



# Redux 교육자료

작성자: @jeong deokchan ( [develoduck@gmail.com](mailto:develoduck@gmail.com) )

[Flux Design Pattern](#)

[Dispatcher](#)

[Store](#)

[View](#)

[Action](#)

[Redux란?](#)

[기존 State 관리 방식의 문제점](#)

[Redux 핵심 개념](#)

[Action](#)

[Reducer](#)

[Store](#)

[실습](#)

[앱 생성 \( `create-react-app` \)](#)

[기존 방식으로 `FilteredList` 구현하기](#)

[Redux를 적용하여 리팩토링하기](#)

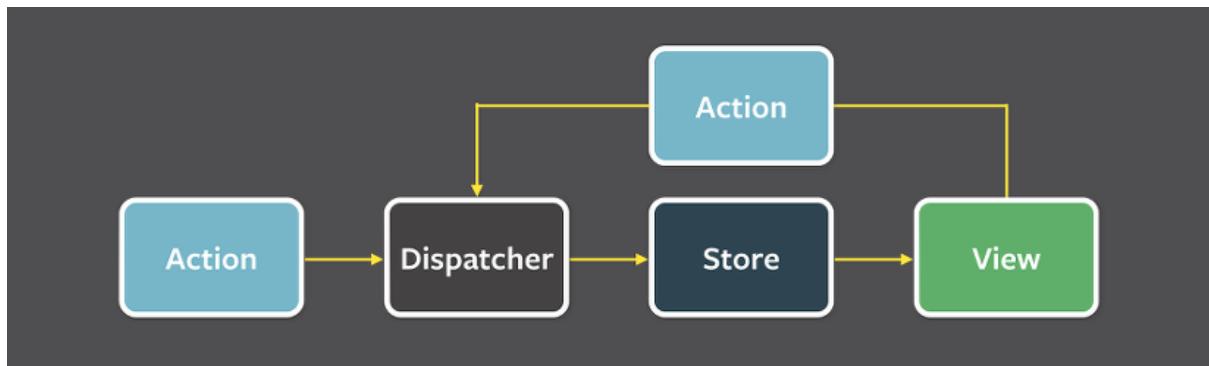
[Redux 라이브러리 설치](#)

[Redux 코드 작성](#)

[마무리 - react-redux flow diagram](#)

[Sample Code](#)

# Flux Design Pattern



## Dispatcher

Dispatcher는 Flux의 모든 데이터 흐름을 관리하는 허브 역할을 하는 부분입니다. Action이 발생되면 Dispatcher로 전달되는데, Dispatcher는 전달된 Action을 보고, 등록된 콜백 함수를 실행하여 Store에 데이터를 전달합니다. Dispatcher는 전체 어플리케이션에서 한 개의 인스턴스만 사용됩니다.

## Store

어플리케이션의 모든 상태 변경은 Store에 의해 결정이 됩니다. Dispatcher로 부터 메시지를 수신 받기 위해서는 Dispatcher에 콜백 함수를 등록해야 합니다. Store가 변경되면 View에 변경되었다는 사실을 알려주게 됩니다. Store은 싱글톤으로 관리됩니다.

## View

Flux의 View는 화면에 나타내는 것 뿐만 아니라, 자식 View로 데이터를 흘려 보내는 뷰 컨트롤러의 역할도 합니다.

## Action

Dispatcher에서 콜백 함수가 실행 되면 Store가 업데이트 되게 되는데, 이 콜백 함수를 실행 할 때 데이터가 담겨 있는 객체가 인수로 전달되어야 합니다. 이 전달 되는 객체를 Action이라고 하는데, Action은 대체로 액션 생성자(Action creator)에서 만들어집니다.

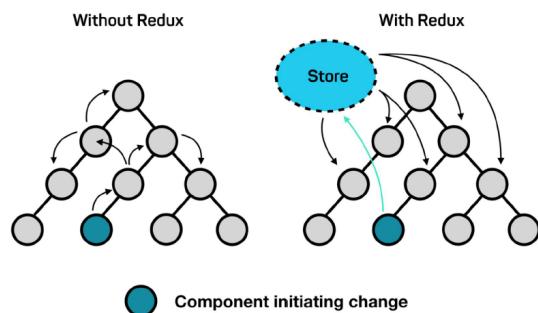
# Redux란?

Redux is a predictable state container for JavaScript apps.  
리액트는 Javascript 어플리케이션을 위한 **예측가능한** 상태 컨테이너  
이다.

## 기존 State 관리 방식의 문제점

Props는 부모로부터만 전파되기 때문에,  
모든 상태를 부모로부터 받는 것은 비효율  
적이고, 복잡한 구조일 경우 상태 예측이  
힘듦.

따라서, 앱의 전역상태 관리가 필요 →  
**Redux 도입의 필요성**



💡 단순히 View 기능만 하는  
Presentational Component  
에서는 state를 별도로 관리할  
필요가 없다.

## Redux 핵심 개념

### Action

- Action은 상태에 어떤 변경을 가했는지를 기록한 객체이자, 애플리케이션 상태를 업데이트하는 **유일한** 방법
- Properties
  - type: 업데이트 의도를 나타내는 속성. 의도가 명확하게 보일 수 있도록 고유한 값으로 설정(필수값)

- type 속성 이외의 속성은 임의로 작성 가능. 단, 상태 업데이트에 필요한 데이터로만 한정하여 보내는 것이 정석.

```
/**
 * list.action.js
 */

// action type 상수 정의
export const SEARCH = 'SEARCH';

// action 생성 함수 정의 (실제 동작할 로직을 여기에 작성)
export const search = keyword => {
  return {
    type: SEARCH,
    keyword // keyword: keyword와 같다.
  };
};
```

## Reducer

- reducer는 현재 상태를 가지고 action에 따라 새로운 state를 반환  
state는 절대로 undefined일 수 없기 때문에 항상 기본 state를 반환하거나 일종의 state를 리턴하는 형태로 구현 (undefined는 절대로 안된다)

```
/**
 * list.reducer.js
 */

import { SEARCH } from '../actions/list.actions';

// 초기상태 선언
const initialState = {
  keyword: ''
};

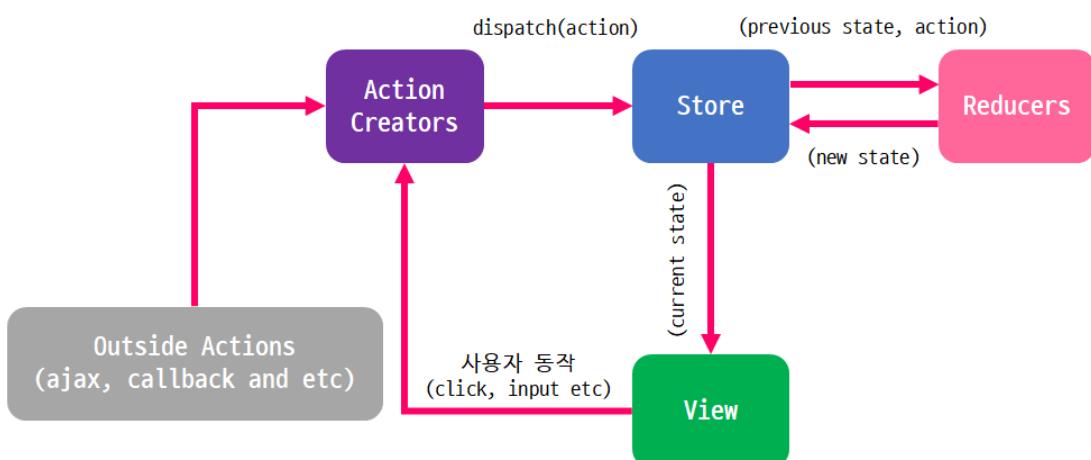
// Reducer 선언
const list = (state = initialState, action) => {
  switch (action.type) {
    case SEARCH:
      return {
        ...state,
        keyword: action.keyword
      };

    default:
      return state;
  }
};
```

```
export default list;
```

## Store

- 모든 state를 관리하기 위한 저장소
- 커다란 JSON 형태 정도로 이해해도 무방함
- 앱당 하나만 존재



## 실습

### 앱 생성 ( create-react-app )

```
> npx create-react-app practice-react-redux --use-npm
// --use-npm 옵션을 추가하지 않으면 create-react-app은 기본적으로 yarn 패키지 매니저로 설치된다.
// 
// cd practice-react-redux
```

```
// code .
// npm start
```

## 기존 방식으로 FilteredList 구현하기

### 1. components/FilteredList.js

```
import React, { Component } from 'react';
import List from './List';

export default class FilteredList extends Component {
  constructor() {
    super();

    this.state = {
      keyword: ''
    };
  }

  doSearch = e => {
    this.setState({ keyword: e.target.value });
  };

  render() {
    const { keyword } = this.state;
    const frameworks = ['React', 'Angular', 'Vue', 'Ember', 'Woowahan'];

    return (
      <div>
        <input type="text" onChange={e => this.doSearch(e)} />
        <List items={frameworks} keyword={keyword} />
      </div>
    );
  }
}
```

### 2. components/List.js

```
import React, { Component } from 'react';

export default class List extends Component {
  render() {
    const { items, keyword } = this.props;

    return (
      <ul>
        {items}
```

```
        .filter(item => item.indexOf(keyword) > -1)
        .map((item, i) => (
          <li key={i}>{item}</li>
        )));
      </ul>
    );
}
}
```

### 3. App.js

```
import React, { Component } from 'react';
import FilteredList from './components/FilteredList';

class App extends Component {
  render () {
    return (
      <div className="App">
        <FilteredList />
      </div>
    );
  }
}

export default App;
```

## Redux를 적용하여 리팩토링하기

### Redux 라이브러리 설치

```
> npm install --save redux react-redux
// npm install --save redux
// npm install --save react-redux
```

### Redux 코드 작성

1. 루트 폴더안에 redux라는 폴더를 생성한다.
2. redux 폴더 안에 actions와 reducers 폴더를 각각 생성한다.
3. App.js 파일 리팩토링

```

import React from 'react';
import FilteredList from './components/FilteredList';
import { createStore } from 'redux';
import { Provider } from 'react-redux';
import { reducers } from './redux/reducers';

// store 생성
const store = createStore(reducers);

function App() {
  return (
    // Provider 컴포넌트는 추후 connect 함수를 통해 특정 컴포넌트와
    // 그에 맞는 상태를 연결하기 위해 store를 제공한다.
    <Provider store={store}>
      <div className="App">
        <FilteredList />
      </div>
    </Provider>
  );
}

export default App;

// 기존 소스 -----

```

```

// function App() {
//   return (
//     <div className="App">
//       <FilteredList />
//     </div>
//   );
// }

// export default App;

```

#### 4. action 작성

- redux/actions/list.actions.js

```

// action type 상수 선언
export const SEARCH = 'SEARCH';

// 액션생성(action creator) 함수 선언 (실제 동작할 로직을 여기에 작성)
export const search = keyword => {
  return {
    type: SEARCH,
    keyword // keyword: keyword와 같다.
  };
};

```

- redux/actions/index.js

```
export * from './list.actions';
```

## 5. reducers 작성

- redux/reducers/list.reducer.js

```
import { SEARCH } from '../actions/list.action';

// 초기상태 선언
const initialState = {
  keyword: ''
};

// Reducer 선언
const list = (prevState = initialState, action) => {
  switch (action.type) {
    case SEARCH:
      return {
        ...prevState,
        keyword: action.keyword
      };

    default:
      return prevState;
  }
};

export default list;
```

- redux/reducers/index.js

```
import { combineReducers } from 'redux';
import list from './list.reducer';

/**
 * combineReducers => 여러 reducer들을 합쳐서 반환해주는 함수
 */
export const reducers = combineReducers({
  list
  // 여러 reducer들을 선언하면 된다.
});
```

## 6. FilteredList.js 파일 리팩토링

```
import React, { Component } from 'react';
import List from './List';
import { connect } from 'react-redux';
import { search } from '../redux/actions/list.action';

class FilteredList extends Component {
  constructor() {
    super();
  }

  render() {
    const { keyword, doSearch } = this.props;
    const frameworks = ['React', 'Angular', 'Vue', 'Ember', 'Woowahan'];

    return (
      <div>
        <input type="text" onChange={e => doSearch(e.target.value)} />
        <List items={frameworks} keyword={keyword} />
      </div>
    );
  }
}

/**
 * 컴포넌트의 props에 매핑시켜줄 상태를 정의
 * @param {*} state store에서 내려줄 state
 */
const mapStateToProps = currentState => {
  return {
    keyword: currentState.list.keyword
  };
};

/**
 * 컴포넌트의 props에 매핑하여 dispatch할 action들을 정의
 * @param {*} dispatch
 */
const mapDispatchToProps = dispatch => {
  return {
    doSearch: keyword => dispatch(search(keyword))
  };
};

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(FilteredList);

// 기준 소스 -----
// export default class FilteredList extends Component {
//   constructor() {
```

```

//      super();

//      this.state = {
//          keyword: ''
//      };

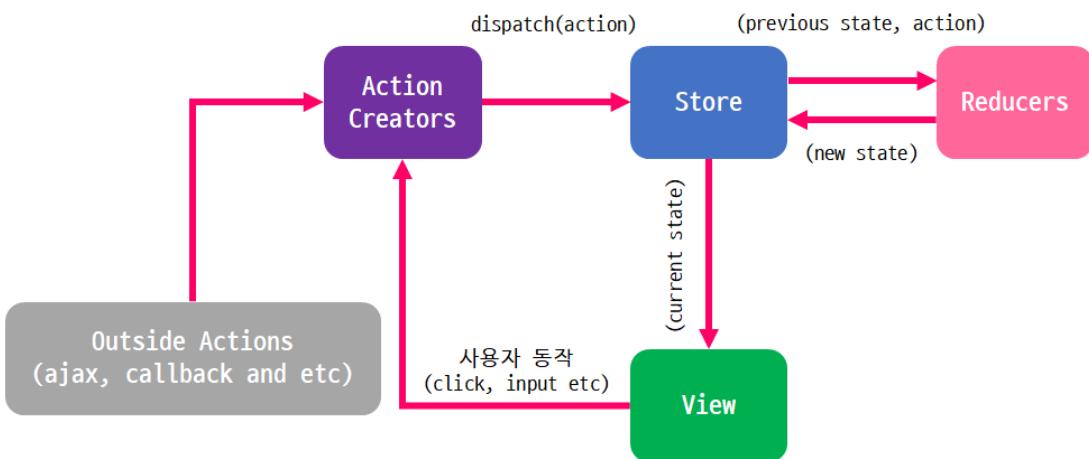
//      doSearch = e => {
//          this.setState({ keyword: e.target.value });
//      };

//      render() {
//          const { keyword } = this.state;
//          const frameworks = ['React', 'Angular', 'Vue', 'Ember', 'Woowahan'];

//          return (
//              <div>
//                  <input type="text" onChange={e => this.doSearch(e)} />
//                  <List items={frameworks} keyword={keyword} />
//              </div>
//          );
//      }
// }

```

## 마무리 - react-redux flow diagram



## Sample Code

<https://github.com/deokchanjung/react-redux-tutorial>

