Android アプリ設計パターン入門

日高正博 @mhidaka
小西裕介 @konifar
藤原聖 @satorufujiwara
吉岡毅 @tsuyogoro
今井智章 @tomoaki_imai

PEAKS

はじめに

本書はモバイル分野のAndroidアプリケーション設計手法を解説する入門書です。

設計手法の選択は、とても難しい判断であり正解はやってみないとわからないものです。適切な設計手法を選択するために知識や経験が問われます。

バランスの問題と言い換えてもいいかもしれません。アプリの要件や開発プロセスなどプロジェクトの背景や開発者の技術的嗜好およびスキルセットなど技術的背景によってベストな手法が変化していくでしょうし、新規開発では大胆にチャレンジできても機能追加を繰り返す継続的な開発では地に足をつけた方法を選ぶことがあります。

開発者はこのような背景を抱えながらも、設計を通じてコミュニケーションを図り、今後の変更に柔軟に対応できて不具合が出にくい強固なソフトウェアを目指します。近年ではスマートフォンが普及した結果、利用シーンも拡大し続けています。アプリの果たす役割はより大きくなり、求められる設計も変化していくでしょう。

本書は設計手法をすべて収集して解説するといった図録的な役割は持ち合わせていません。設計について議論できる土台を作ることを目指しています。前半では設計手法の選択に役立つようにアプリの設計を評価して利点や欠点を率直にまとめています。後半ではプロジェクトの背景や技術的背景を知り、どのようなケースで有用なのかという実例を収録します。

これらの事例は自分たちのアプリ開発で、どの設計がベストなのか疑問を話し合う土台として有効に機能するはずです。自分たちのプロジェクトを改善するための観点や見えていなかった課題の気付きといった形で新しい視点を獲得することが本書の役割です。収録内容は、アプリ開発のプロフェッショナルが四苦八苦しながら形にした知識、知見です。役に立つに違いありません。

さきほどプロフェッショナルと表現しましたが、彼らが最初からアプリ開発に精通していたわけではありません。視点を変えれば、いたって普通の開発者です。モバイルアプリのアーキテクチャは移り変わりが激しく、常に試行錯誤しています。

現時点でよいものでも将来に渡って性能を維持できるかは保証されておらず、業界の未来を見通せる預言者は現れていません。間違えたことをやっていないだろうか、正解に近づいているのかという不安が常に開発者につきまといます。開発者の多くが設計上の悩みをもっています。本書を元に議論し、設計に関して多くの視点を獲得してください。設計手法のよりよい選択に繋がると信じています。

著者代表 @mhidaka

サンプルコード

次のリポジトリに本書のサンプルコードが掲載されています。

- https://github.com/TechBooster/Architecture-Patterns-Samples

各章でサンプルコードの指定がある場合は、そちらも合わせて確認ください。

クラウドファンディングとPEAKS

本書は技術書クラウドファンディング・サービスである「PEAKS」のプロジェクトとして開始され、630人の支援者のサポートによって作られました。出資者特典である「アーリーアクセス」でいただいたご意見も反映されております。

PEAKSではこんな本を作りたい！という方を募集しています。次の窓口からご連絡いただければ幸いです。

- https://peaks.cc/requests

TechBoosterとは

TechBoosterはAndroidをはじめとしたモバイルのための技術サークル[注1)]です。オープンソースへの貢献や社会還元を目的にウェブサイトや書籍でモバイル技術を解説しています。次のウェブサイトで刊行物一覧を確認できます。

- https://techbooster.booth.pm

お問い合わせ先

本書に関するお問い合わせ：support@techbooster.zendesk.com

注1) TechBoosterのWebサイト http://techbooster.org/

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

第1部
アプリの設計を知る

第 1 章

Androidアプリの基本構成

日高正博／ @mhidaka

ソフトウェア設計は課題を解決して目的を達成する手段といえます。アプリケーションの設計には答えはありませんが、設計手法を知ることは、よりよい開発の一助となります。ここではAndroidプラットフォームの特徴、特に設計上の課題となり得る部分を説明していきます。Androidアプリ開発に熟練した開発者であれば読み飛ばしても構いません。

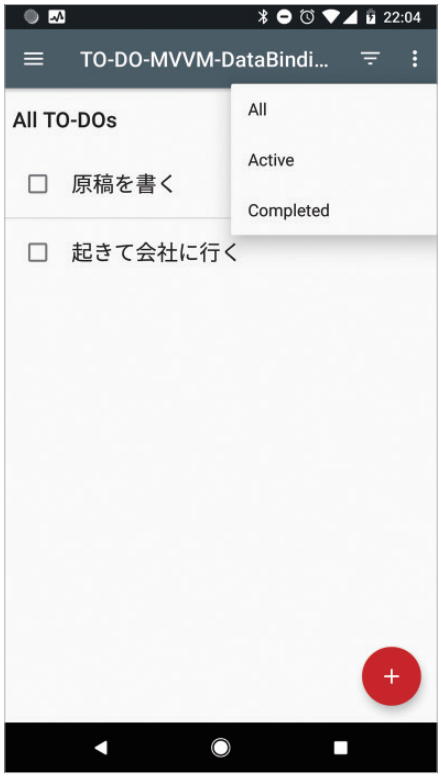
- アプリの基本構成（MVP、MVVM）について学ぶ
- プラットフォームの制約と設計の歩み
- ライブラリの役割、アーキテクチャに与える影響

1.1 議論の前提となるアプリケーションと仕様

本書の第1部では、特定のアプリケーションをターゲットに議論を進めます。アプリの基本構成について共通の知識となるアーキテクチャパターンをまとめます。題材としてGoogleの提供するAndroid Architectureリポジトリを選択しています。

- https://github.com/googlesamples/android-architecture

アプリケーションの仕様を統一することで各アーキテクチャの違いを明瞭にする狙いがあります。アプリケーションはごく簡単なTODOアプリです。シンプルな仕様なので迷うことは少ないでしょう（図1.1）。



● 図 1.1　TODO アプリのスクリーンショット

ユーザーはTODOを追加し、編集、削除が可能です。ローカルストアへの保存のほかサーバサイドとの通信も考慮します。TODOアプリは

- トップ画面
- 詳細画面
- 統計画面
- 追加画面
- 編集画面

の5つをもっています。

設計の良し悪しに絶対的な指標はありませんが、開発対象のターゲットを統一することで各アーキテクチャパターンがとるアプローチの違いを理解しやすくなります。自分の手に馴染んだ道具として利点を把握し、使いこなせるようになることを目指します。

第 2 章

MVVMパターンを使ったアプリ構成

日高正博／ @mhidaka

本章ではAndroid Architecture Blueprintsのアーキテクチャ実装をベースにMVVMパターンを解説します。

- https://github.com/googlesamples/android-architecture

Androidフレームワークは、Androidアプリの作成および設計方法を柔軟に決定できます。自由度の高さは尊重されるべきですが、時として巨大なクラス、一貫性のないネーミングルールの出現、アプリ内でのアーキテクチャ不一致という歓迎せざる状態に繋がります。

このような状態になればテスタビリティの劣化はもちろん継続的な開発、保守、メンテナンスを困難にします。Android Architecture Blueprintsは、これらの問題を解決または回避するための戦略について考えていくプロジェクトです。

2.1 基本コンセプト

Android Architecture Blueprintsは同じ仕様のアプリを異なるアーキテクチャ、ライブラリを使用して実装するサンプル集です。学習のためのリファレンス・議論の出発点、自分のアプリの参考としてください。

MVVMやMVPなどのパターンをさまざまな方法を使用してアプリを構築していますが、標準的な実装サンプルというわけではありません（もちろん参考になります）。

開発者ごとにアプリに要求する仕様、コンセプトの優先順位は異なるため、そのまま適用してもうまくいくわけではないためです。

大事な点はコードを構造化する方法、アーキテクチャを活用した設計手法、またこれらのアーキテクチャを使ってアプリを継続的に開発する際の影響を確認することです。

サンプルのTODOアプリはシンプルなUIを採用しています。初見でも簡単に理解できるほどシンプルであることを目指していますが、設計の違いやメリット・デメリットを紹介するには十分複雑なテーマです。

2.2 セットアップ

Android Architecture Blueprintsリポジトリでは、ブランチごとに異なるアーキテクチャが置かれています。

データバインディングをつかったMVVMパターンでの実装例を見たい場合、次のコマンドで取得してください。

```
git clone git@github.com:googlesamples/android-architecture.git
git checkout todo-mvvm-databinding
```

2.3 サンプルアプリの設計

サンプルアプリの設計を乱暴にいってしまうとMVVMアーキテクチャのViewModelは、MVPアーキテクチャのPresenterと同様の役割を果たします。2つのアーキテクチャはViewがViewModelまたはPresenterと、それぞれ通信する方法が異なります。

アプリケーションがMVVMアーキテクチャのViewModelを変更すると、ビューはライブラリまたはフレームワークによって自動的に更新されます。ViewModelはViewへのアクセスに必要な参照を持たないためViewModelから直接Viewを更新できません（自動的な更新に任せます）。

MVPアーキテクチャではPresenterからViewを更新できます。Viewに必要な参照があるためです。変更が必要な場合は、Presenterから明示的にViewを呼び出して更新できます。

MVVMパターンを利用する場合、レイアウトファイルを使用してViewModelから観測可能なフィールドをTextViewやImageViewなどの特定のUI要素にバインドします。データバインディングライブラリは、次の図に示すようにViewとViewModelが双方向で同期することを保証します（図2.1）。

第 3 章

MVPパターンを使ったアプリ構成

日高正博／ @mhidaka

本章ではAndroid Architecture Blueprintsのアーキテクチャ実装をベースにMVPパターンを解説します。

- https://github.com/googlesamples/android-architecture

Androidフレームワークは、Androidアプリの作成および設計方法を柔軟に決定できます。自由度の高さは尊重されるべきですが、時として巨大なクラス、一貫性のないネーミングルールの出現、アプリ内でのアーキテクチャ不一致という歓迎せざる状態に繋がります。

このような状態になればテスタビリティの劣化はもちろん継続的な開発、保守、メンテナンスを困難にします。Android Architecture Blueprintsは、これらの問題を解決または回避するための戦略について考えていくプロジェクトです。

本章の「基本コンセプト」の内容は第2章 MVVMパターンを使ったアプリ構成と重複しています。独立して読んでも意味が伝わるように書籍を構成している都合の措置です。すでに読んだ部分については読み飛ばしてください。

3.1 基本コンセプト

Android Architecture Blueprintsは同じ仕様のアプリを異なるアーキテクチャ、ライブラリを使用して実装するサンプル集です。学習のためのリファレンス・議論の出発点、自分のアプリの参考として使ってください。

MVVMやMVPなどのさまざまなパターンを使用してアプリを構築していますが、標準的な実装サンプルというわけではありません（もちろん参考になります）。

開発者ごとにアプリに要求する仕様、コンセプトの優先順位は異なるため、そのまま適用してもうまくいくわけではないからです。

大事な点はコードを構造化する方法、アーキテクチャを活用した設計手法、またこれらのアーキテクチャを使ってアプリを継続的に開発する際の影響を確認することです。

サンプルのTODOアプリはシンプルなUIを採用しています。初見でも簡単に理解できるほどシンプルであることを目指していますが、設計の違いやメリット・デメリットを紹介するには十分複雑なテーマです。

3.2 セットアップ

Android Architecture Blueprintsリポジトリでは、ブランチごとに異なるアーキテクチャが置かれています。

MVPパターンでの実装例を見たい場合、次のコマンドで取得してください。

```
git clone git@github.com:googlesamples/android-architecture.git
git checkout todo-mvp
```

3.3 サンプルアプリの設計

これからMVPパターンを適用したTODOアプリを紹介します。Android Architecture BlueprintsではMVPパターンによる設計（todo-mvpブランチ）を基準として他の設計手法と比較対照しています。

MVPパターンを知ることはAndroid Architecture Blueprintsを読み進める基礎づくりともいえます。サンプルアプリでは次の点に注意してください。

- アーキテクチャのフレームワークを使わず、基本的なMVPパターンでの設計

MVPパターンに有用なライブラリとしては有名なところでDaggerやRxJava 2がありますし、MVPを発展させたClean Architectureなど今回紹介するものより実用的な派生・実装例があります。他方で理解のためにライブラリの知識やアーキテクチャに関する知識が問われます。MVPパターンのコンセプトを理解する目的であればシンプルなMVPの実装を見るほうがよいでしょう。

サンプルアプリでは次のような特徴があります

- コールバックを利用して非同期タスクを処理しています
- データはRoomを使用して単純なSQLiteデータベースとしてローカル保存

次の図はサンプルアプリでのMVPパターンをあらわしたものです（図3.1）。

第2部

生きた設計を見る

第 4 章

差分開発にみる設計アプローチ

吉岡毅／@tsuyoyo

フリマアプリ「メルカリ」は、2013年7月にリリースされて以来、多くのユーザーを獲得し、2017年12月時点で日本国内（JP）のダウンロード数が6000万件を超えました。2014年に米国（US）、2017年に英国（UK）でもアプリをローンチしており、4年という期間をかけて拡大してきた大規模サービスのひとつです。

- https://www.mercari.com/jp/

サービスを支えるクライアントアプリであるAndroid版メルカリアプリは、4年間でさまざまな変化を遂げてきました。

本章では、4年という期間をかけて積み重ねてきたコードにどんなことが起きて、我々がそれらに対してどのように向き合い、アクションを取ってきたのかということを紹介します。紹介を通じて、本章の目的である「差分開発の中で生まれてしまうレガシーなコードへの、改善アプローチの一例」を伝えられればと考えています。

本章で触れる内容は教科書ではなく「今生きている、ナマモノ」をお届けするというスタンスで書いています。読者の皆さんの何かしらの気付きになれば幸いです。

なお、ご存知の読者もいるかもしれませんが、メルカリのAndroidアプリではJP、US、UK、3つのコードベースが存在しています。本章で取り扱うコードベースは、創業時から育て上げてきたもので、現在ではJPのコードベースになっているものになります。

4.1 差分開発ってなんだろう

タイトルでも取り上げている「差分開発」という言葉は、単純に「オリジナルのコードベースに変更や追加を行いアプリを育てていくこと」と解釈します。重要なことは「既存のアプリが持っていた機能が正しく動作することをキープしつつ、新規機能を追加していくこと」です。

ただし、この差分開発においてエンジニアが時々のベストを尽くしたつもりになっていても、機能を成長させようとした時に、なぜか「あれ、この機能をここに追加しようとするとツラいな…」という状況に直面することが多いように感じます。

プロジェクトのスケジュールに追われるあまり、その場しのぎ的な変更を入れ、機能としては無事リリースするものの、技術的な負債を残してしまうケースや、適当なタイミングでリファクタリ

ングをしたいけれどビジネスを成長させる方にプライオリティが置かれ、結局その負債は放置せざるを得なくなってしまうケースといった厳しいサイクルが回ってしまう訳です。

サービスが成功してビジネスが大きくなっていけば、それに越したことはありません。ただビジネスが大きくなっていくと次に待ち受けてくるのは「組織・チームの拡大」です。そうなった時に、積み上げてきた負債たちは急に牙を剥き始めます。チームの拡大を阻止するかの如く、困難な問題を突き付けてくるのです。

メルカリはまさに今そのフェーズの真っ只中です。困難な問題たちと上手く付き合いながらチームの拡大を徐々に成し遂げていると筆者は感じていますし、改善を積み重ねつつ新しいことにチャレンジしています。

4.2 開発初期から継ぎ足してきた秘伝のタレ

開発初期から積み重なってきた「負債」による問題と戦う日々ではありますが、負債が具体的にどんな問題を引き起こしているのか、包み隠さず紹介していきます。

筆者はアプリの立ち上がりから、およそ2年半のタイミングでチームに加わっています。ちょうど今のタイミングから見るとローンチから真ん中くらいの時期です。ある程度の負債が存在する状態からスタートし、自分でも負債を作りましたし、負債の解消にも取り組んできました。

一点、誤解が無いようにしたいのですが、ここでいう「負債」は筆者自身あまりネガティブなものとは考えていません。というのも、これらは「少しでも早くお客さまにワクワクできる体験を届け続け、しかもきちんと動作するものをお届けする」ことを達成し続けてきた結果であり、その瞬間瞬間ではベストの選択であったと考えているからです。ここから先は「負債」ではなく、あえて「秘伝のタレ」と呼んでいきます。

ただし事実として秘伝のタレたちは「困ったこと」を引き起こしています。困ったことを引き起こすプラクティスってこういうことがあるのか、という気付きになれば幸いです。また併せて「こうすれば改善できるはず or 改善した」という見解も書いていきます。

4.2.1 超多段継承地獄

Androidフレームワークでは Java 言語が使われており、Java 自身が継承を言語仕様で持つこともあり、責務を分散させるためにクラスの親子関係を構築するプログラミング上の工夫は、Android 開発でも定石でしょう。ところが、継承関係は時に余計な複雑さを生み出すことがあります。

実は、メルカリアプリの中に多く存在しているリスト画面には裏に6段の Fragment 継承が存在し、出品画面を表示する機能をもつ画面は4段の Activity 継承が必須となっています。たとえばリスト画面の継承関係を例に挙げると、

第 5 章

OSSにおける設計者の役割

小西裕介／ @konifar

国内の大規模なAndroidカンファレンスのDroidKaigiでは、公式アプリのリポジトリを公開して**OSS（オープンソースプロジェクト）**として皆で作り上げるという取り組みを行っています。

- https://github.com/DroidKaigi/conference-app-2017

DroidKaigi 2017では、リポジトリ公開後に国内外問わず60人以上のエンジニアから250を超えるPull Requestが集まり、大きな盛り上がりを見せました。

筆者はOSSのリポジトリオーナーとして携わり、アプリの大枠作成からマイルストーンの設定、Issueの整理、Pull Requestのレビューなどを行ってきました。

本章では、スキルやバックグラウンドが違う多数のエンジニアが関わるOSSにおいて、筆者が当時何を考えて初期のアプリの骨子を作ったのか、振り返ってみてどんな反省点があったのかを実際のコードをまじえながら説明します。仕上がった設計のコードだけ見ても「なぜ」こんな風に作ったのかはわかりません。設計の意図を読み解く助けになれば幸いです。

5.1 公式アプリの概要

はじめに、**公式アプリ（DroidKaigi 2017公式アプリ）**で何ができるかを簡単に説明します。公式アプリは次の4つの機能に分かれており、Bottom Navigationでそれぞれの画面を遷移できます。

- セッション
- 地図
- 情報
- 設定

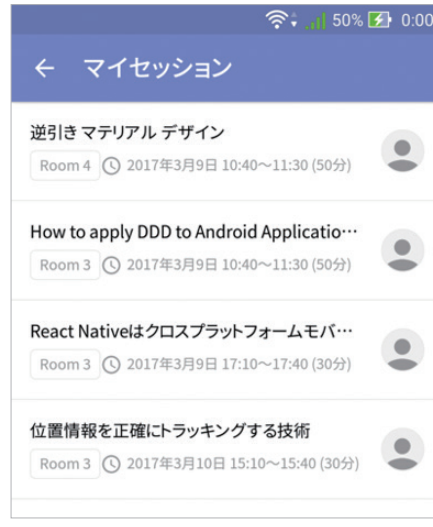
5.1.1 セッション

カンファレンス開催期間である2日間分の全67セッションを確認できます。タイムテーブル兼セッション一覧（図5.1）は番組表のようになっていて、タップすると詳細ページ（図5.2）が開きます。

● 図 5.1　セッション一覧

● 図 5.2　セッション詳細

セッション機能では参加予定のセッションを登録でき、その一覧はマイセッションページ（図5.3）で確認できます。



● 図 5.3　マイセッション

またセッションの内容やスピーカーの名前でセッションを検索（図5.4）できます。

● 図 5.4　セッション検索

第７章

チームとアーキテクチャ

今井智章／@tomoima525

フリマアプリ「メルカリ」のUS版は2017年3月にフルスクラッチでアプリをリニューアルしました。このリニューアルプロジェクトでは、既存のコードで発生していた多くの課題を３ヶ月で解決するというチャレンジに、チーム一丸となって挑みました。本章では、チームでどのような観点から検討し設計に落とし込んだか、またこの設計がチームにどのような影響を与えたか説明していきます。

7.1 大胆に機能追加、変更ができるようにアプリを作り直す

最初にUS版メルカリをフルスクラッチを書き直すまでの経緯について触れておきましょう。2013年7月、メルカリは日本版をリリースし、2014年9月にUS版をリリースしました。当時のUS版メルカリは日本版メルカリと同一レポジトリで管理していました。

US版と日本版で異なる決済や配送方法などの機能はGradleのFlavor機能を使ってコードを切替えられるようにし、多くの部分で共通化を図っていました。この戦略は、US版メルカリをスピーディに立ち上げ、運用することを狙いとしています。実際にUS版メルカリは著者も含めた２人のAndroidエンジニアにより１ヶ月半という短期間で開発、リリースされています。

US版リリース後から２年も経過すると、さまざまな問題が表出してきました。一番の大きな問題は、US版と日本版の機能の乖離で、UXにまで影響を及ぼしていました。たとえば出品機能は配送方法だけでなく出品フローも異なります。それらのロジックを同一クラス上でUS向け、日本向けで切替えていたため、USでの変更が日本のアプリに思わぬ影響を及ぼすことがありました。コードレビューや検証に大幅な時間が要求されるようになり、大胆な機能変更やスピード感のあるリリースが次第に難しくなってきていました。

そのような状況を踏まえ、現行アプリと同等な機能を持ちつつ、新機能もスピーディにリリースできる新しいUS版メルカリを開発するプロジェクトが発足しました。レガシーなコードベースを刷新するために設計も実装もすべてフルスクラッチで書き直すことになりました。

書き直すにあたって筆者のチームでは大きく２つの観点から設計を検討しました。

- 既存の開発におけるペインポイントを解決する
- 3年間運用されたアプリを３ヶ月で書き直す

それぞれに関して課題と結果としてどのような設計方針が定まったか説明していきます。

Column. Mercari Plus

余談ですが、当初はUS版メルカリアプリと新US版メルカリアプリのABテストを行うために別名アプリ（Mercari Plus）として一部のユーザーに向けてリリースしていました。パッケージ名はチームのQAエンジニアがラーメン好きだったことから`com.mercariapp.ramen`（メルカリアプリは`com.mercariapp.mercari`）という名前でした。やがてMercari Plusの方がユーザーのよい反応が得られることがわかったため、現行のアプリをリプレイスする運びとなりました。

7.2 既存の開発におけるペインポイントを解決する

フルスクラッチでアプリを作り直す意義は、既存の課題を小手先ではなく解決することにあります。一番の課題として「US向けと日本向けのロジックが混在している」と前述しましたが、それ以外にも多くの課題がありました。主だったものを列挙しましょう。

ビューとビジネスロジックの密結合とテストの不在

いくつかのActivityはビジネスロジックがそのまま書かれた部分が多くありました。ビュー周りのロジックはある程度移譲しているのにもかかわらず、2000行を超え、テストを書くこともままならない状況でした。リリース時より3年間突っ走って価値を生み出してきたコードは偉大ですが、偉大すぎて触ることも恐れ多い存在となっていました

適切なモジュール化がなされていない

たとえばAPIリクエストを行うためのクラスはシングルトンにstaticメソッドを備えていました。これはあらゆるクラスから容易に呼べる便利さもある反面、依存性の問題を発生させます。テストが書けなくなることも問題ですが、その機能を他のモジュールに切り替えることも困難になります

ライフサイクルにおけるインスタンスステート管理

読者もご存知のとおり、画面回転や何らかの設定変化が発生した時にAndroidはActivity上の情報が破棄されます。画面を再生成する際に再度のAPIリクエストを避けるため、データをシリアライズして`onSavedInstanceState`で保管する実装になっていました。このような作りはデータ保存、復元ロジックが冗長になりがちですし、実際に状態管理ロジックが複雑になっていました

第3部
設計を考える

第 8 章

Android Architecture Components

藤原聖／@satorufujiwara

Android Architecture Components はGoogleから提供されているライブラリ群です。次のURLがリファレンスになります。

- https://developer.android.com/topic/libraries/architecture/index.html

このライブラリ群は名前のとおり、アプリケーションのアーキテクチャにとても影響を与えるライブラリです。リファレンス内「Recommended app architecture」の項目にて、これらのライブラリを使ったお勧めの設計が提案されています。本章ではドキュメントを参考にしながらArchitecture Componentsを使ったアプリケーション設計について考えてみます。

8.1 Android Architecture Componentsとは

Android Architecture Components はGoogle I/O 2017でα版が公開され、2017年11月6日に安定版となる1.0.0が公開されました。コンセプトは「堅牢で、テストがしやすく、保守しやすいアプリ」を作るためのライブラリ群です。次の5つのモジュールを公開しており、好きなように組み合わせて使えます。

- Lifecycles
- ViewModel
- LiveData
- Room
- Paging

これらのライブラリは2017年に登場したGoogle Maven Repository[注1)]にて公開されており、プロジェクトの`build.gradle`に次のように記述することでそれぞれのライブラリを導入できます（リスト8.1）。

注 1）https://maven.google.com

● リスト 8.1　build.gradle

```
repositories {
    jcenter()
    google() // Google Maven Repository を追加
}

dependencies {
    // Lifecycles のみ (ViewModel や LiveData が不要の場合 )
    implementation "android.arch.lifecycle:runtime:1.0.3"
    annotationProcessor "android.arch.lifecycle:compiler:1.0.0"

    // ViewModel と LiveData
    implementation "android.arch.lifecycle:extensions:1.0.0"
    annotationProcessor "android.arch.lifecycle:compiler:1.0.0"

    // Room
    implementation "android.arch.persistence.room:runtime:1.0.0"
    annotationProcessor "android.arch.persistence.room:compiler:1.0.0"

    // Paging
    implementation "android.arch.paging:runtime:1.0.0-alpha4-1"
}
```

Paging ライブラリのみ2017年12月現在、αバージョンが最新となっています。まだまだ安定していないため導入の際は注意が必要です。

中心となるライブラリ以外にも、テスト用やRxJavaと連携するためのユーティリティライブラリなどが公開されており、必要にあわせて適宜導入することになります（リスト8.2)。

● リスト 8.2　Architecture Components のユーティリティライブラリ

```
dependencies {
    // LiveData 用のテストヘルパー
    testImplementation "android.arch.core:core-testing:1.0.0"

    // Room 用のテストヘルパー
    testImplementation "android.arch.persistence.room:testing:1.0.0"

    // Room の RxJava サポート
    implementation "android.arch.persistence.room:rxjava2:1.0.0"

    // LiveData の ReactiveStreams および RxJava サポート
    implementation "android.arch.lifecycle:reactivestreams:1.0.0"
}
```

ご存知のとおり、AndroidにおいてはActivityやFragmentが複雑なライフサイクルを持っています。画面回転時などにActivityが再生成され、それに対応するための処理を書いたことあるAndroid開発者も多いことでしょう。

Android Architecture Componentsは複雑なライフサイクルの扱いを簡単にし、Androidアプリのアーキテクチャ設計の助けとなるライブラリです。ライフサイクルの状態やイベントの情報を得ることができるLifecyclesというライブラリが中心となります。またViewModelは画面回転時のActivityの再生成を乗り越えて生存するコンポーネントで、UIに使う状態やデータを保持するクラスとして適しています。LiveDataはViewModelからのデータをActivityやFragmentが受け取るために使われます。LiveDataのおかげでActivityやFragmentの複雑なライフサイクルへの対応が簡単になります。次節以降ではLifecycles、ViewModel、LiveData、これら3つのライブラリを使ったAndroidのアーキテクチャ設計について解説します。

本書では使いませんが上記以外のライブラリであるRoom、Pagingは次のような機能があります。

Room

データ永続化用のライブラリで、SQLiteデータベースへアクセスするための抽象的なレイヤーとして、SQLiteをより使いやすくなるようなAPI群が提供されている。今後SQLiteを使う場合はRoomの使用を推奨している

Paging

データベースやWeb APIなどからデータを何回かに分けて読み込む際に使うことができるライブラリ。RecyclerViewなどのリスト表示用UIコンポーネントと組み合わせ使うことが想定されている

8.2 Architecture Componentsの中心：Lifecyclesコンポーネント

Android Architecture Componentsの中心となるのはLifecyclesコンポーネントです。ActivityやFragmentといったライフサイクルをもつコンポーネントのライフサイクルの状態やイベントの情報を得ることができるライブラリです。次の2つのenumクラスが用意されています。

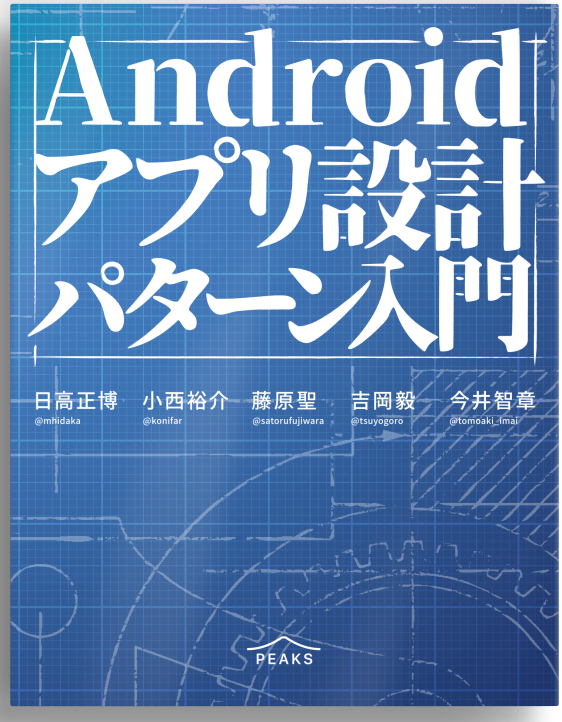
`Lifecycle.Event`

ActivityやFragmentなどのライフサイクルに応じて送信されるイベント

`Lifecycle.State`

`Lifecycle.Event`に応じた、現状のActivityやFragmentの状態

この続きを読むには...



Android アプリ設計パターン入門

日高 正博 / 小西 裕介 / 藤原 聖 / 吉岡 毅 / 今井 智章

電子版：¥2,800　製本版：¥3,200

購入して続きを読む 〉

PEAKS