**MVVMとMVP**

**MVVMとは**

ユーザが入力した情報と格納したデータをデータバインディングにて自動的に処理する設計

もともと、WPF（Windows Presentation Foundation）などで生まれた考え方で、現在はAndroidのプラットフォームやWebブラウザ上でのJavaScriptのフレームワーク（Vue.js等）で採用されている。

・Model…アプリケーションが扱うデータの処理とデータの変更をViewに通知

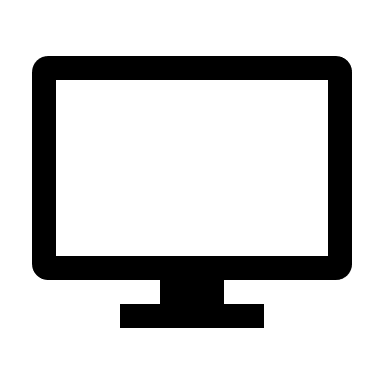
・View…ユーザへの出力処理

・ViewModel…ModelとViewを紐づける

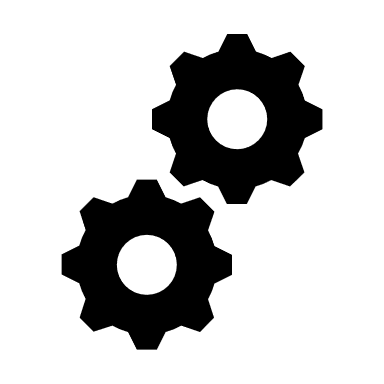
・データバインディング…ViewとViewModelを結びつける仕組み、ViewとViewModelのどちらかの値が書き変われ 　 ば、両方の値が変更される。（相互でデータの更新をやり取りしている）



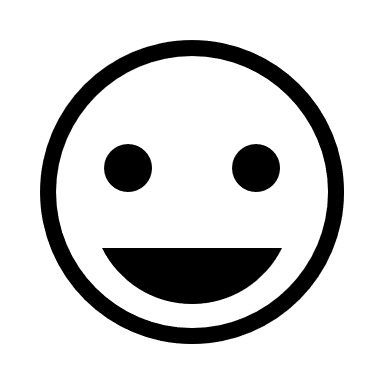
Model



View



ViewModel



ユーザ

データバインディング

画面表示

ユーザアクション

データ処理

データ取得

**MVVMのメリットとデメリット**

**メリット**

・MVCに比べてユーザが入力したものを素早くデータを書き換えられる。

**デメリット**

・データバインディングをすることが前提なので、それに対応していないフレームワークではMVVMを採用できない。

**MVPとは**

異なる複数のViewを同時にサポートする設計。

GoogleがMVPベースのサンプルアプリを多く提供し簡単にAndroidアプリ開発を簡単に始められるため、モバイル開発の現場ではMVPを採用することが多い。

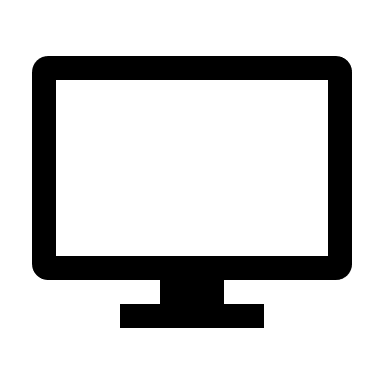
・Model…アプリケーションが扱うデータの処理とデータの変更をViewに通知

・View…ユーザへの出力処理

・Presenter…データ変更の通知をとりまとめViewに通知する

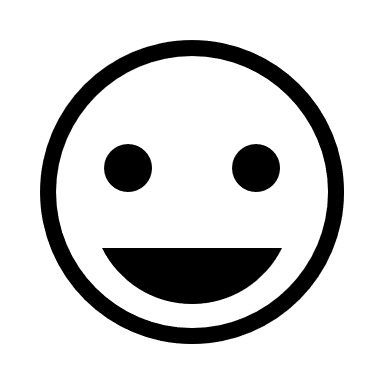


Model



View

Presenter



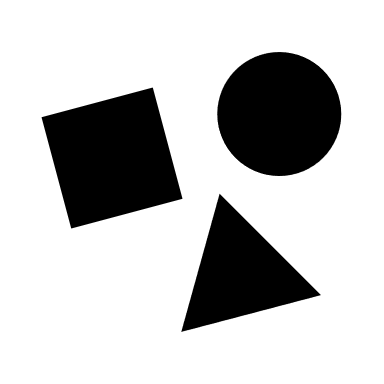
ユーザ

画面表示

ユーザアクション

データ処理

データ取得



処理後データの反映

ユーザアクションの伝達

**MVPのメリットとデメリット**

**メリット**

・ModelとViewの独立性が高いため各モジュールに及ぼす影響最小限にでき、変更に強い。

・ビジネスロジックに関わる機能をPresenter側に集約したことでModel単体でテストしやすくなり、Viewの独立性も高いためView単体のテストもしやすくなった。

**デメリット**

・Presenterの機能を細かく分離しないとテストが複雑になる。

・規模が大きくなるとModel、View、Presenterの組み合わせの数が膨大になってしまう。