How to Manage `React State`

DEVELOPER MASTER CLASS < Developer Circle: Hanoi >

Woo Gim

** Find me on Facebook **

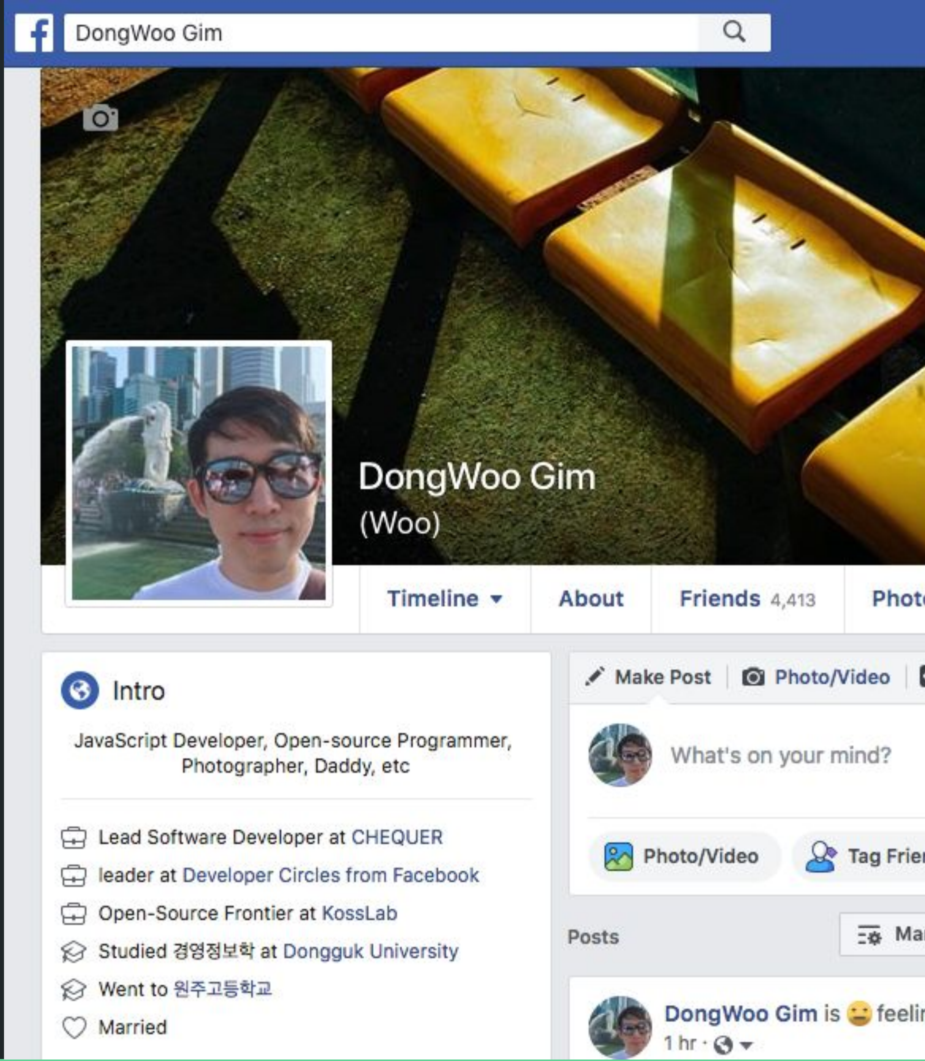
I love JavaScript.

Developer in Korea over 14 years.

Web & Mobile Full-stack Developer.

Developer Circle: Seoul Lead.

https://www.facebook.com/woo.gim



DongWoo Gim

DongWoo Gim
(Woo)

Intro

JavaScript Developer, Open-source Programmer, Photographer, Daddy, etc

Lead Software Developer at CHEQUER

leader at Developer Circles from Facebook

Open-Source Frontier at KossLab

Studied 경영정보학 at Dongguk University

Went to 원주고등학교

Married

DongWoo Gim is

1 hr ·

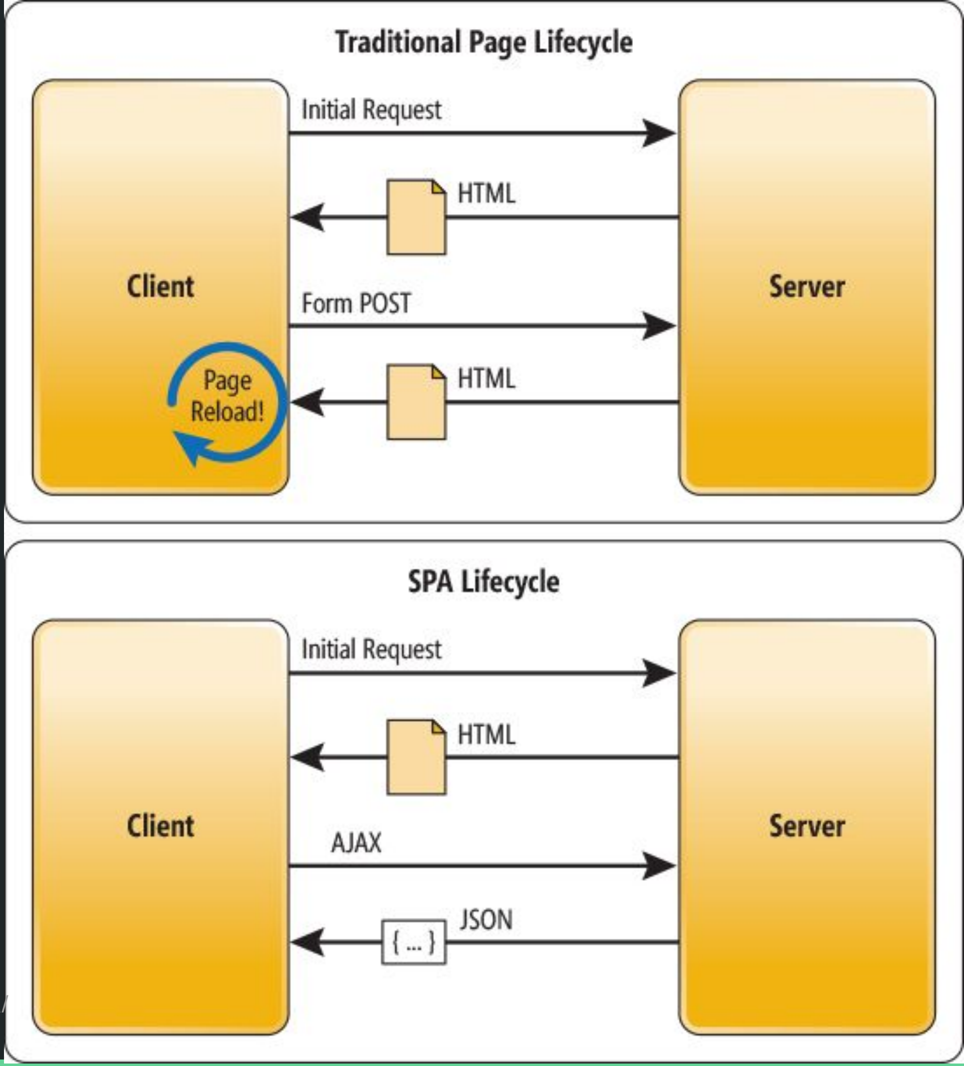
What is SPA?

A single-page application (SPA) is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.

This approach avoids interruption of the user experience between successive pages...

** Facebook is the most famous SPA. **

https://thiyaratech.wordpress.com/2016/06/27/what-is-single-page-applicationsspa/



SPA needs `View Library`

React

by Facebook

Declarative rendering

Virtual Dom

Component Based

Vue

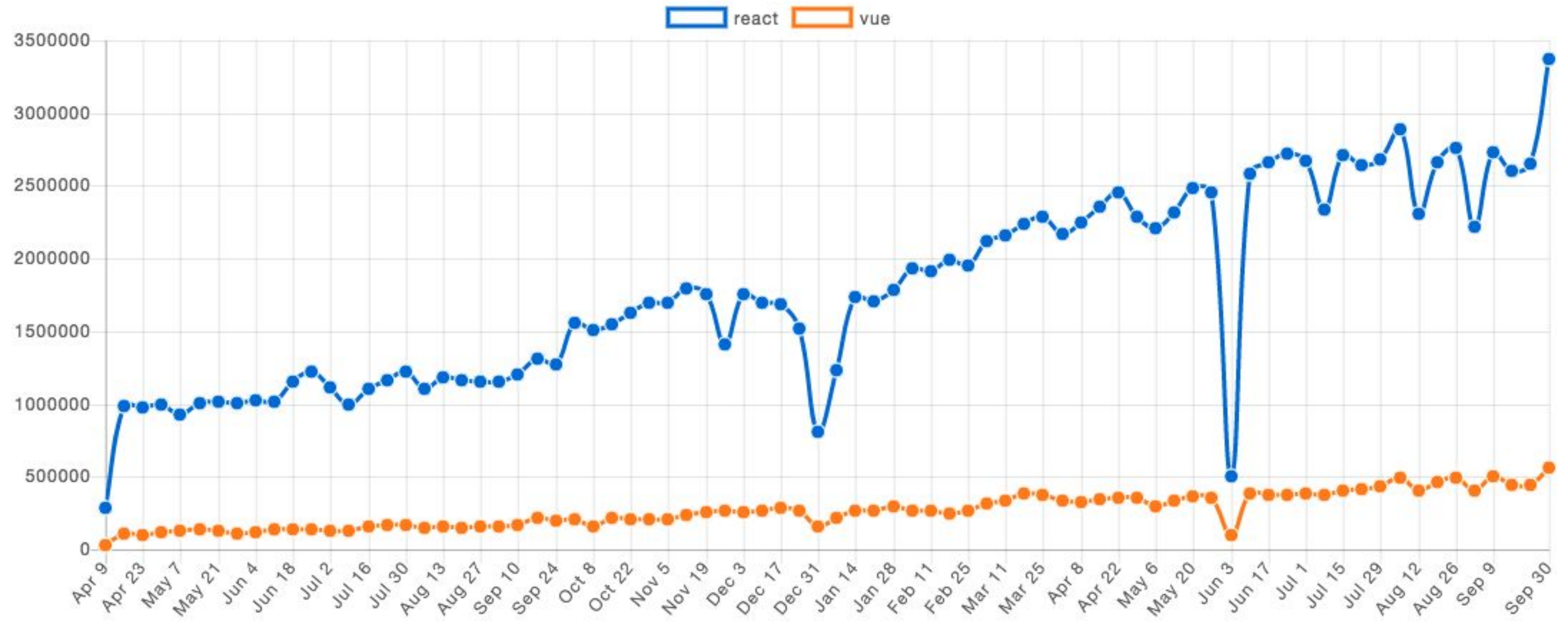
by Evan You

Declarative rendering

Virtual Dom

Component Based

The Winner is.. React!



https://www.npmtrends.com/react-vs-vue

React is more honest than Vue

React is simple, less Magic.

React uses plain JavaScript. (Class and Method)

Faster, bolder improvements. React is developed by a big team, but is very fast.

React supports TypeScripts seamlessly. TypeScript is really awesome!

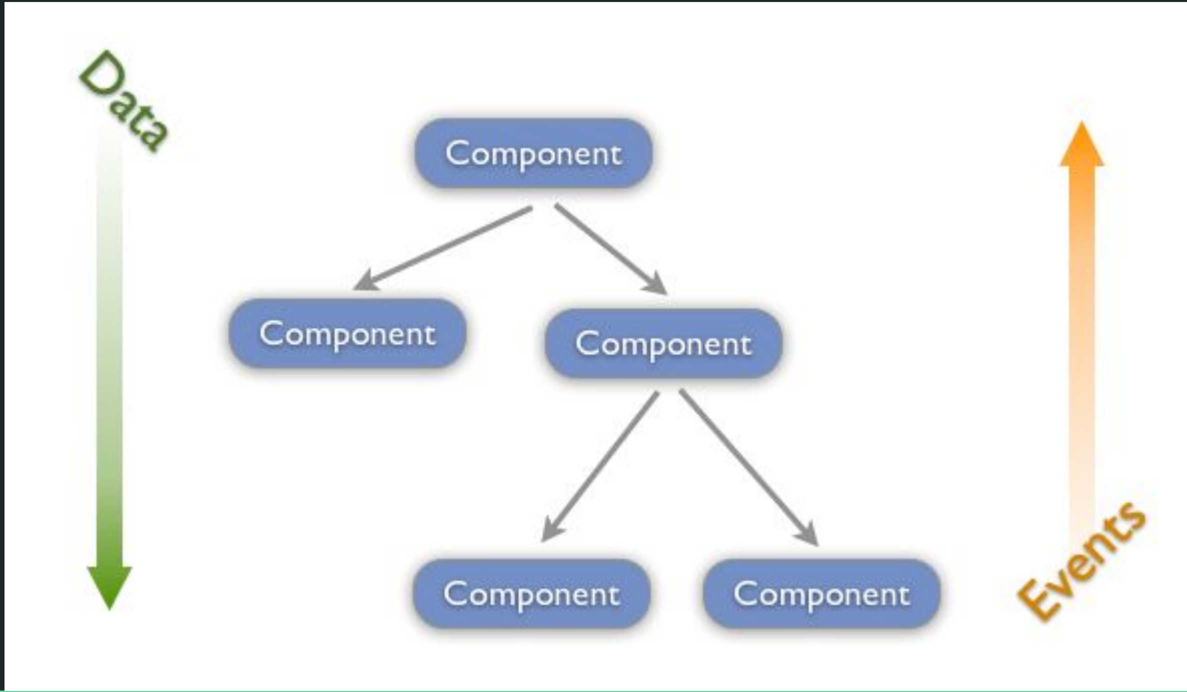
But React is more difficult to learn than Vue.

React needs `State Management`.

State Management is called `Store`.

React is only an UI library.

One way data binding.



https://medium.embengineering.com/

Redux

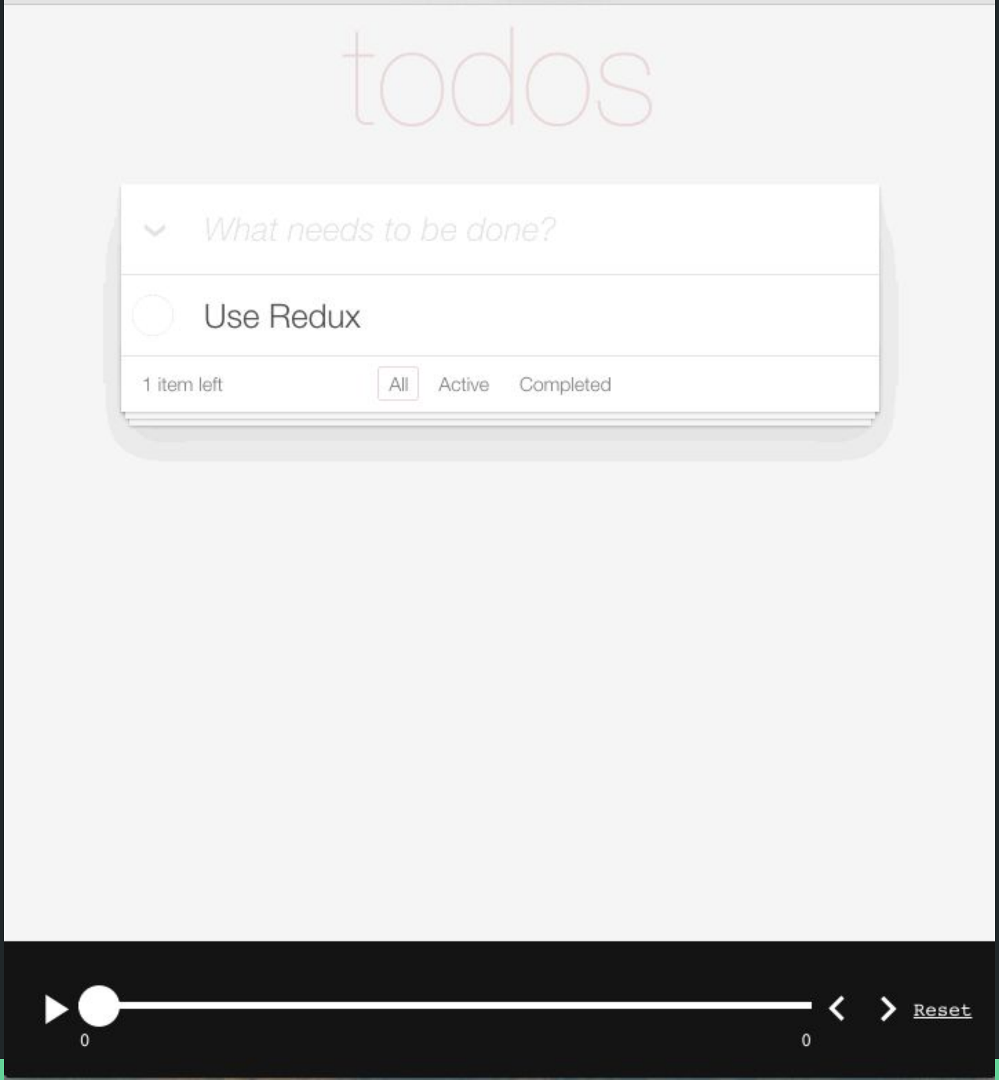
Redux is a predictable state container for SPA.

It helps write applications that behave consistently,
run in different environments (client, server, native)
and are easy to test!

+ Deterministic view render

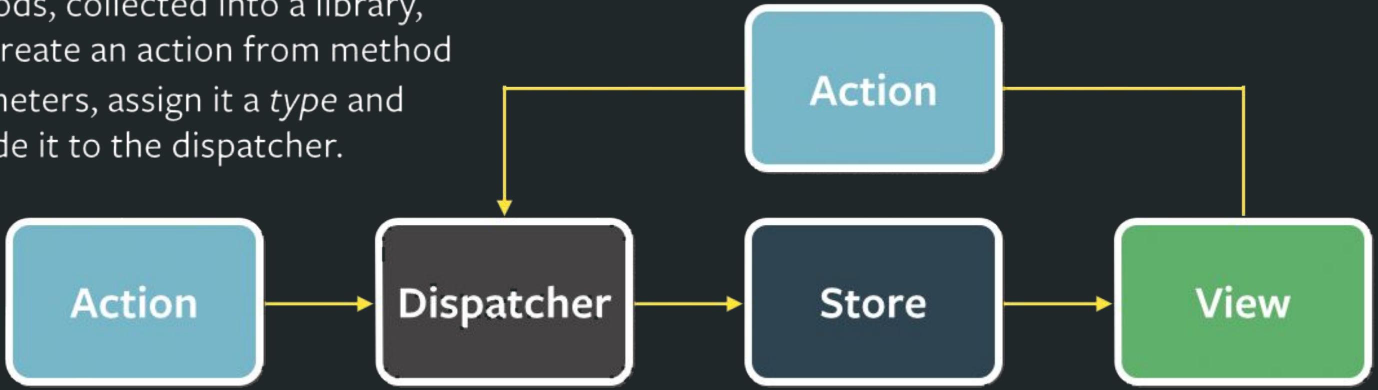
+ Deterministic state reproduction

> This enables time-travel debugging.


todos
What needs to be done?
Use Redux
1 item left
All
Active
Completed
0
0
Reset

Redux is the flux pattern.

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.



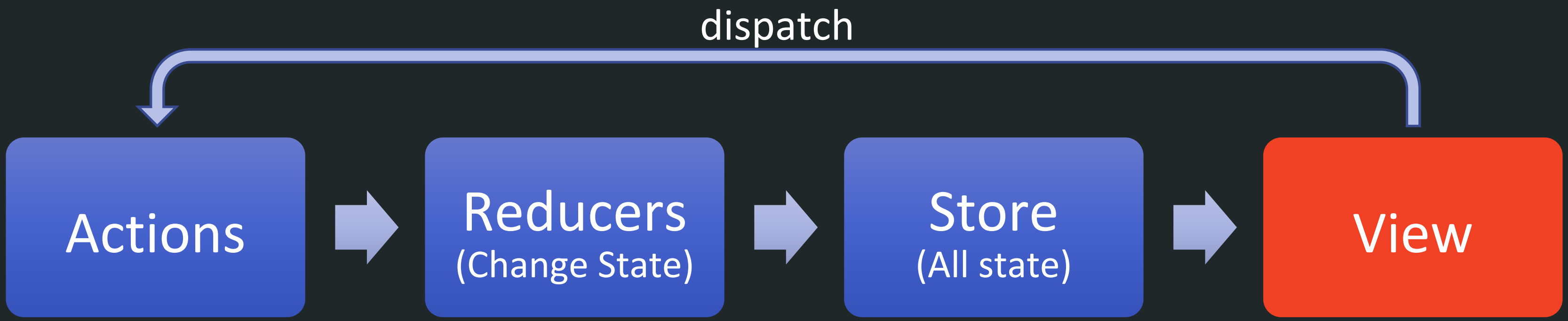
Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

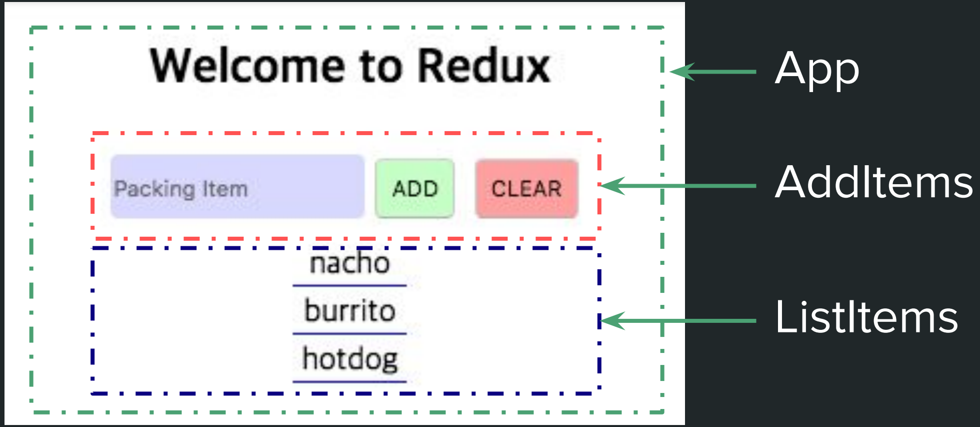
Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

https://medium.com/@murtazazaidi_

Redux Architecture



Sample App


Welcome to Redux
Packing Item
ADD
CLEAR
nacho
burrito
hotdog
App
AddItems
ListItems

Action

```
export const ItemsActions = {
  ADD_ITEM: "ADD_ITEM",
  CLEAR: "CLEAR",
  SET_NEW_ITEM_NAME: "SET_NEW_ITEM_NAME"
};

// Action creators store.dispatch actions
export const ItemActionCreators = {
  addItem: () => {
    return {
      type: ItemsActions.ADD_ITEM
    };
  },

  clear: () => {
    return {
      type: ItemsActions.CLEAR
    };
  },

  setNewItemName: value => {
    return {
      type: ItemsActions.SET_NEW_ITEM_NAME,
      value
    };
  }
};
```

Reducer

```
const INITIAL_STATE = {
  myItems: ["nacho", "burrito", "hotdog"],
  newItemName: ""
};

export function reducer(state = INITIAL_STATE, action) {
  switch (action.type) {
    case ItemsActions.ADD_ITEM:
      return {
        ...state,
        myItems: [...state.myItems, state.newItemName],
        newItemName: ""
      };
    case ItemsActions.CLEAR:
      return {
        ...state,
        myItems: []
      };
    case ItemsActions.SET_NEW_ITEM_NAME:
      return {
        ...state,
        newItemName: action.value
      };
    default:
      return state;
  }
}

export default { items: reducer };
```

Store

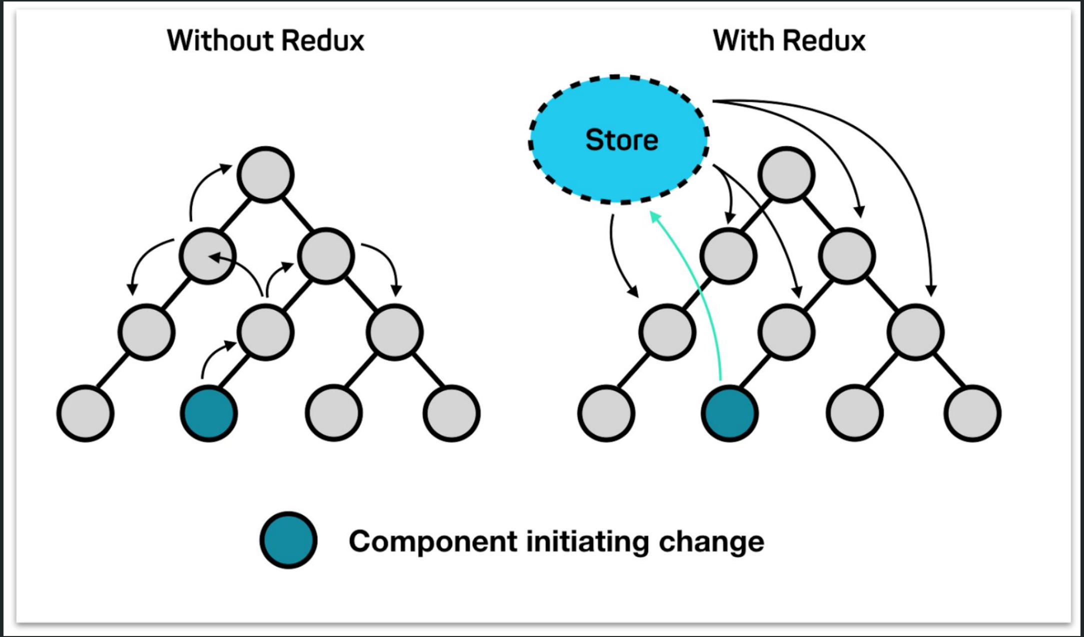
```
// RootReducer
export default combineReducers({
  ...ItemReducers
});

// store
const baseStore = createStore(RootReducer, applyMiddleware(...middleware));
export default initialState => {
  return baseStore;
};
```

View

```
// AddItems
const mapStateToProps = state => ({ value: state.items.newItemName });

const { setNewItemName, addItem, clear } = ItemActionCreators;

const mapDispathToProps = {
  setNewItemText: e => setNewItemName(e.target.value),
  addItem,
  clear
};

export default connect(mapStateToProps, mapDispathToProps)(AddPackingItem);

// App
export default class App extends React.Component {
  render() {
    return (
      <Provider store={store}>
        <div style={styles}>
          <h2>Welcome to Redux</h2>
          <AddItems />
          <ListItems />
        </div>
      </Provider>
    );
  }
}
```

React with Redux

Redux stores inject states into a specific component.



https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/

Redux has many middlewares

redux-thunk

Simple middleware for asynchronous.

redux-saga

An alternative side effect model for Redux apps

redux-observable

RxJS middleware for action side effects in Redux using `Epics`

React Context API is similar to Redux

React 16.3+ is provided with Context API.

Provider and Consumer are a Pair.

Context holds `No State`. But we can use states in the Provider component.

The state in the Provider component can be manipulated by the Consumer.

Pass actions down through the Context.

Context is suitable for very simple apps.

Provider

```
export default class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      allItems: ["nachos", "burritos", "hot dog"],
      newItemName: "",
      addItem: this.addItem,
      setNewItemName: this.setNewItemName,
      clear: this.clear
    };
  }

  addItem = () => {
    this.setState(state => ({
      allItems: [...state.allItems, state.newItemName],
      newItemName: ""
    }));
  };

  setNewItemName = event => {
    this.setState({ newItemName: event.target.value });
  };

  clear = () => {
    this.setState({ allItems: [] });
  };
```

Provider

```
  render() {
    return (
      <div style={styles}>
        <h2>Welcome to React 16 Context</h2>
        <PackingContext.Provider value={this.state}>
          <AddItems />
          <ListItems />
        </PackingContext.Provider>
      </div>
    );
  }
}

// context
export const PackingDefaults = {
  allItems: ["nacho", "burrito", "hotdog"],
  newItemName: ""
};

export const PackingContext = React.createContext({
  // Default value of the context - Only if component is outside of provider
  // Usually replaced right away on setting value at top level for provider
  ...PackingDefaults
});
```

Consumer

```
export default class AddItems extends Component {
  render() {
    return (
      <PackingContext.Consumer>
        {({ newItemName, addItem, setNewItemName, clear }) => (
          <AddPackingItem
            addItem={addItem}
            setNewItemText={setNewItemName}
            value={newItemName}
            clear={clear}
          />
        )}
      </PackingContext.Consumer>
    );
  }
}

export default class ListItems extends Component {
  render() {
    return (
      <PackingContext.Consumer>
        {({ allItems }) => <SimpleList value={allItems} />}
      </PackingContext.Consumer>
    );
  }
}
```

MobX is an alternative for Redux

Simple, scalable state management.

MobX does not need `setState`.

Anything that can be derived from the application state, should be derived

** Automatically **

Developed as TypeScript.

Observable Store

```
import { observable } from "mobx";

class ObservableListStore {
  @observable allItems = ["nacho", "burrito", "hotdog"];
  @observable newItemName = "";

  addItem = () => {
    this.allItems.push(this.newItemName);
    this.newItemName = "";
  };

  clear = () => {
    this.allItems = [];
  };

  setNewItemName = e => {
    this.newItemName = e.target.value;
  };
}

const observableListStore = new ObservableListStore();
export default observableListStore;
```

View

```
import { observer } from "mobx-react";

@observer
export default class AddItems extends Component {
  render() {
    return (
      <AddPackingItem
        addItem={ListStore.addItem}
        setNewItemText={ListStore.setNewItemName}
        value={ListStore.newItemName}
        clear={ListStore.clear}
      />
    );
  }
}

@observer
export default class ListItems extends Component {
  render() {
    return <SimpleList value={[...ListStore.allItems]} />;
  }
}
```

Redux vs MobX

Immutable	Mutable
Many action codes	Simple update
Pure object	Combine with instance
Low serialize cost	High serialize cost
Tree shape	Not tree
Time-travel	No time-travel

https://www.slideshare.net/hyunseoblee7/mobx-mst-state-management-109460634

MobX-state-tree

Model Driven State Management.

Central in MST (mobx-state-tree) is the concept of a living tree.

The tree has a shape (type information) and state (data).

Strictly protected objects enriched with ** runtime type information **.

From this living tree, immutable, structurally shared,
snapshots are automatically generated.

MST is suitable for apps that deal with complex data.

MST Store

```javascript
import { types } from "mobx-state-tree";

const defaultItemList = ["nacho", "burrito", "hotdog"];
const { model, optional, array, reference, string } = types;

export const ListStoreModel = model({
  allItems: optional(array(string), defaultItemList),
  newItemName: ""
}).actions(self => ({
  addItem() {
    self.allItems.push(self.newItemName);
    self.newItemName = "";
  },
  setNewItemName(e) {
    self.newItemName = e.target.value;
  },
  clear() {
    self.allItems = [];
  }
}));
```

View

```
import { observer, inject } from "mobx-react";

@inject("listStore")
@observer
export default class AddItems extends Component {
  render() {
    return (
      <AddPackingItem
        addItem={this.props.listStore.addItem}
        setNewItemText={this.props.listStore.setNewItemName}
        value={this.props.listStore.newItemName}
        clear={this.props.listStore.clear}
      />
    );
  }
}

@inject("listStore")
@observer
export default class ListItems extends Component {
  render() {
    return <SimpleList value={[...this.props.listStore.allItems]} />;
  }
}
```

MobX-state-tree

Immutable	Mutable
Many action codes	Simple update
Pure object.	Combine with instance
Low serialize cost	High serialize cost
Tree shape	Not tree
Time-travel	No time-travel

Runtime type checking

Observable

Model Driven

Memoization

Weaknesses of MST

Lots of magic

Slow performance

Difficult model typing

Less reference

Poor documentation

Conclusion

Redux

For the normal. Common use.

Context API

For the simple apps.

MobX

For high productivity.

MobX-state-tree

For the complex model

React state museum

A whirlwind tour of React state management systems by example.

https://github.com/GantMan/ReactStateMuseum

REACT STATE MUSEUM

Thank you.

Woo Gim (gimdongwoo@gmail.com)