

How to work as a Team

自己紹介

- Name : **azu**
- Twitter : [@azu_re](#)
- Website: [Web scratch](#), [JSer.info](#)



目的

- 新規でそこそこ複雑なウェブページを作る(アプリに近い)
- ある程度柔軟に拡張でき、メンテできるような設計にしたい
- 無難に**React** + 何かでちゃんと設計して作っていききたい
- この設計部分をどう決めていくのかという話

現状

- チームに**React/Flux/Redux**を触ったことがない人が多い
- どれが(主に**View**以外の設計)ベストかは分からない
 - **Flux**的な部分の話


ものごとは変わる。
混乱は変わらない

混乱の原因

- 情報過多
- 情報不足
- 適切でない情報
- 上記の組み合わせ！

via P21 今日からはじめる情報設計

情報の共有

- 情報不足
 - そもそも**React**などを知らない人には知ってもらう必要がある
- **Flux** 何がいいのか分からない問題
 -  既存の実装でサンプルを作ってみてもらう
 - どれがしっくりくるのかを聞く

サンプルづくり

- サンプルのテーマ決め
 - 新しく作るページに必要な要素を簡単に盛り込む
 - タイトル、画像の切り替え、アニメーション、非同期でデータを読み込むなど
- テーマを決めたら作る

サンプルづくり

→ Flux Util

→ Redux

→ Fluxible

→ material-flux

→ でそれぞれ作った

→ **View(React)**は全て同じものを参照して、**View**と他はちゃんと分離出来ることも証明してみせた

作ったサンプルをみてもらおう

- 🤔 **Action/ActionCreator**が謎い
 - ある処理をするので、**execute**を持つ**Usecase**クラスとしてみると親しみが湧く
- 🤔 **Store**って何? モデルではない?
 - モデルというよりは**State**の入れ物に近い
 - **State**とわかるような名前にしてみるとか

▶ 意見を聞いて実装してみる

Usecase

→ `execute`を持っている`executable`クラスっぽい

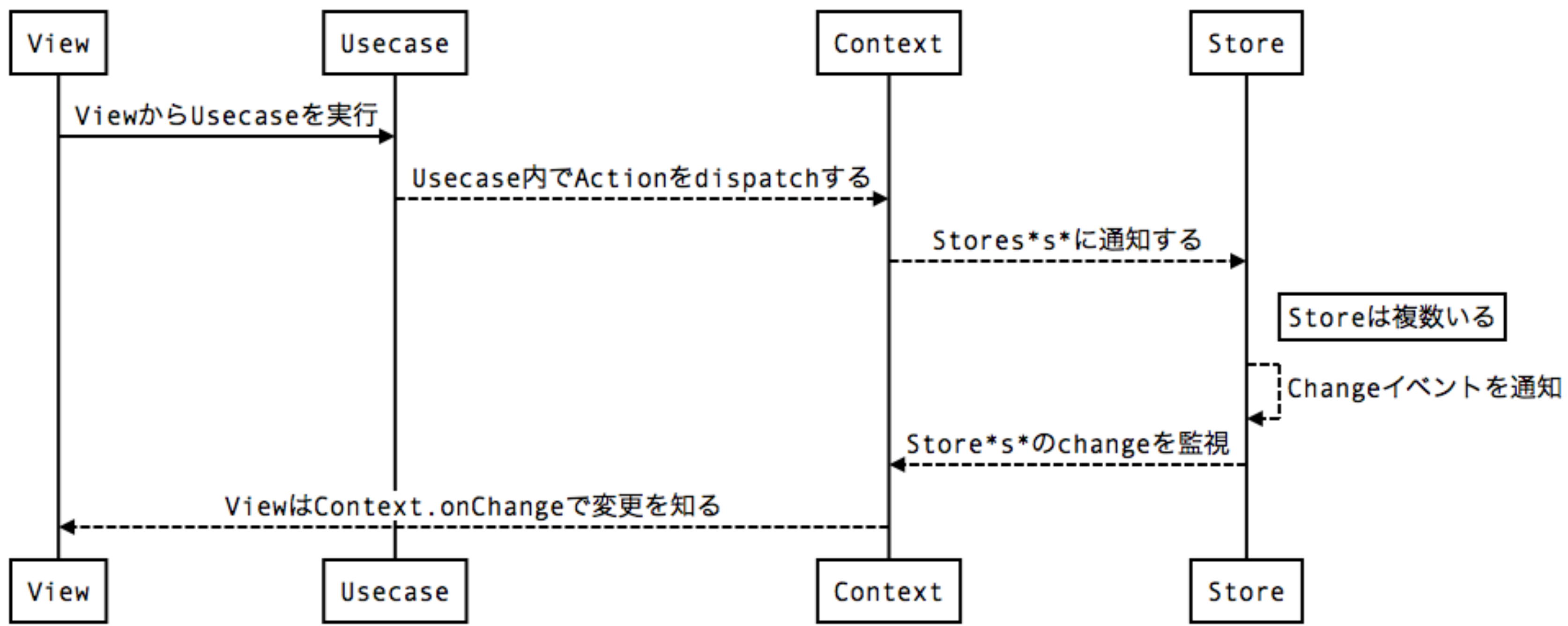
```
class UseCase {  
    // executeは暗黙的に呼ばれる  
    execute(context) {  
        // dispatch  
        context.dispatch(key, value);  
    }  
}
```

Usecaseの実行

```
const stateStore = new StateStore();
const dispatcher = new Dispatcher();
// dispatcherとstoresを紐つけるContext
const appContext = new AppContext({
  dispatcher,
  stores: [stateStore]
});
// Usecaseの実行
context.execute(new UseCase({ data }));
```

→ API: Actions | Fluxible と近い

クリックしたら何か起こる



Viewをレンダリングしたい

View

Store

Storeがgetter的なものを持っているので、データくれ

データを返す

レンダリングする

View

Store

また見てもらう

- まあまあ良さそう
- でも**Store**はホントにただの入れ物っぽい
- **Usecase**にロジックが集中しそう
- ドメインとかレイヤーをちゃんと考えるの課題
- ちゃんとこれでアプリの動くのかが不安
- **My thought about beyond flux**

現状の課題/混乱をまとめる

直面していると思われる混乱を
描いてみてください

Fluxと両立することができる???

Flux設計

ドメイン

語彙豊かなモデル

エラー/ローディング

グの簡潔さ

ユーザーやステークホルダーが
あなたの混乱を説明するとしたら、
どんな単語を使うと思いますか？

メンテナンス

分かりやすさ

語彙豊か

テストしやすさ

エラー表示

ドメイン

ローディング

面倒臭さ

トレードオフ

行うことは知ること

「知っている」だけでは十分ではありません。「多くを知りすぎている」ことでも、私たちはぐずぐずと手間取ることもありえます。

実際に行うことよりも、知り続ける事を優先すると、ある時点から混乱がましてきます。

via P32 今日からはじめる情報設計

手を動かさないと進まない

- ずっと頭で考えてても設計はうまくいかない
- 実際に作る/メンテするのは**Team**なので全員が書かないと意味ない
- サンプルは作ってあるので、**Team**の人にも自分なりに書いてもらう

Teamの人にも書いてもらう間に

- **React**は**Component**志向
- **Component**とデータフローの設計は並行できる
-  **Component**の開発を進めておく

設計と実装

@azu

チームリーダー

チームメンバー

サンプル作り

設計レビュー

意見

意見を反映したサンプル作り

設計レビュー

実際にやってみないと
上手くいくか分からない

どの設計なら理解できるか
書いてみて確認

Component作り

今

設計の決定



作り始めて考える事

- 使う言葉
- 使わない言葉
- 要件(名詞 + 動詞)

使う言葉

用語	定義	由来
Container Component	stateを持つReact Component	https://facebook.github.io/flux/docs/flux-utils.html#container
View Component	stateを持たないReact Component(ただのView)	
Fluxアーキテクチャ	Dispatcher、ActionCreator、Store、Componentsという要素を持つ一方向のデータフローを持つアーキテクチャ。 それぞれの要素が厳密にどうあるべきかは定義しない	https://github.com/facebook/flux/
ActionCreator	Actionオブジェクトを作成するクラス/関数	
Actionオブジェクト	{ type : "commit", data : "value" } のようなtypeと何かしらのデータを持ったオブジェクトのこと	
Store	ビジネスロジックを持たないモデルのこと。 データを格納、取り出すことが出来る容器となるクラス	
Usecase	ビジネスロジックのこと。 ある目的を実現するための1連の処理フローを表現したもの。 1つのUsecaseが複数の処理を手続き的に行う事を認める。 Usecaseが画面の表示については扱わない。	IDDD
デザインカンブ	デザイン見本のこと (Prototなどでデザイナーさんが作った見本)	

使わない言葉

用語	理由	代わりに使う言葉
Smart Component	Reduxの人が言ってただけ	Container Component
Dumb Component	Reduxの人が言ってただけ	View Component
モック	デザイン的なモックはカンブに統一	デザインカンブ

たたかいははじまったばかりだ