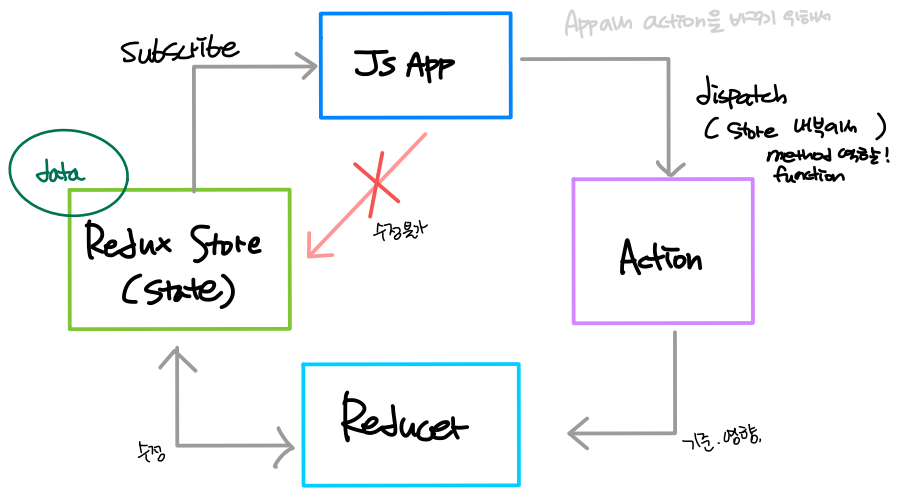
Redux

I love the light in your eyes and the dark in your heart

Redux

js앱을 위한 state 를 보관하는 요소



1. JS app 을 위한 라이브러리
   react/angular/view/ vanilla js에 모두 사용 가능

2. state container
   js app을 위한 state을 보관하는 곳
   앱의 state는 앱의 모든 각각 컴포넌트에 대표된다?

3. predictable
   app state는 변화 할 수 있음 -> (추적할 수 있음) 예측 가능함

   * react와 같이 쓰는 이유?
   react redux -> react랑 redux를 함께 연결해주는 라이브러리

- Store : app에 state를 지정 | Shop -> 가게
- Action: 무슨일이 일어난 것인지, state의 변화를 알려줌 | BUY_CAKE 케익을 사려는 의도
- Reducer: store와 action을 묶어줌 | shopkeeper

1. 하나의 store의 object tree안에 모든 app의 state가 저장됨
2. (read only)state를 바꿀수 있는 방법 :: action을 내보내는것 - 무슨일이 일어났는지 묘사하는 object
3. state tree가 action에 의해 바뀌는 것을 구체화 하기 위해서는, pure reducer 를 작성해야함.

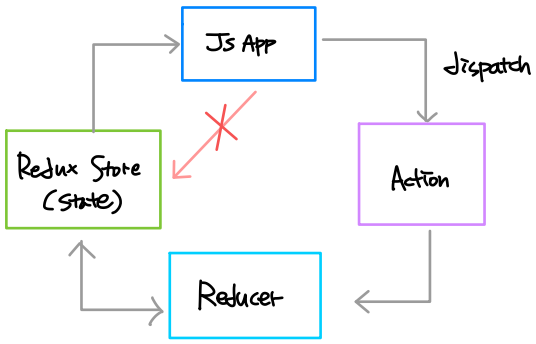
** Action: app이 store와 상호작용할 수 있는 유일한 방법
   app에서 redux store로 정보를 전송
   평범한 js object
   수행할 action의 type을 가리키는 'type' property를 가짐

** Reducer : store에 보내는 액션에 반응하여 app의 state가 어떻게 변화하는지 구체화 하는것
   state, action을 매개변수로 받는 function과 app의 nextState(변화할 state)를 return함
   (previousState, action) => newState

** Redux Store: 전체 app에서 단지 하나의 store만 가지고 있음
   - app state를 가지고 있음
   - getState() 로 접근할 수 있음
   - dispatch(action) 로 state를 업데이트 할 수 있음 (method/function과 비슷한 역할)
   - subscribe(listner) 로 listner를 등록할 수 있음
   - subscribe(listner) 로 리턴한 함수를 통해 listner의 등록을 제어할 수 있음

** Redux function
   - combine() :: 여러개의 reducer를 한번에 묶어줌

redux_basic




```
1  // import redux from 'redux'; -> in react
2  const redux = require("redux");
3  const createStore = redux.createStore;
4
5  const BUY_CAKE = "BUY_CAKE";
6
7  // Action :: Just object with type prop
8  function buyCake() {
9    return {
10     type: BUY_CAKE,
11     info: "First redux Action",
12   };
13 }
14
15 const initialState = {
16   numOfCakes: 10,
17 };
18
19 // Reducer
20 // pure function (previousState, action) => newState
21 const reducer = (state = initialState, action) => {
22   switch (action.type) {
23     case BUY_CAKE:
24       return {
25         // return new object
26         ...state, // copy of the state object + updated state
27         numOfCakes: state.numOfCakes - 1,
28       };
29     default:
30       return state;
31   }
32 };
33
34 // Redux Store - created
35 const store = createStore(reducer);
36
37 // 1. getState(): it will show Initial state in store
38 console.log("Initial state: ", store.getState()); // 10
39
40 // 2. subscribe(): anytime the store updates, reload state
41 const unsubscribe = store.subscribe(() => {
42   console.log("Updated state: ", store.getState());
43 });
44
45 // 3. dispatch() : update state when it's called
46 store.dispatch(buyCake()); // 1st - 9
47 store.dispatch(buyCake()); // 2nd - 8
48 store.dispatch(buyCake()); // 3rd - 7
49
50 // 4.
51 unsubscribe();
```

Action

Reducer

Store

① action 정의

② Reducer

③ Store 생성 + Reducer 연결.

```
Initial state:  { numOfCakes: 10 }
Updated state:  { numOfCakes: 9 }
Updated state:  { numOfCakes: 8 }
Updated state:  { numOfCakes: 7 }
```

redux_basic

multiple_reducer1

```
1  const redux = require("redux");
2  const createStore = redux.createStore;
3
4  const BUY_CAKE = "BUY_CAKE";
5  const BUY_ICECREAM = "BUY_ICECREAM";
6
7  // Action :: Just object with type prop
8  function buyCake() {
9    return {
10     type: BUY_CAKE,
11     info: "First redux Action",
12   };
13 }
14
15 function buyIceCream() {
16   return {
17     type: BUY_ICECREAM,
18   };
19 }
20
21 const initialState = {
22   numOfCakes: 10,
23   numberOfIceCreams: 20,
24 };
25
26 // Reducer
27 // pure function (previousState, action) => newState
28 const reducer = (state = initialState, action) => {
29   switch (action.type) {
30     case BUY_CAKE:
31       return {
32         // return new object
33         ...state, // copy of the state object + updated state
34         numOfCakes: state.numOfCakes - 1,
35       };
36     case BUY_ICECREAM:
37       return {
38         ...state,
39         numberOfIceCreams: state.numberOfIceCreams - 1,
40       };
41     default:
42       return state;
43   }
44 };
45
46 // Redux Store - created
47 const store = createStore(reducer);
48
49 // 1. getState(): it will show Initial state in store
50 console.log("Initial state: ", store.getState()); // 10
51
52 // 2. subscribe(): anytime the store updates, reload state
53 const unsubscribe = store.subscribe(() => {
54   console.log("Updated state: ", store.getState());
55 });
56
57 // 3. dispatch() : update state when it's called
58 store.dispatch(buyCake()); // 1st - 9 , 20
59 store.dispatch(buyCake()); // 2nd - 8 , 20
60 store.dispatch(buyCake()); // 3rd - 7 , 20
61 store.dispatch(buyIceCream()); // 4rd - 7, 19
62 store.dispatch(buyIceCream()); // 5rd - 7, 18
63
64 // 4.
65 unsubscribe();
```

Action

Reducer

Store

```
Initial state:  { numOfCakes: 10, numberOfIceCreams: 20 }
Updated state:  { numOfCakes: 9, numberOfIceCreams: 20 }
Updated state:  { numOfCakes: 8, numberOfIceCreams: 20 }
Updated state:  { numOfCakes: 7, numberOfIceCreams: 20 }
Updated state:  { numOfCakes: 7, numberOfIceCreams: 19 }
Updated state:  { numOfCakes: 7, numberOfIceCreams: 18 }
```

redux_basic

multiple_reducer2

```
const redux = require("redux");
const createStore = redux.createStore;
const combineReducers = redux.combineReducers;

const BUY_CAKE = "BUY_CAKE";
const BUY_ICECREAM = "BUY_ICECREAM";

// Action :: Just object with type prop
function buyCake() {
  return {
    type: BUY_CAKE,
    info: "First redux Action",
  };
}

function buyIceCream() {
  return {
    type: BUY_ICECREAM,
  };
}

// be able to manage seperate folders
const initialCakeState = {
  numOfCakes: 10,
};
const initialIceCreamState = {
  numberOfIceCreams: 20,
};

// Reducer
// pure function (previousState, action) => newState
const cakeReducer = (state = initialCakeState, action) => {
  switch (action.type) {
    case BUY_CAKE:
      return {
        // return new object
        ...state, // copy of the state object + updated state
        numOfCakes: state.numOfCakes - 1,
      };
    default:
      return state;
  }
};

const iceCreamReducer = (state = initialIceCreamState, action) => {
  switch (action.type) {
    case BUY_ICECREAM:
      return {
        ...state,
        numberOfIceCreams: state.numberOfIceCreams - 1,
      };
    default:
      return state;
  }
};

// Redux Store - created (combined!)
const rootReducer = combineReducers({
  cake: cakeReducer,
  iceCream: iceCreamReducer,
});
const store = createStore(rootReducer);

// 1. getState(): it will show Initial state in store
console.log("Initial state: ", store.getState()); // 10

// 2. subscribe(): anytime the store updates, reload state
const unsubscribe = store.subscribe(() => {
  console.log("Updated state: ", store.getState());
});

// 3. dispatch() : update state when it's called
store.dispatch(buyCake()); // 1st - 9 , 20
store.dispatch(buyCake()); // 2nd - 8 , 20
store.dispatch(buyCake()); // 3rd - 7 , 20
store.dispatch(buyIceCream()); // 4rd - 7, 19
store.dispatch(buyIceCream()); // 5rd - 7, 18

// 4.
unsubscribe();
```

Action

Reducer

Store

```
Initial state:  { cake: { numOfCakes: 10 },
  iceCream: { numberOfIceCreams: 20 } }
Updated state:  { cake: { numOfCakes: 9 }, iceCream: { numberOfIceCreams: 20 } }
Updated state:  { cake: { numOfCakes: 8 }, iceCream: { numberOfIceCreams: 20 } }
Updated state:  { cake: { numOfCakes: 7 }, iceCream: { numberOfIceCreams: 20 } }
Updated state:  { cake: { numOfCakes: 7 }, iceCream: { numberOfIceCreams: 19 } }
Updated state:  { cake: { numOfCakes: 7 }, iceCream: { numberOfIceCreams: 18 } }
```

Redux_async

data fetching

middleware
: Redux를 확장하는데 사용
: action을 dispatch하는것과 reducer에 도달하는 순간 사이에서 third party 익스텐션 포인트를 제공함
: middleware를 로깅, 버그 보고, 비동기적 업무를 수행할때 사용함

Async action creators(packages)
- axios : API end point에 요청
- redux-thunk : async action creators 정의 (action 이 아닌 function을 리턴하게 해줌!)
- middleware : 적용

```
const redux = require("redux");
const createStore = redux.createStore;
const applyMiddleware = redux.applyMiddleware;
const thunkMiddleware = require("redux-thunk").default;
const axios = require("axios");

const initialState = {
  loading: false,      // 컴포넌트 안에서 로드될때 까지 로딩 스피너를 보여줌
  users: [],
  error: "",           // 'error reason'
};

// Actions
const FETCH_USERS_REQUEST = "FETCH_USERS_REQUEST";
const FETCH_USERS_SUCCESS = "FETCH_USERS_SUCCESS";
const FETCH_USERS_FALIURE = "FETCH_USERS_FALIURE";

const fetchUsersRequest = () => {
  return {
    type: FETCH_USERS_REQUEST,
  };
};

const fetchUsersSuccess = users => {
  return {
    type: FETCH_USERS_SUCCESS,
    payload: users,
  };
};

const fetchUsersFaliure = error => {
  return {
    type: FETCH_USERS_FALIURE,
    payload: error,
  };
};

// Reducer
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case FETCH_USERS_REQUEST:
      return {
        ...state,
        loading: true,
      };
    case FETCH_USERS_SUCCESS:
      return {
        loading: false,
        users: action.payload,
        error: "",
      };
    case FETCH_USERS_FALIURE:
      return {
        loading: false,
        users: [],
        error: action.payload,
      };
  }
};



// Action craetors -> for async
const fetchUsers = () => {
  return function(dispatch) {
    dispatch(fetchUsersRequest()); // load = true
    axios
      .get("https://jsonplaceholder.typicode.com/users") // dummy js
      .then(response => {
        // response.data is the array of users
        const users = response.data.map(user => user.id);
        dispatch(fetchUsersSuccess(users));
      })
      .catch(error => {
        // error.message is the error description
        dispatch(fetchUsersFaliure(error.message));
      });
  };
};

// Store
const store = createStore(reducer, applyMiddleware(thunkMiddleware));
store.subscribe(() => {
  console.log(store.getState());
});
store.dispatch(fetchUsers());
```

React redux

CakeContainer.js

```
import React from "react";
import { connect } from "react-redux";
import { buyCake } from "../redux";

function CakeContainer(props) {
  // got props!
  console.log(props);
  return (
    <div>
      <h2> Number of cakes - {props.numOfCakes}</h2>
      <button onClick={props.buyCake}> Buy cake </button>
    </div>
  );
}

// use this as function name
// redux state => props binding function
const mapStateToProps = state => {
  return {
    numOfCakes: state.numOfCakes,
  };
};

// dispatch => props binding function
const mapDispatchToProps = dispatch => {
  return {
    buyCake: () => dispatch(buyCake()),
  };
};

// export default CakeContainer; // => default export without redux
export default connect(mapStateToProps, mapDispatchToProps)(CakeContainer);
// [state, dispatch] props => give to CackeContainer component
```

src
- components
  - CackContainer.js U
- redux
  - cake
    - cakeAction.js U
    - cakeReducer.js U
    - cakeTypes.js U
  - index.js U
  - store.js U
- App.js M
- App.test.js
- index.css M
- index.js M

CakeTypes.Js

```
// Action type

export const BUY_CAKE = "BUY_CAKE";
```

Cake Action.Js

```
import { BUY_CAKE } from "./cakeTypes";

// Action
export const buyCake = () => {
  return {
    type: "BUY_CAKE",
  };
};
```

Action

CakeReducer.Js

```
import { BUY_CAKE } from "./cakeTypes";

// Reducer
const initialState = {
  numOfCakes: 10,
};

const cakeReducer = (state = initialState, action) => {
  switch (action.type) {
    case BUY_CAKE:
      return {
        ...state,
        numOfCakes: state.numOfCakes - 1,
      };
    default:
      return state;
  }
};

export default cakeReducer;
```

Reducer

Store.Js

```
import { createStore } from "redux";
import cakeReducer from "./cake/cakeReducer";

const store = createStore(cakeReducer);
// Store 생성.
export default store;
```

Store

App.Js

```
import React from "react";
import CakeContainer from "./components/CackContainer";

import { Provider } from "react-redux";
import store from "./redux/store";

function App() {
  return (
    // imform this is our store.
    <Provider store={store}>
      <div className="App">
        <CakeContainer />
      </div>
    </Provider>
  );
}

export default App;
```

① react-redux 안에 <Provider> 컴포넌트로 전역 컨테이너 다른 컴포넌트들 감싸줌.

→ 생성해놓은 Store를 이곳에 추가해 줌!

* 이제 리액트가 Store를 'store'로 변경해줌!

Index.Js

```
export { buyCake } from "./cake/cakeAction";
```

이것은.. ?

Connect() 함수를 이용하지 않고 React Hook 을 이용한 방법

CakeContainer.Js.

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { buyCake } from "../redux";

function HooksCakeContainer() {
  // == mapStateToProps()
  const numOfCakes = useSelector(state => state.numOfCakes);
  // == mapDispatchToProps()
  const dispatch = useDispatch();

  return (
    <div>
      <h2>Num of cakes - {numOfCakes}</h2>
      <button onClick={() => dispatch(buyCake())}> Buy cake </button>
    </div>
  );
}

export default HooksCakeContainer;
```

이벤트핸들러.

buyCake (Action) 함수로 변경!

→ 같은 결과를 보여줌

React Redux - multiple reducers

CakeContainer.JS

```
// redux state => props binding function
const mapStateToProps = state => {
  return {
    numOfCakes: state.cake.numOfCakes,
  };
};
```

IceCreamContainer.JS

CakeContainer 와 같은 형식

```
import React from "react";
import {connect} from "react-redux";
import {buyIceCream} from "../redux";

function IceCreamContainer(props) {
  // got props!
  return (
    <div>
      <h2> Number of iceCreams - {props.numOfIceCreams}</h2>
      <button onClick={props.buyIceCream}> Buy iceCream </button>
    </div>
  );
}

// use this as function name
// redux state => props binding function
const mapStateToProps = state => {
  return {
    numOfIceCreams: state.iceCream.numOfIceCreams,
  };
};

// dispatch => props binding function
const mapDispatchToProps = dispatch => {
  return {
    buyIceCream: () => dispatch(buyIceCream()),
  };
};

// export default CakeContainer; // => default export without redux
export default connect(mapStateToProps, mapDispatchToProps)(IceCreamContainer);
// [state, dispatch] props => give to CackeContainer component
```

```
src
  components
    CackContainer.js U
    HooksCakeContainer.js U
    IceCreamContainer.js U
  redux
    cake
      cakeAction.js U
      cakeReducer.js U
      cakeTypes.js U
    iceCream
      iceCreamAction.js U
      iceCreamReducer.js U
      iceCreamTypes.js U
    index.js U
    rootReducer.js U
    store.js U
  App.js M
```

IceCreamTypes.js

```
export const BUY_ICECREAM = "BUY_ICECREAM";
```

IceCream Action.js

```
import { BUY_ICECREAM } from "./iceCreamTypes";

// Action
export const buyIceCream = () => {
  return {
    type: BUY_ICECREAM,
  };
};
```

IceCream Reducers.

```
import { BUY_ICECREAM } from "./iceCreamTypes";

// Reducer
const initialState = {
  numOfIceCreams: 20
};

const iceCreamReducer = (state = initialState, action) => {
  switch (action.type) {
    case BUY_ICECREAM:
      return {
        ...state,
        numOfIceCreams: state.numOfIceCreams - 1,
      };
    default:
      return state;
  }
};

export default iceCreamReducer;
```

rootReducer.js

```
import { combineReducers } from "redux";
import cakeReducer from "./cake/cakeReducer";
import iceCreamReducer from "./iceCream/iceCreamReducer";

const rootReducer = combineReducers({
  cake: cakeReducer,
  iceCream: iceCreamReducer,
});

export default rootReducer;
```

```
function App() {
  return (
    // imform this is our store.
    <Provider store={store}>
      <div className="App">
        <CakeContainer />
        <IceCreamContainer />
      </div>
    </Provider>
  );
}

export default App;
```

Store.js

```
import { createStore } from "redux";
import rootReducer from "./rootReducer";

const store = createStore(rootReducer);

export default store;
```

index.js

```
export { buyCake } from "./cake/cakeAction";
export { buyIceCream } from "./iceCream/iceCreamAction";
```

Number of cakes - 9

Buy cake

Number of iceCreams - 17

Buy iceCream

Reducer가 두개이므로 따로따로 변화 시킬 수 O.

Redux-logger

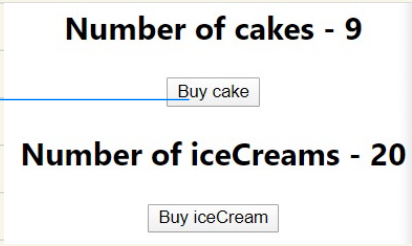
redux-logger 를 설치 + Middleware 로 입력하면,
Redux의 현재 상태 · 조작을 해줌 !

Store.JS

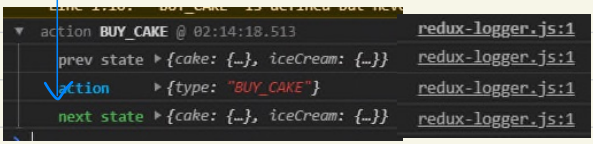
```
import { createStore, applyMiddleware } from "redux";
import logger from "redux-logger";
import rootReducer from "./rootReducer";

const store = createStore(rootReducer, applyMiddleware(logger));

export default store;
```



click!



Redux-Devtool Extension

1) Google Chrome 상단에 등록
2) Redux - Devtool Github 에서 설치
   ① google 에 'redux-devtool' 검색
   ② 가장 위에 들어감
   ③ 1.3 옵션에 Use redux-devtools-extension 방식으로 설치
   ④ import 시키기 (instruction 잘 따라하면 됨)

→ 사용방법은 - 검색

Redux-payload

이벤트시, 해당 동작만큼 dispatch 실행

NewCakeContainer.js

```jsx
import React, { useState } from "react";
import { connect } from "react-redux";
import { buyCake } from "../redux";

function NewCakeContainer(props) {
  // got props!
  const [number, setNumber] = useState(1);
  return (
    <div>
      <h2> Number of cakes - {props.numOfCakes}</h2>
      <input
        type="text"
        value={number}
        onChange={e => setNumber(e.target.value)}
      />
      <button onClick={() => props.buyCake(number)}> Buy {number} cake </button>
    </div>
  );
}

// use this as function name
// redux state => props binding function
const mapStateToProps = state => {
  return {
    numOfCakes: state.cake.numOfCakes,
  };
};

// dispatch => props binding function
const mapDispatchToProps = dispatch => {
  return {
    buyCake: number => dispatch(buyCake(number)),
  };
};

// export default CakeContainer; // => default export without redux
export default connect(mapStateToProps, mapDispatchToProps)(NewCakeContainer);
// [state, dispatch] props => give to CackeContainer component
```

CakeContainer 와 같은 역할.

dispatch Action function

CakeAction.Js

```jsx
import { BUY_CAKE } from "./cakeTypes";

// Action
export const buyCake = (number = 1) => {
  return {
    type: "BUY_CAKE",
    payload: number,
  };
};
```

∴ 기본 = 1

CakeReducer.Js

```jsx
import { BUY_CAKE } from "./cakeTypes";

// Reducer
const initialState = {
  numOfCakes: 10,
};

const cakeReducer = (state = initialState, action) => {
  switch (action.type) {
    case BUY_CAKE:
      return {
        ...state,
        numOfCakes: state.numOfCakes - action.payload,
      };
    default:
      return state;
  }
};

export default cakeReducer;
```

(기존 케이크수) - (Payload에서 받은 number)

App.Js

```jsx
function App() {
  return (
    // imform this is our store.
    <Provider store={store}>
      <div className="App">
        <CakeContainer />
        <NewCakeContainer />
      </div>
    </Provider>
  );
}
```

mapStateToProps & mapDispatchToPros

```jsx
import React from "react";
import { connect } from "react-redux";
import { buyCake, buyIceCream } from "../redux";

function ItemContainer(props) {
  return (
    <div>
      <h2>Item - {props.item}</h2>
      <button onClick={props.buyItem}> BuyItem </button>
    </div>
  );
}

// redux state => props binding function
// ownProps = component's own property
const mapStateToProps = (state, ownProps) => {
  const itemState = ownProps.cake
    ? state.cake.numOfCakes
    : state.iceCream.numOfIceCreams;

  return {
    item: itemState,
  };
};

const mapDispatchToProps = (dispatch, ownProps) => {
  const dispatchFunction = ownProps.cake
    ? () => dispatch(buyCake())
    : () => dispatch(buyIceCream());
  return {
    buyItem: dispatchFunction,
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(ItemContainer);
// if mapStateToProps doesn't exist
export default connect(null, mapDispatchToProps)(ItemContainer);
```

(Dispatch Props)만 얻을때 (State Props) 자리를 null로 채워준다.

```jsx
function App() {
  return (
    // imform this is our store.
    <Provider store={store}>
      <div className="App">
        <ItemContainer cake />
        <ItemContainer />
      </div>
    </Provider>
  );
}
```

프로퍼티