしまいます。

こういった、同じ装飾を何度も繰り返す必要がある場合には、styles.xmlが便利です。

styles.xmlには、すでにいくつかの指定が書かれていますが、それはそのままにしておいて、追記してきます。

まずはstyles.xmlに<style name="head">～</style>を追記します。名前は自分で決めることができます。

```
values/styles.xml
<resources>
    ~中略~
    <style name="head">
        </style>
</resources>
```

それができたら、activity_main.xmlに以下のコードを追記します。

activity_main.xml

```
<TextView
    ~中略~
    style="@style/head" />
```

styles.xmlで作成した名前を、レイアウトXMLで参照します。書き方はstrings.xmlやcolors.xmlのときと同じです。

ここまで準備完了です。次は、指定してある属性を、ひとつずつそのままstyles.xmlに移していくますが、書き方が少し異なりますので、間違えないように注意してください。

activity_main.xml に書かれているコード

```
android:layout_width="wrap_content"
```

styles.xml に書くコード

```
<item name="android:layout_width">wrap_content</item>
```

activity_main.xmlでは上のように書かれているコードは、styles.xmlに書くときは下のコードのようになります。

他の属性についても、共通するものはすべてまとめてしまいましょう。この結果、以下の2つのコードのようになります。

activity_main.xml

```
<TextView
    style="@style/head"
    android:text="@string/animal" />
```

values/styles.xml

```
<resources>
    ~中略~
    <style name="head">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textSize">16sp</item>
        <item name="android:textStyle">bold</item>
        <item name="android:textColor">@color/keycolor_text</item>
        <item name="android:background">@color/keycolor</item>
        <item name="android:padding">10dp</item>
        <item name="android:layout_marginBottom">5dp</item>
    </style>
</resources>
```

styleを参照するときは、「`android:`」という接頭辞はつかないので注意しましょう。

同じく、本文についてもまとめられる属性はまとめておきましょう。

自分で完成することができたら、完成サンプルのソースコードも確認してましょう。



演習問題

今までの内容の復習として次のレイアウトを作成してみましょう。

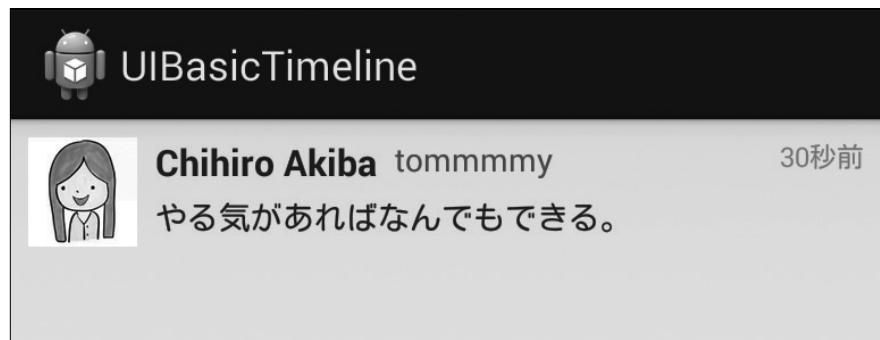


図19:完成サンプル



図20:ヒント

演習用に、まず「UIBasicTimeline」という新規プロジェクトを作成します。



図21: UIBasicTimelineという新規プロジェクトを作成する

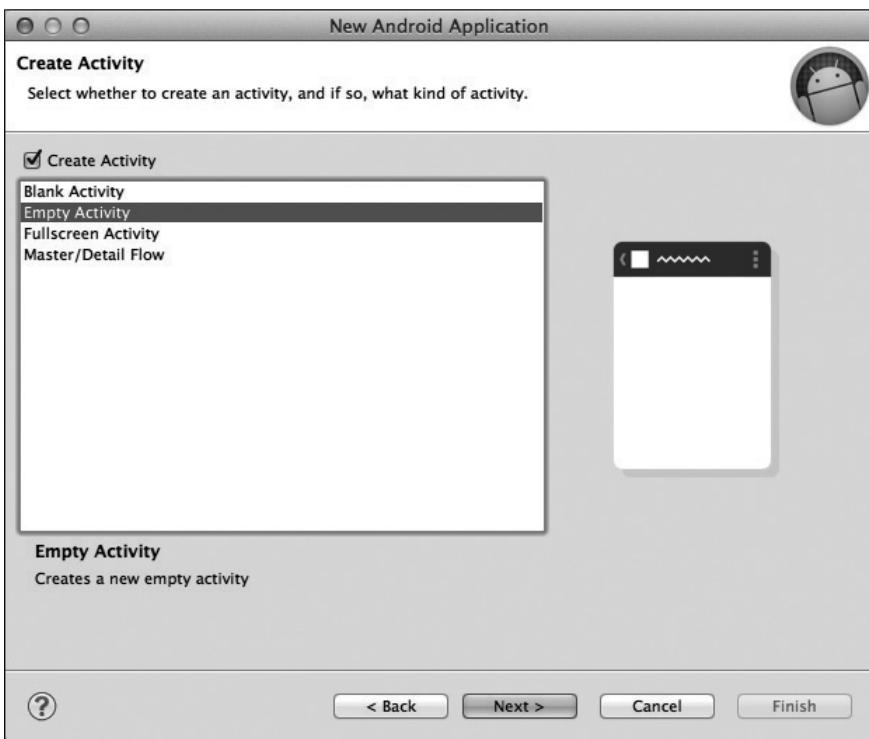


図22: Empty Activityを選択する

タイムライン用に使う画像を、resフォルダー内の「drawable-xhdpi」フォルダーにコピーします。画像は、自分の好きなものを使ってください。
ここまでできたら、ボタンやテキスト、画像を配置してレイアウトを作成していきましょう。必要に応じて、colors.xmlやstrings.xml、styles.xmlも使うようにしましょう。