Chapter 24: Using ReactJS in Flux way

It comes very handy to use Flux approach, when your application with ReactJS on frontend is planned to grow, because of limited structures and a little bit of new code to make state changes in runtime more easing.

Section 24.1: Data Flow

> This is outline of comprehensive Overview.

Flux pattern assumes the use of unidirectional data flow.

1. **Action** — simple object describing action `type` and other input data.
2. **Dispatcher** — single action receiver and callbacks controller. Imagine it is central hub of your application.
3. **Store** — contains the application state and logic. It registers callback in dispatcher and emits event to view when change to the data layer has occurred.
4. **View** — React component that receives change event and data from store. It causes re-rendering when something is changed.

> As of Flux data flow, views may also **create actions** and pass them to dispatcher for user interactions.

Reverted

To make it more clearer, we can start from the end.

- Different React components (*views*) get data from different stores about made changes.

> Few components may be called **controller-views**, cause they provide the glue code to get the data from the stores and to pass data down the chain of their descendants. Controller-views represent any significant section of the page.

- *Stores* can be remarked as callbacks that compare action type and other input data for business logic of your application.
- *Dispatcher* is common actions receiver and callbacks container.
- *Actions* are nothing than simple objects with required `type` property.

> Formerly, you'll want to use constants for action types and helper methods (called **action creators**).