フロントエンドMVCとFlux

SED | SIROK技術勉強会
2015/08/13

新しい当たり前を、つくる

About Me

- @sangotaro
- Frontend Engineer
- SIROK: 2ヶ月目

Fluxやりたいので、Flux布教する 

まずはMVCの復習

MVC

MVC

- GUIアプリを実装するためのデザインパターン
- サーバーサイドMVCは派生系
- いろんな流派があるので言及はしない

Model

データとビジネスロジックを担当。データの変更をViewに通知する。

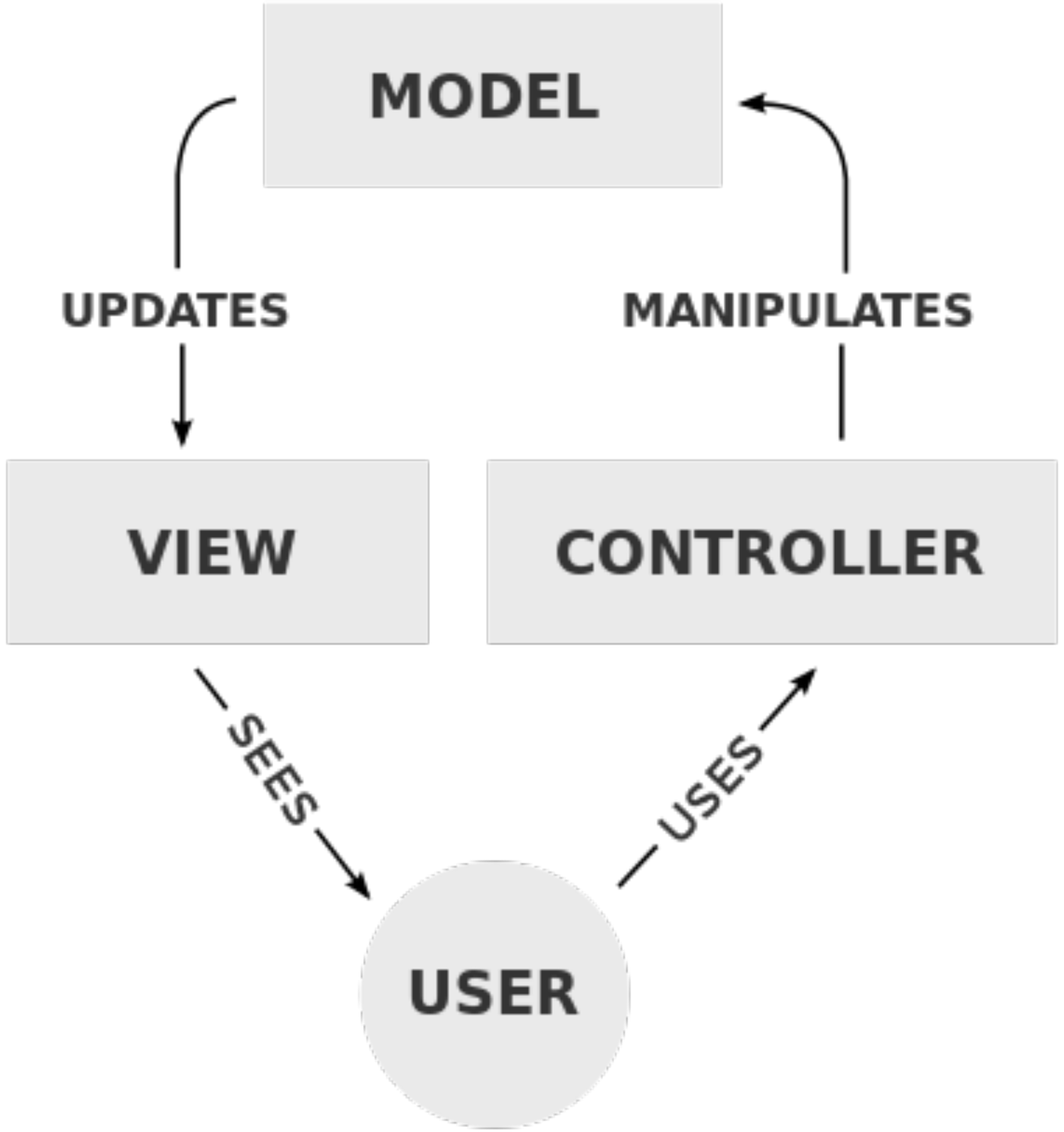
View

データを表示する。通常は階層構造をになる。

Controller

ユーザからの入力をモデルに伝える。

引用元: **https://ja.wikipedia.org/wiki/Model_View_Controller**



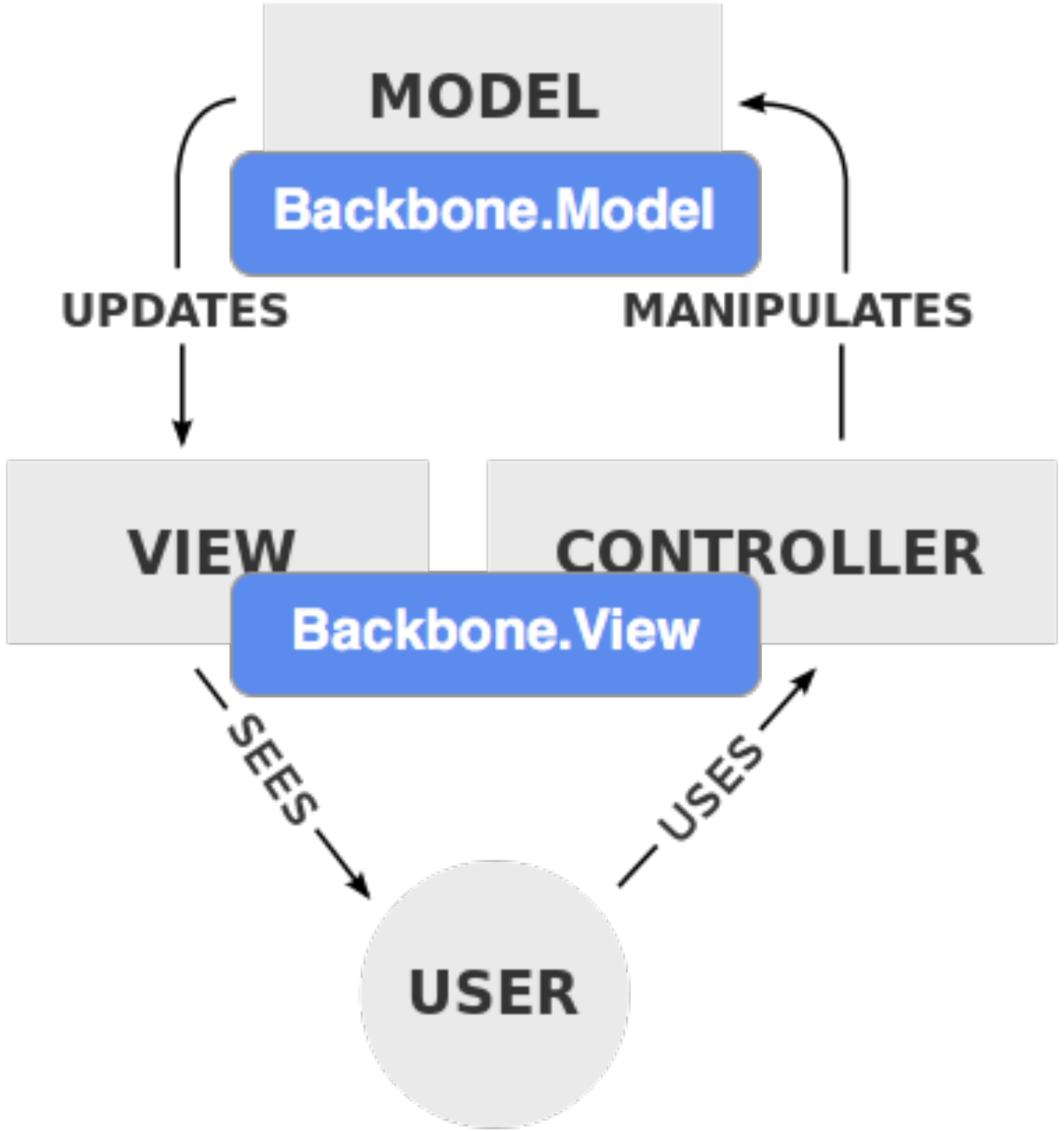
JSライブラリの有名どころ

- Backbone.js
- AngularJS
- Knockout.js
- Ember.js

それぞれ厳密にはMVCではないので、総称としてMVW or MV*と呼ばれる。

Backbone.jsでは

- **Backbone.View**: Controller、View
- **Backbone.Model**: Model



GUIアプリの設計

GUIアプリ設計のポイント

複雑な"状態"にどう立ち向かうか

状態(State)???

- 状態 = UIの状態
- データそのものではない❌
- どちらもJSONのようなデータ構造で表現できる

データと状態

ex. 在庫数が0のとき、購入ボタンが押せない

データ => 在庫数0　状態 => ボタンが無効

```
let data = {
  inventory: 0
}
let state = {
  buyButton: 'disable'
}
```

難しさ

- 管理する状態が多すぎる😵
- どこで状態を管理するか🏠
- だれが状態を変更するか👥

Backbone.jsでやってみると

公式の図

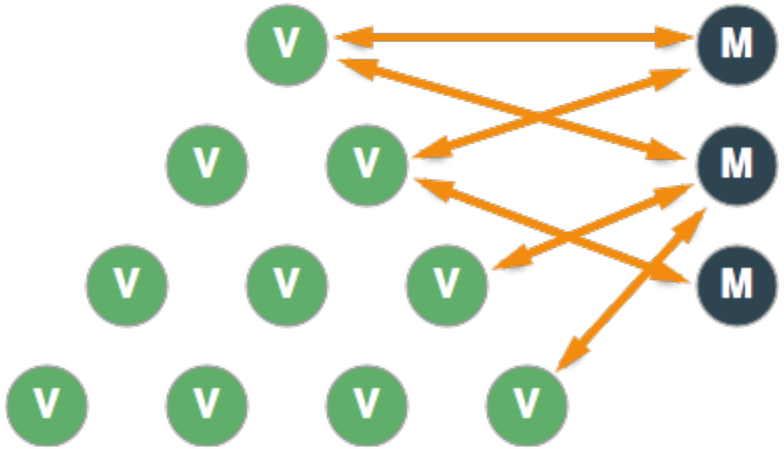
引用元: **http://backbonejs.org/**

シンプルでよさそう?

実際は複雑だし、決めること多い😖

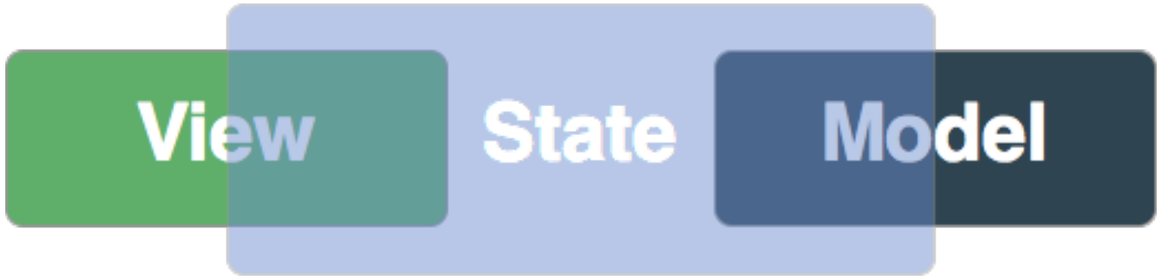
大量のView、大量のModelが相互に関連する

- Model-VIew間の双方向データフロー
- 親View、子View
- 複雑度が爆発



UIの状態はどこにあるのか

- Model or View(親View or 子View)
- そもそも意識して状態を管理しているか?
  - ModelがただのデータModelになってないか?
  - Viewで複雑なデータ処理をしていないか？



とはいえ学びはある

Backbone.jsが教えてくれたこと

Viewの部品化

- 再利用可能なView
- 最近流行りのコンポーネント指向へ
  - React
  - Angular
  - Web Components

オブザーバーパターン

- Modelが変更されたらイベント発火
- ViewはModelのイベント監視

引用元: **http://backbonejs.org/**

Flux

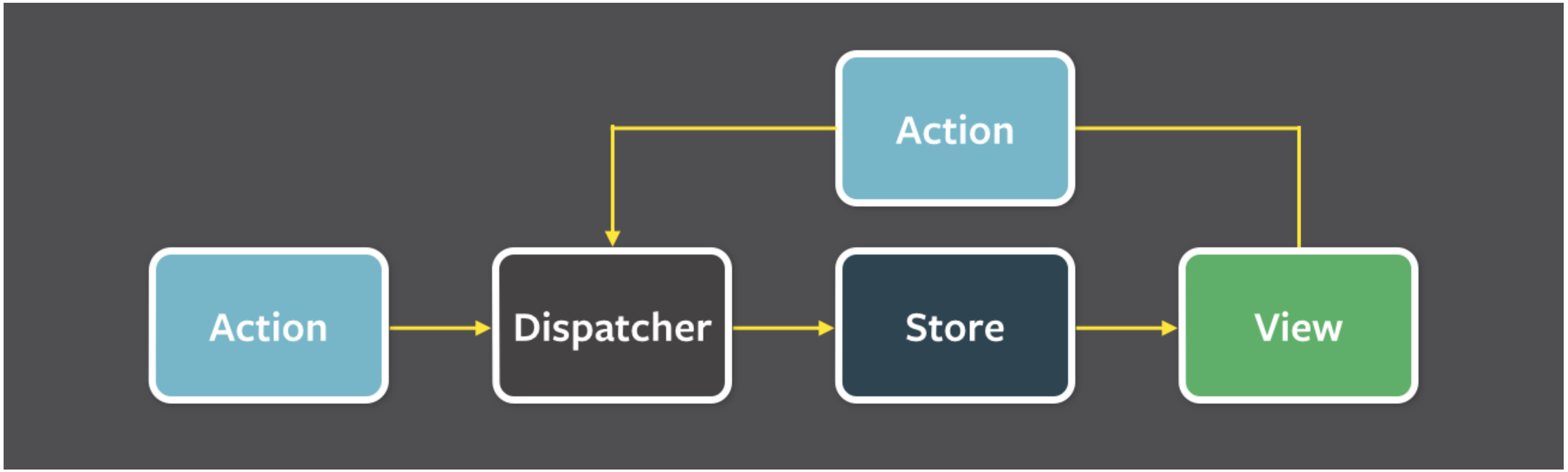
Fluxとは

- Facebookが提唱したGUIアプリのアーキテクチャ
  - ライブラリではない
  - 実装が乱立
- **Unidirectional data flow** (単方向データフロー)
  - オブザーバーパターンで実現

Fluxの登場人物

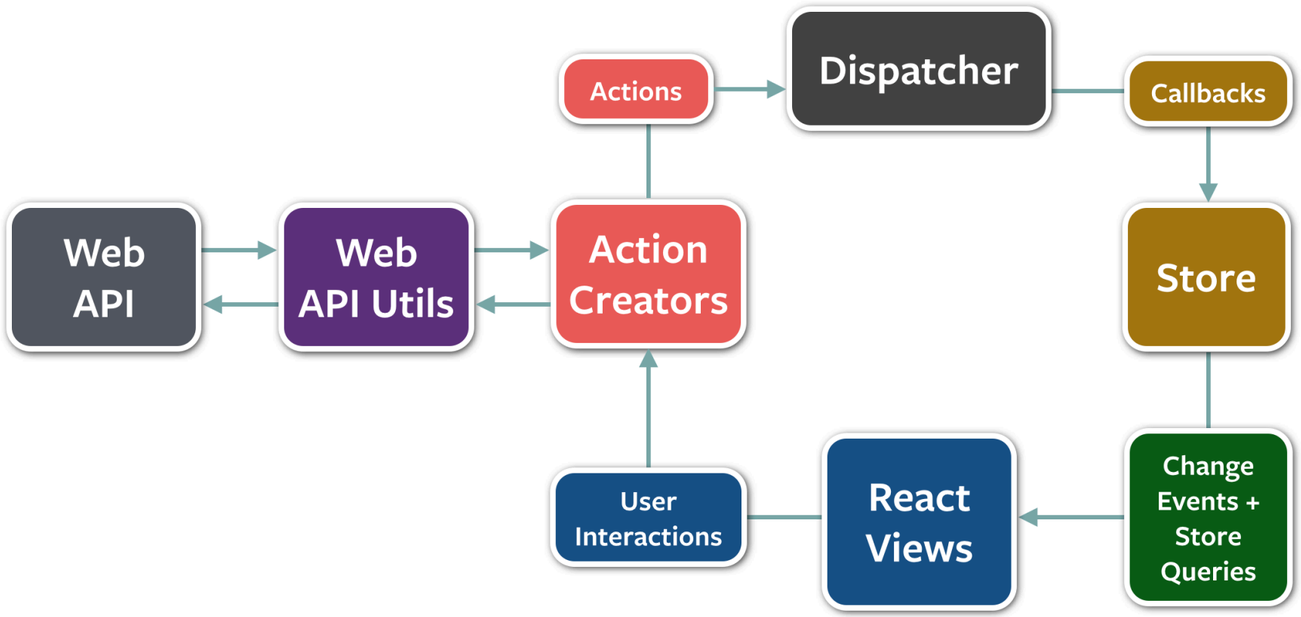
- Action
- Dispatcher
- Store
- View

有名な図



引用元: **https://facebook.github.io/flux/docs/overview.html#content**

有名な図2



引用元: https://github.com/facebook/flux/tree/master/examples/flux-todomvc/

構成例(TODOを作るだけのアプリ)

```
src
├── actions
│   └── todo-action-creators.js
├── app-constants.js
├── app-dispatcher.js
├── app.js
├── stores
│   └── todo-store.js
└── views
    ├── todo-controller-view.react.js
    └── todo-item.react.js
```

Action & Action Creator

Fluxのデータフローの開始点🔥

Action

タイプとデータをもつオブジェクト(like イベント)

Action Creator

Actionを生成するヘルパー関数(or メソッド)

```
// todo-action-creators.js
import AppDispatcher from '../app-dispatcher';
import AppConstants = from '../app-constants';

var TodoActionCreators = {
  create: function(text) {
    AppDispatcher.dispatch({
      actionType: AppConstants.TODO_CREATE, // type
      text: text // data
    });
  }
}

export default TodoActionCreators;
```

Dispatcher

- グローバルなEventEmitterみたいなもの
- ActionをStoreに届ける(オブザーバーパターン)

```
// Actionを起動(Pub) like EventEmitter#emit
AppDispatcher.dispatch(action);

// コールバックの登録(Sub) like EventEmitter#on
AppDispatcher.registor(callback);
```

Store

- データと状態を管理(状態管理ロジックもある)
- Actionでしか変更できない(No Setters, only gettes)
- Dispatcherにコールバックを登録する

```
// todo-store.js
let _todos = {}; // private data

// setter
function create(text) {
  let id = (+new Date()).toString();
  _todos[id] = {
    id: id,
    text: text
  };
}

class TodoStore extend EventEmitter {
  constructor() {
    // TODO: Register dispatcher callback
  }
  // getter
  getAll() {
    return _todos;
  }
  emitChange() {
    this.emit('change');
  }
}

export default new TodoStore;
```

```
// Register dispatcher callback
AppDispatcher.register(action => {
  switch(action.actionType) {
    case AppConstants.TODO_CREATE:
      let text = action.text.trim();
      if (text !== '') {
        create(text);
        this.emitChange(); // Emit change event
      }
      break;
    default:
      // no op
  }
});
```

View

- ex. React componets
- Storeの"状態"変化のみを監視
- Action Creatorをコール
- 大きく分けて2種
  - Controller View
  - View

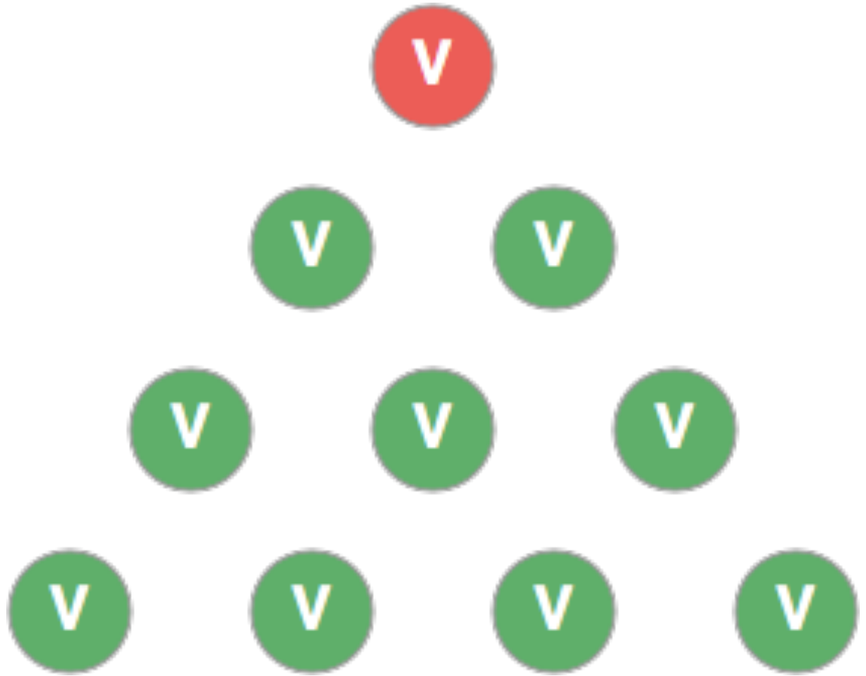
Controller View & View

Controller View

- ルート(に近い)View
- Storeの変化を監視
- 自身と子Viewをレンダリング

View

- 流れてきたデータを元にレンダリング



ざっくりまとめると

Actionを起点にデータが一周する

以上

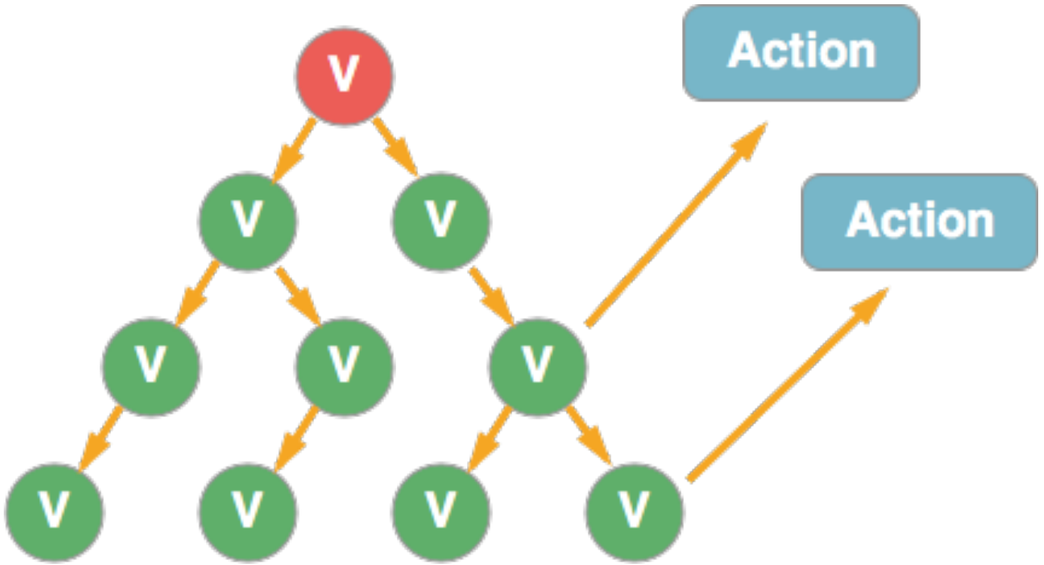
MVCとどこが違う?

MVCとの違い

- 登場人物それぞれの役割が明確
- Unidirectional data flow

View内のデータフロー

- 親Viewから子Viewへの一方通行
  - 逆はできれば避ける
- 子から親へのバケツリレーはしない
  - Actionを生成するべき



状態はStoreがもつ

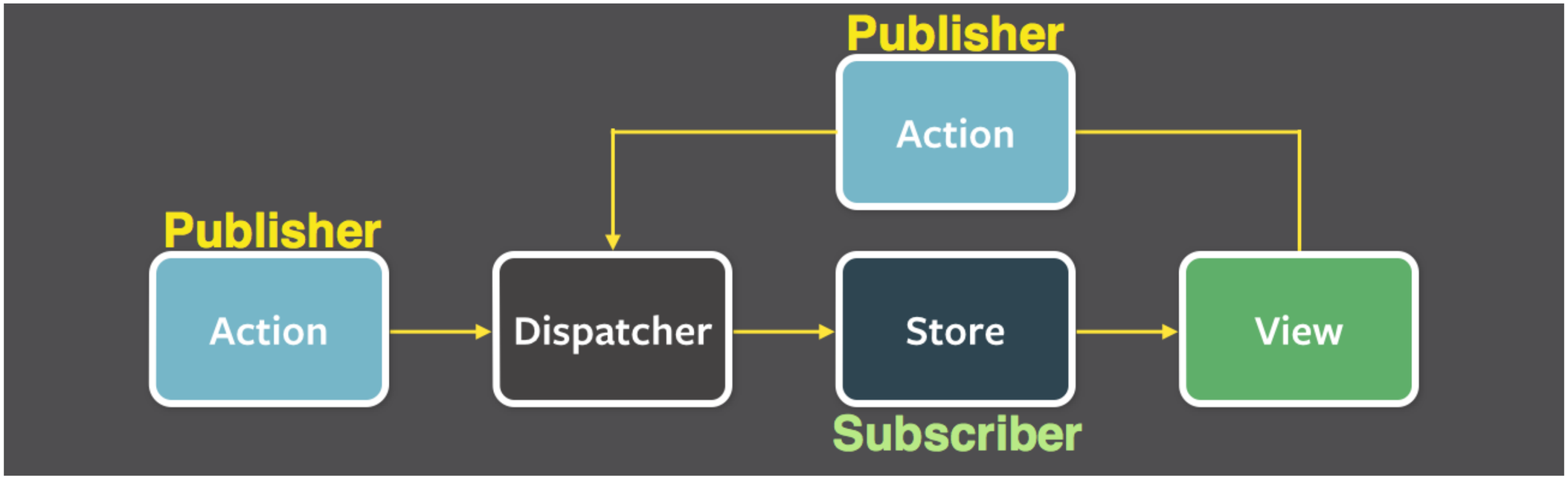
- View自身は状態を直接変更しない
  - Storeにある状態はread-only
  - ViewはAction生成するだけ

StoreとModelの違い

- StoreはModelより役割が明確
  - (データだけでなく)状態ももつ
  - Actionでしか変更できない

実装がシンプル

- Pub/Subが明確に分離している



ぶっちゃけオレオレMVCでは?

FacebookのオレオレMVC説

- たぶんそう
- 新しい登場人物を定義して明確な役割を与えた
- FluxというMV*系と思われないネーミング

最近のFlux

facebook/fluxの不満

- とっつきにくい
  - DispacherはただのEventEmitterでは?
  - Action (Creator)はただの関数では?
- シングルトン
  - サーバーサイドで困る
  - テストしにくい

それでもUnidirectional data flowはいいよね

Flux実装

- Flux Comparison
  - URL: https://github.com/voronianski/flux-comparison
- スター数上位⭐
  - Redux
  - Reflux
  - Alt

まとめ

Flux最高だからやろう

Reactじゃなくてもできるよ

次回

SED ～SIROK技術勉強会 #4「機械学習と線形代数の基礎」～

ベントに参加しています

里

への参加をキャンセル

ジを送る

ジを確認する

このイ

イベント管理

イベント

メッセージ

メッセー

メッセー

日時： 2015/08/18 (火) 19:30 to 20:30
Googleカレンダーに追加

定員： 20 人

参加費： 無料