

Redux 2021

- *<https://slides.com/woongjae/redux2021>*
- *<https://github.com/xid-mark/redux2021>*



Mark Lee

이웅재

Lead Software Engineer @ProtoPie

Microsoft MVP

TypeScript Korea User Group Organizer

Marktube (Youtube)

1. Redux Basic

1-1) Redux 개요 1-2) Action - 액션

1-3) Reducers - 리듀서 1-4) createStore

1-5) combineReducers 1-6) Redux 를 React 에 연결

Redux 개요

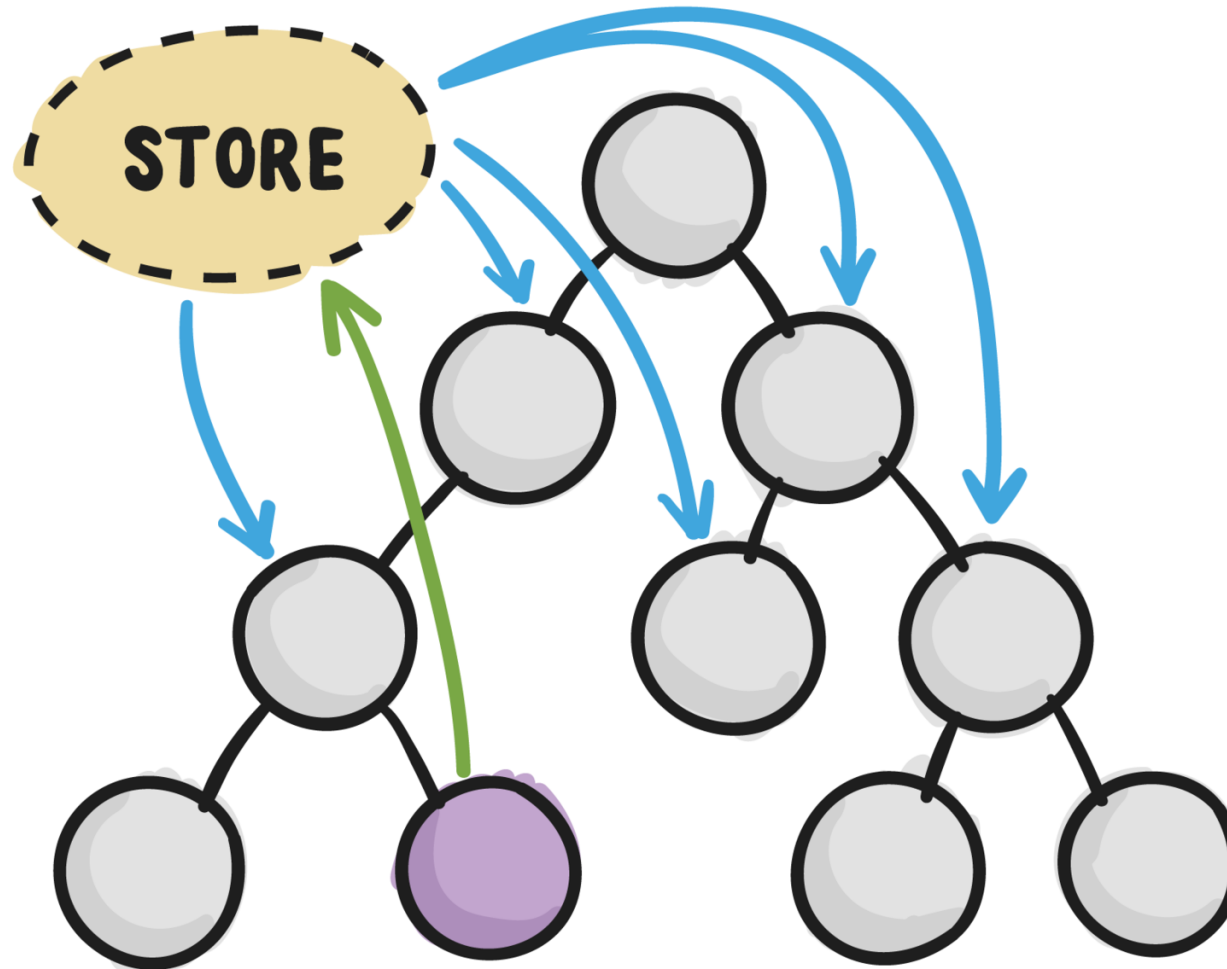
Component - Communication

<https://cloud.protopie.io/p/irg8jMXuGov?ui=false&mockup=false&scaleToFit=true>

Component - Communication - Redux

<https://cloud.protopie.io/p/Ycc3KrJvjgA?ui=false&mockup=false&scaleToFit=true>

Redux



"(1) 단일 스토어를 만드는 법" 과, "(2) 리액트에서 스토어 사용하는 법" 을 익히는 시간

- 단일 스토어다!
- [만들기] 단일 스토어 사용 준비하기
 - import **redux**
 - **액션**을 정의하고,
 - **액션**을 사용하는, **리듀서**를 만들고,
 - **리듀서**들을 합친다.
 - 최종 합쳐진 **리듀서**를 인자로, 단일 스토어를 만든다.
- [사용하기] 준비한 스토어를 리액트 컴포넌트에서 사용하기
 - import **react-redux**
 - connect 함수를 이용해서 컴포넌트에 연결


```
npx create-react-app redux-start
```

```
cd redux-start
```

```
npm i redux
```

Action - 액션

리덕스의 액션이란 ?

- 액션은 사실 그냥 객체 (**object**) 입니다.
- 두 가지 형태의 액션이 있습니다.
 - `{ type: 'TEST' }` // `payload` 없는 액션
 - `{ type: 'TEST', params: 'hello' }` // `payload` 있는 액션
- `type` 만이 필수 프로퍼티이며, `type` 은 문자열 입니다.

리덕스의 액션 생성자란 ?

```
function 액션생성자( ...args ) { return 액션; }
```

- 액션을 생성하는 함수를 "**액션 생성자 (Action Creator)**" 라고 합니다.
- 함수를 통해 액션을 생성해서, 액션 객체를 리턴해줍니다.
- *createTest('hello');* // { type: 'TEST', params: 'hello' } 리턴

리덕스의 액션은 어떤 일을 하나요 ?

- 액션 생성자를 통해 액션을 만들어 냅니다.
- 만들어낸 액션 객체를 리덕스 스토어에 보냅니다.
- 리덕스 스토어가 액션 객체를 받으면 스토어의 상태 값이 변경 됩니다.
- 변경된 상태 값에 의해 상태를 이용하고 있는 컴포넌트가 변경됩니다.
- 액션은 스토어에 보내는 일종의 인풋이라 생각할 수 있습니다.

액션을 준비하기 위해서는 ?

- 액션의 타입을 정의하여 변수로 빼는 단계
 - 강제는 아닙니다. (그러므로 안해도 됩니다.)
 - 그냥 타입을 문자열로 넣기에는 실수를 유발할 가능성이 큽니다.
 - 미리 정의한 변수를 사용하면, 스펠링에 주의를 덜 기울여도 됩니다.
- 액션 객체를 만들어 내는 함수를 만드는 단계
 - 하나의 액션 객체를 만들기 위해 하나의 함수를 만들어냅니다.
 - 액션의 타입은 미리 정의한 타입 변수로 부터 가져와서 사용합니다.

액션 준비 코드

```
// actions.js
```

```
// 액션의 type 정의
```

```
// 액션의 타입 => 액션 생성자 이름
```

```
// ADD_TODO => addTodo
```

```
export const ADD_TODO = 'ADD_TODO';
```

```
// 액션 생산자
```

```
// 액션의 타입은 미리 정의한 타입으로 부터 가져와서 사용하며,
```

```
// 사용자가 인자로 주지 않습니다.
```

```
export function addTodo(text) {
```

```
  return { type: ADD_TODO, text }; // { type: ADD_TODO, text: text }  
}
```

Reducers - 리듀서

리덕스의 리듀서란 ?

- 액션을 주면, 그 액션이 적용되어 달라진(안달라질수도...) 결과를 만들어 줌.
- 그냥 함수입니다.
 - *Pure Function*
 - *Immutable*
 - 왜용 ?
 - 리듀서를 통해 스테이트가 달라졌음을 리덕스가 인지하는 방식

리덕스의 리듀서란 ?

```
function 리듀서(previousState, action) {  
  return newState;  
}
```

- 액션을 받아서 스테이트를 리턴하는 구조
- 인자로 들어오는 *previousState* 와 리턴되는 *newState* 는 다른 참조를 가지도록 해야합니다.

리듀서 함수 만들기

```
// reducers.js
import { ADD_TODO } from './actions';

export function todoApp(previousState, action) {
  if (previousState === undefined) {
    return [];
  }
  if (action.type === ADD_TODO) {
    return [...previousState, { text: action.text }];
  }
  return previousState;
}
```

createStore

redux 로 부터 import

스토어를 만드는 함수

```
const store = createStore(리듀서);
```

- *createStore<S>(*
 reducer: Reducer<S>,
 preloadedState: S,
 enhancer?: StoreEnhancer<S>
): Store<S>;

스토어 만들기

```
// store.js
import { todoApp } from './reducers';
import { createStore } from 'redux';
import { addTodo } from './actions';

const store = createStore(todoApp);
console.log(store);

console.log(store.getState());

setTimeout(() => {
  store.dispatch(addTodo('hello'));
}, 1000);

export default store;
```

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

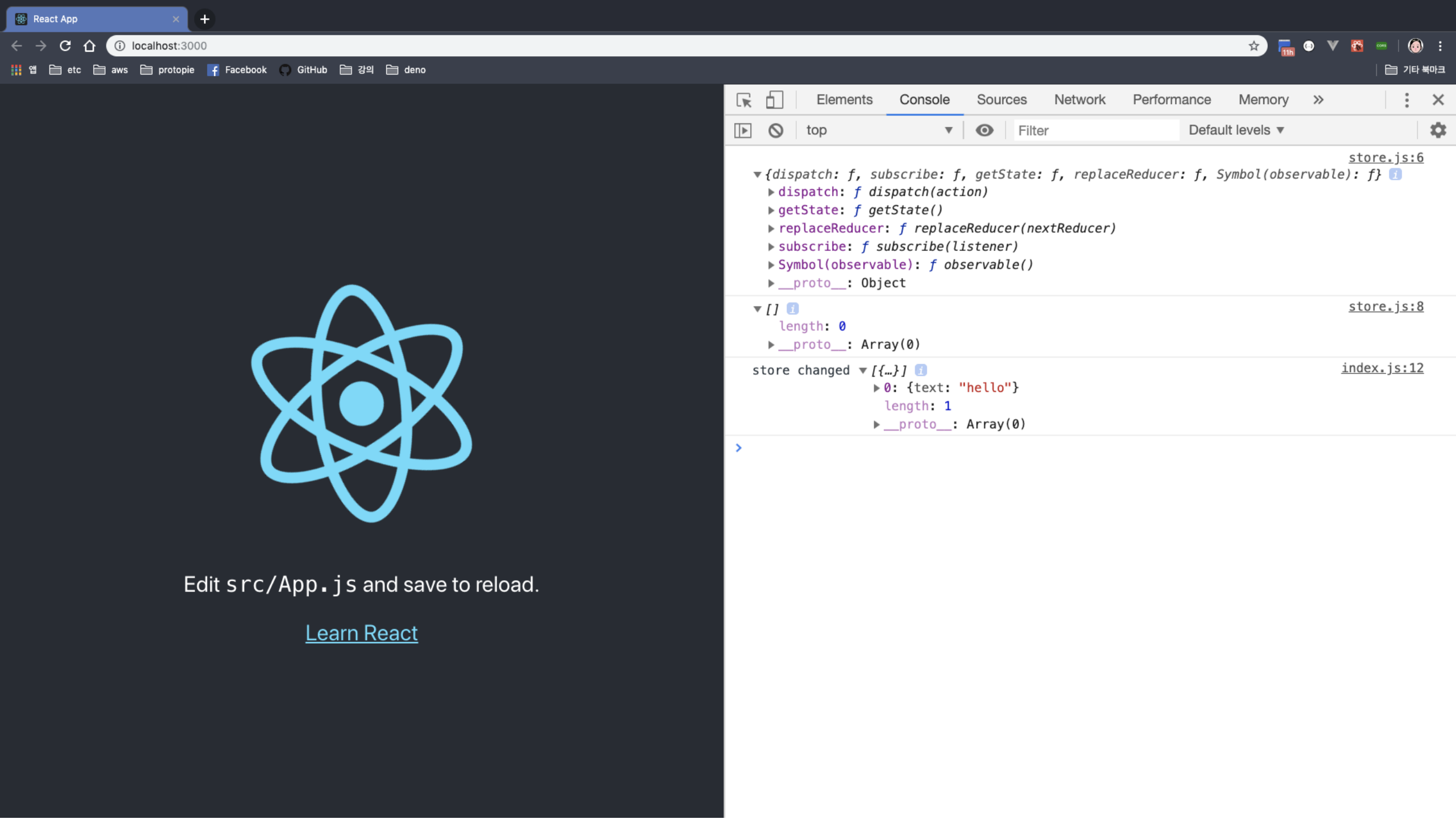
import store from './store';

store.subscribe(() => {
  const state = store.getState();

  console.log('store changed', state);
});

ReactDOM.render(<App />, document.getElementById('root'));

serviceWorker.unregister();
```



Edit `src/App.js` and save to reload.

[Learn React](#)

Developer tools console showing Redux state and actions:

- `store.js:6`
`{dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f}`
 - `dispatch: f dispatch(action)`
 - `getState: f getState()`
 - `replaceReducer: f replaceReducer(nextReducer)`
 - `subscribe: f subscribe(listener)`
 - `Symbol(observable): f observable()`
 - `__proto__: Object`
- `store.js:8`
`[]`
 - `length: 0`
 - `__proto__: Array(0)`
- `index.js:12`
`store changed`
 - `[{...}]`
 - `0: {text: "hello"}`
 - `length: 1`
 - `__proto__: Array(0)`

store

- *store.getState();*
- *store.dispatch(액션);, store.dispatch(액션생성자());*
- *const unsubscribe = store.subscribe(() => {});*
 - 리턴이 *unsubscribe* 라는 점 !
 - *unsubscribe();* 하면 제거
- *store.replaceReducer(다른리듀서);*

로직을 추가하기

action 을 정의하고, action 생성자를 만들고, reducer 를 수정

action 을 정의하고,

```
// actions.js
```

```
// 액션의 type 정의
```

```
// 액션의 타입 => 액션 생성자 이름
```

```
// ADD_TODO => addTodo
```

```
export const ADD_TODO = 'ADD_TODO';
```

```
export const COMPLETE_TODO = 'COMPLETE_TODO';
```

```
// 액션 생산자
```

```
// 액션의 타입은 미리 정의한 타입으로 부터 가져와서 사용하며,
```

```
// 사용자가 인자로 주지 않습니다.
```

```
export function addTodo(text) {
```

```
  return { type: ADD_TODO, text }; // { type: ADD_TODO, text: text }
```

```
}
```

action 생성자를 만들고,

```
// actions.js
```

```
// 액션의 type 정의
```

```
// 액션의 타입 => 액션 생성자 이름
```

```
// ADD_TODO => addTodo
```

```
export const ADD_TODO = 'ADD_TODO';
```

```
export const COMPLETE_TODO = 'COMPLETE_TODO';
```

```
// 액션 생산자
```

```
// 액션의 타입은 미리 정의한 타입으로 부터 가져와서 사용하며,
```

```
// 사용자가 인자로 주지 않습니다.
```

```
export function addTodo(text) {
```

```
  return { type: ADD_TODO, text }; // { type: ADD_TODO, text: text }
```

```
}
```

```
export function completeTodo(index) {
```

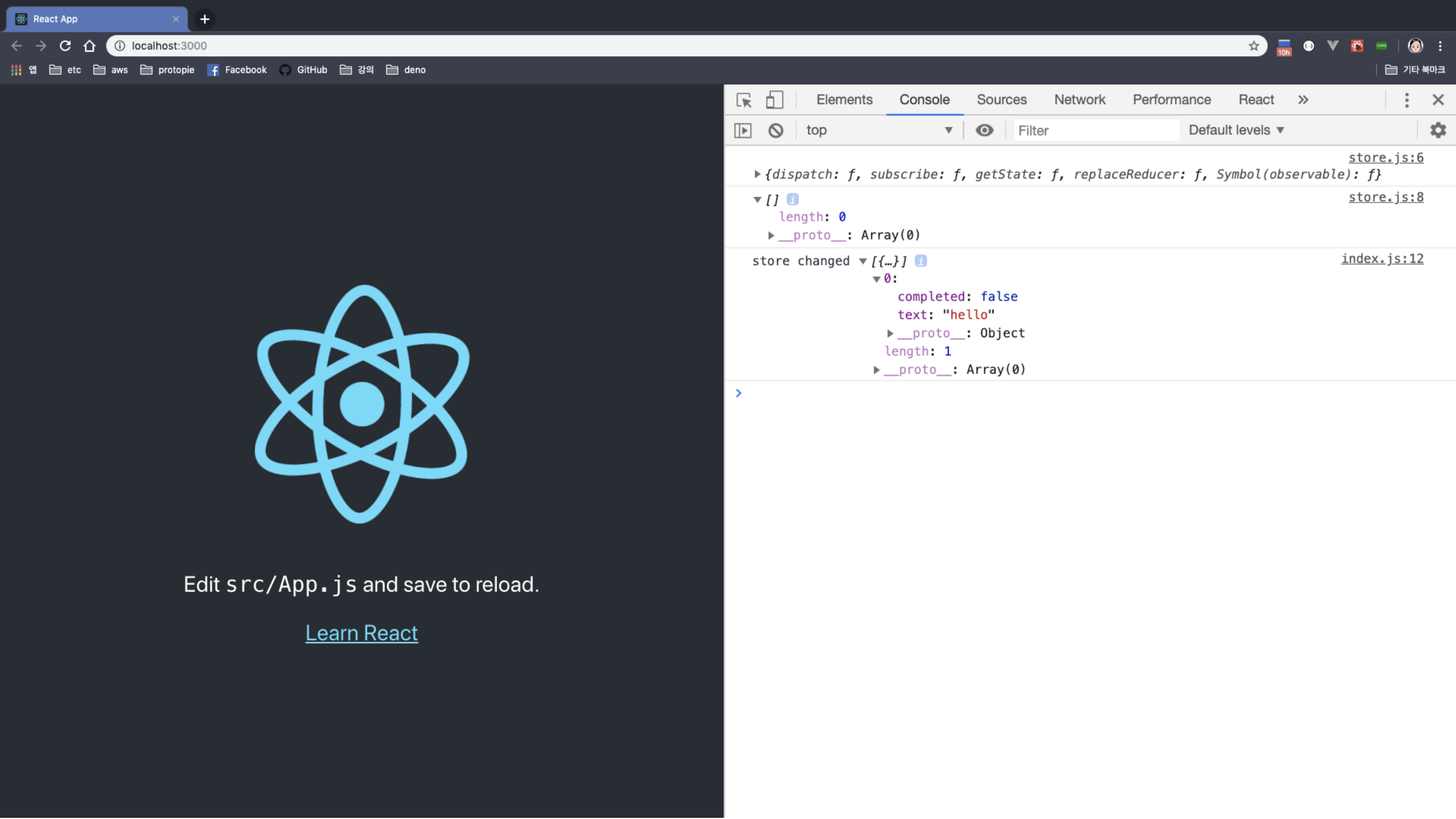
```
  return { type: COMPLETE_TODO, index }; // { type: COMPLETE_TODO, index: index }
```

```
}
```

reducer 를 수정

```
import { ADD_TODO, COMPLETE_TODO } from './actions';

export function todoApp(previousState, action) {
  if (previousState === undefined) {
    return [];
  }
  if (action.type === ADD_TODO) {
    return [...previousState, { text: action.text, completed: false }];
  }
  if (action.type === COMPLETE_TODO) {
    const newState = [];
    for (let i = 0; i < previousState.length; i++) {
      newState.push(
        i === action.index
          ? { ...previousState[i], completed: true }
          : { ...previousState[i] },
      );
    }
    return newState;
  }
  return previousState;
}
```



Edit `src/App.js` and save to reload.

[Learn React](#)

Elements Console Sources Network Performance React >>

top Filter Default levels

store.js:6

▶ {dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f}

store.js:8

▼ []
length: 0
▶ __proto__: Array(0)

index.js:12

store changed ▼ [{...}]
▼ 0:
completed: false
text: "hello"
▶ __proto__: Object
length: 1
▶ __proto__: Array(0)

>

dispatch

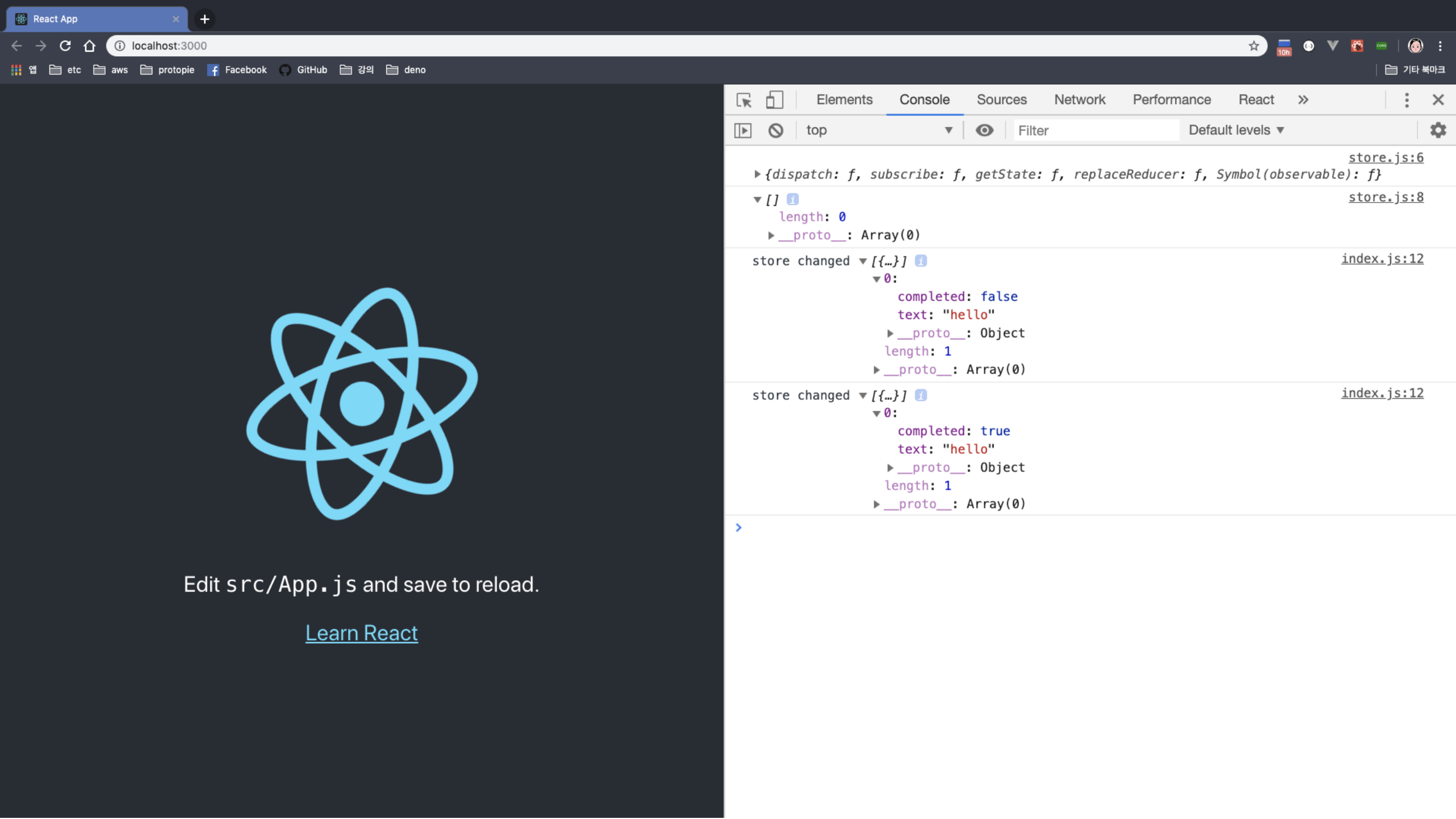
```
// store.js
import { todoApp } from './reducers';
import { createStore } from 'redux';
import { addTodo, completeTodo } from './actions';

const store = createStore(todoApp);
console.log(store);

console.log(store.getState());

setTimeout(() => {
  store.dispatch(addTodo('hello'));
  setTimeout(() => {
    store.dispatch(completeTodo(0));
  }, 1000);
}, 1000);

export default store;
```



Edit `src/App.js` and save to reload.

[Learn React](#)

React App

localhost:3000

Elements Console Sources Network Performance React

top Filter Default levels

store.js:6
▶ {dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f}

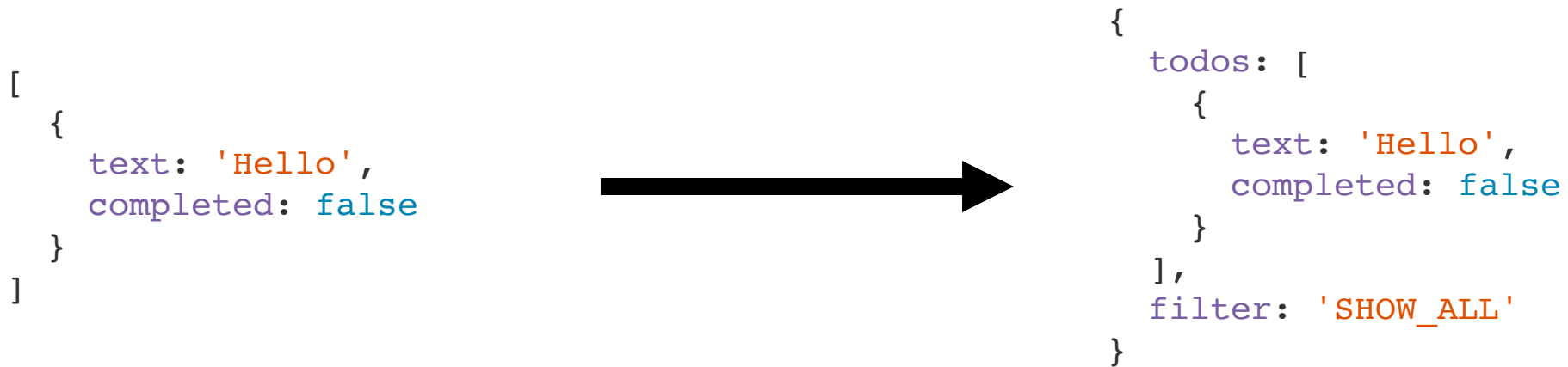
store.js:8
▼ []
length: 0
▶ __proto__: Array(0)

index.js:12
store changed ▼ [{...}]
▼ 0:
completed: false
text: "hello"
▶ __proto__: Object
length: 1
▶ __proto__: Array(0)

index.js:12
store changed ▼ [{...}]
▼ 0:
completed: true
text: "hello"
▶ __proto__: Object
length: 1
▶ __proto__: Array(0)

>

애플리케이션이 커지면, *state* 가 복잡해진다.

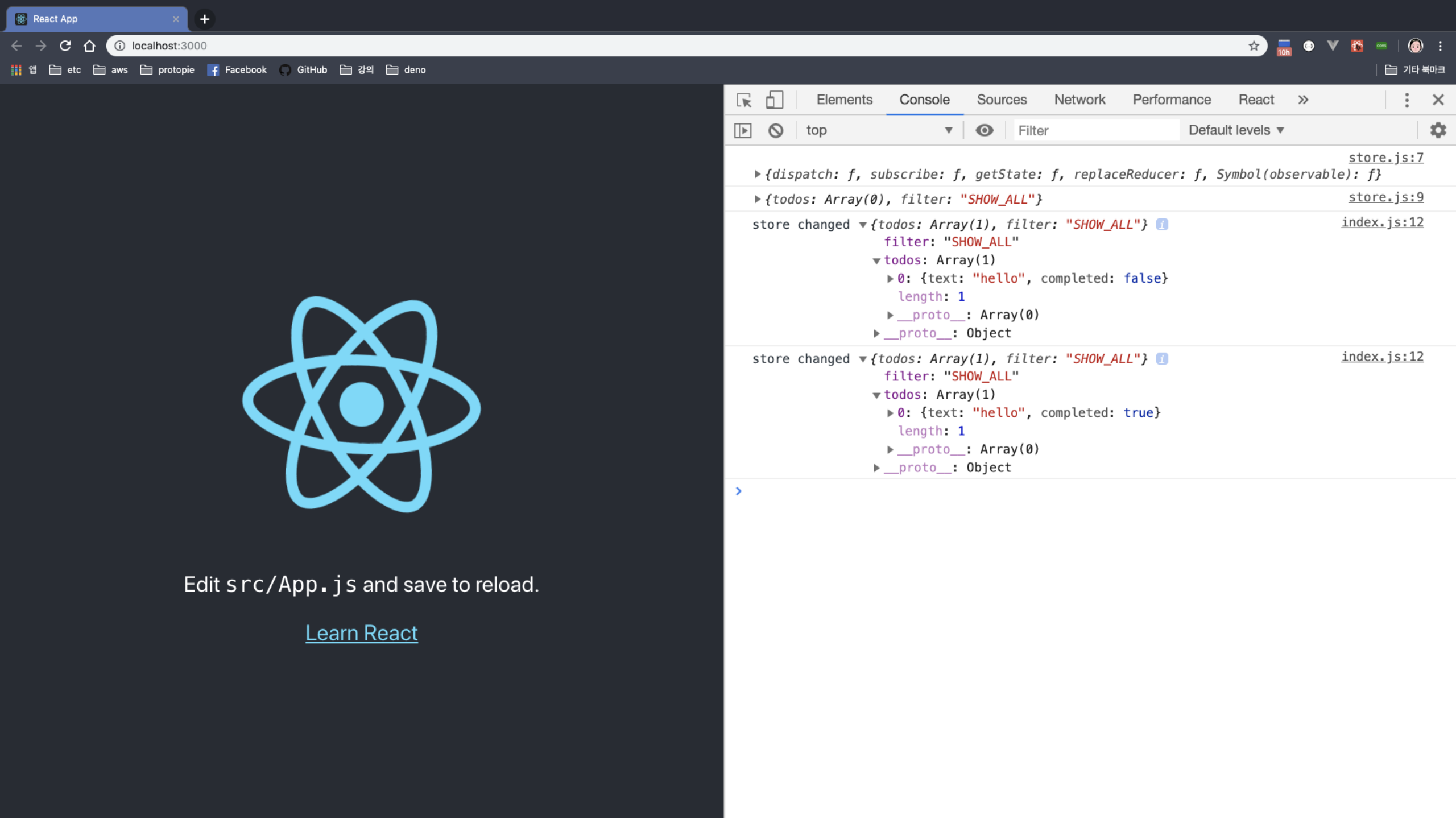


- 리듀서를 크게 만들고, *state* 를 변경하는 모든 로직을 담을 수도 있습니다.
- 리듀서를 분할해서 만들고, 합치는 방법을 사용할 수 있습니다.
 - *todos* 만 변경하는 **액션들**을 처리하는 A 라는 리듀서 함수를 만들고,
 - *filter* 만을 변경하는 **액션들**을 처리하는 B 라는 리듀서 함수를 만들고,
 - A 와 B 를 합침.

한번에 다하는 리듀서

```
import { ADD_TODO, COMPLETE_TODO } from './actions';

export function todoApp(previousState, action) {
  if (previousState === undefined) {
    return { todos: [], filter: 'SHOW_ALL' };
  }
  if (action.type === ADD_TODO) {
    return {
      todos: [...previousState.todos, { text: action.text, completed: false }],
      filter: previousState.filter,
    };
  }
  if (action.type === COMPLETE_TODO) {
    const todos = [];
    for (let i = 0; i < previousState.todos.length; i++) {
      todos.push(
        i === action.index
          ? { ...previousState.todos[i], completed: true }
          : { ...previousState.todos[i] },
      );
    }
    return { todos, filter: previousState.filter };
  }
  return previousState;
}
```



Edit `src/App.js` and save to reload.

[Learn React](#)

Elements Console Sources Network Performance React >>

top Filter Default levels

store.js:7

▶ {dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f}

store.js:9

▶ {todos: Array(0), filter: "SHOW_ALL"}

index.js:12

store changed ▼ {todos: Array(1), filter: "SHOW_ALL"} ⓘ

filter: "SHOW_ALL"

▼ todos: Array(1)

▶ 0: {text: "hello", completed: false}

length: 1

▶ __proto__: Array(0)

▶ __proto__: Object

index.js:12

store changed ▼ {todos: Array(1), filter: "SHOW_ALL"} ⓘ

filter: "SHOW_ALL"

▼ todos: Array(1)

▶ 0: {text: "hello", completed: true}

length: 1

▶ __proto__: Array(0)

▶ __proto__: Object

>

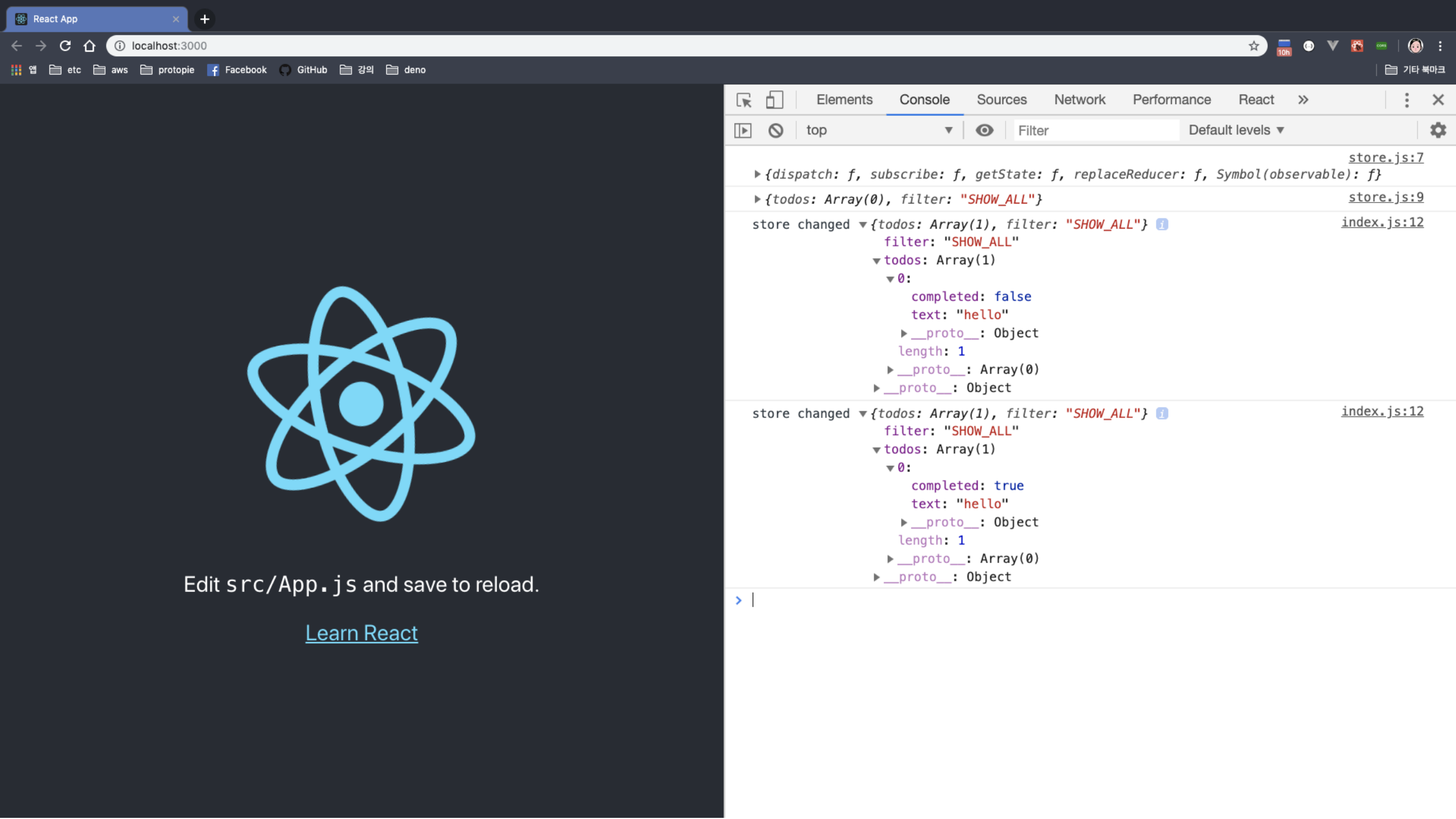
리듀서 분리

```
export function todos(previousState, action) {
  if (previousState === undefined) {
    return [];
  }
  if (action.type === ADD_TODO) {
    return [...previousState.todos, { text: action.text, completed: false }];
  }
  if (action.type === COMPLETE_TODO) {
    const newState = [];
    for (let i = 0; i < previousState.length; i++) {
      newState.push(
        i === action.index
          ? { ...previousState[i], completed: true }
          : { ...previousState[i] },
      );
    }
    return newState;
  }
  return previousState;
}

export function filter(previousState, action) {
  if (previousState === undefined) {
    return 'SHOW_ALL';
  }
  return previousState;
}
```

리듀서 합치기

```
export function todoApp(previousState = {}, action) {  
  return {  
    todos: todos(previousState.todos, action),  
    filter: filter(previousState.filter, action),  
  };  
}
```



Edit `src/App.js` and save to reload.

[Learn React](#)

Elements Console Sources Network Performance React >>

top Filter Default levels

store.js:7

▶ {dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f}

store.js:9

▶ {todos: Array(0), filter: "SHOW_ALL"}

index.js:12

store changed ▼ {todos: Array(1), filter: "SHOW_ALL"} ⓘ

filter: "SHOW_ALL"

▼ todos: Array(1)

▼ 0:

completed: false

text: "hello"

▶ __proto__: Object

length: 1

▶ __proto__: Array(0)

▶ __proto__: Object

index.js:12

store changed ▼ {todos: Array(1), filter: "SHOW_ALL"} ⓘ

filter: "SHOW_ALL"

▼ todos: Array(1)

▼ 0:

completed: true

text: "hello"

▶ __proto__: Object

length: 1

▶ __proto__: Array(0)

▶ __proto__: Object

> |

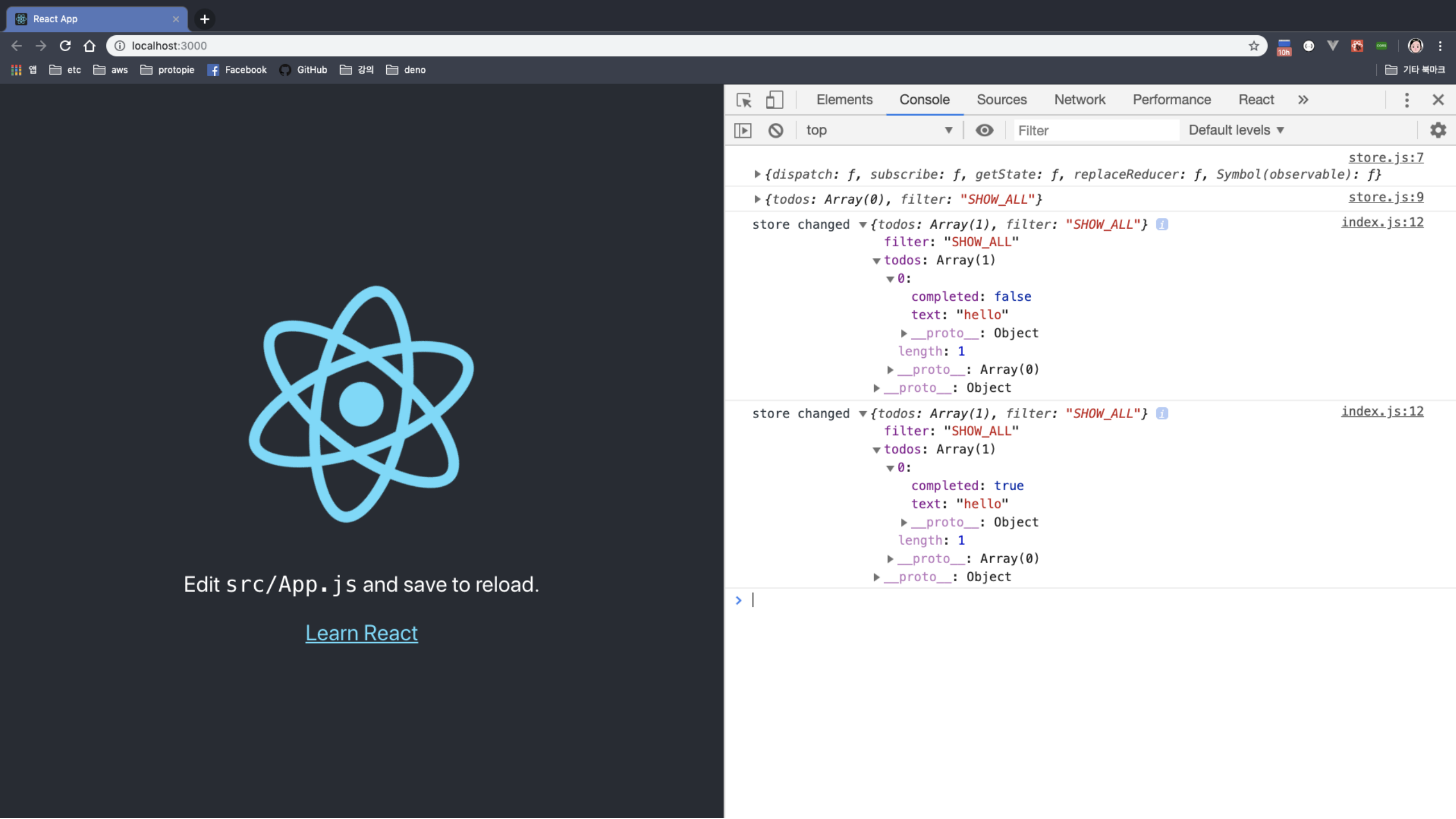
combineReducers

redux 로 부터 import

리덕스에서 제공하는 combineReducers 사용

```
import { combineReducers } from 'redux';
```

```
const todoApp = combineReducers({  
  todos,  
  filter,  
});
```



Redux 를 React 에 연결

react-redux 안쓰고 연결하기

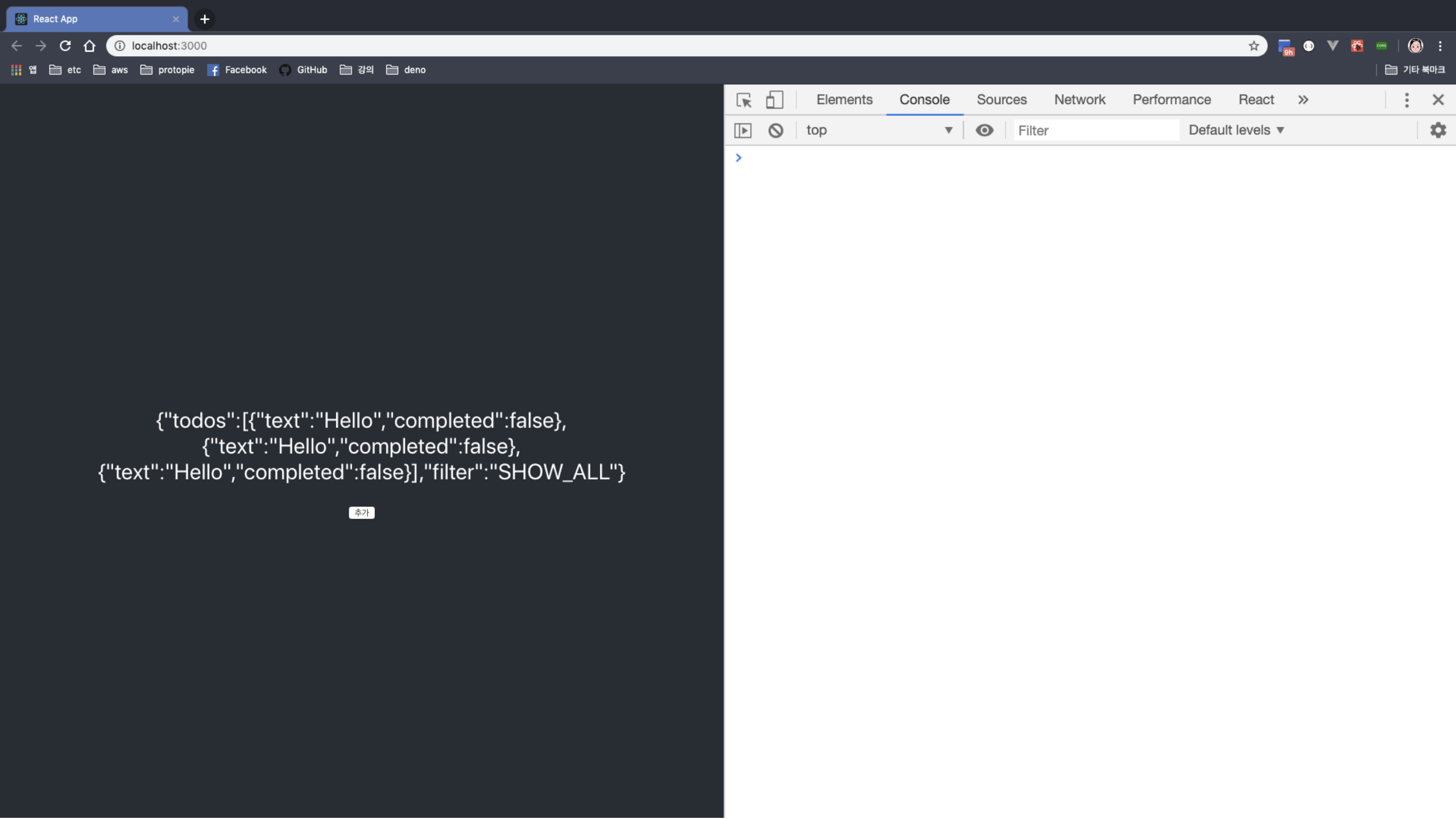
단일 **store** 를 만들고,
subscribe 와 **getState** 를 이용하여,
변경되는 **state** 데이터를 얻어,
props 로 계속 아래로 전달

- **componentDidMount** - **subscribe**
- **componentWillUnmount** - **unsubscribe**

```
// App.js
import React, { useState, useEffect } from 'react';
import './App.css';
import { addTodo } from './actions';

function App({ store }) {
  const [state, setState] = useState(store.getState());
  useEffect(() => {
    const unsubscribe = store.subscribe(() => {
      setState(store.getState());
    });
    return () => {
      unsubscribe();
    };
  });
  return (
    <div className="App">
      <header className="App-header">
        <p>{JSON.stringify(state)}</p>
        <button
          onClick={() => {
            store.dispatch(addTodo('Hello'));
          }}
        >
          추가
        </button>
      </header>
    </div>
  );
}
```

```
export default App;
```



Context

context를 이용하면 단계마다 일일이 props를 넘겨주지 않고도 컴포넌트 트리 전체에 데이터를 제공할 수 있습니다.

일반적인 React 애플리케이션에서 데이터는 위에서 아래로 (즉, 부모로부터 자식에게) props를 통해 전달되지만, 애플리케이션 안의 여러 컴포넌트들에 전해줘야 하는 props의 경우 (예를 들면 선호 로케일, UI 테마) 이 과정이 번거로울 수 있습니다. context를 이용하면, 트리 단계마다 명시적으로 props를 넘겨주지 않아도 많은 컴포넌트가 이러한 값을 공유하도록 할 수 있습니다.

- 언제 context를 써야 할까
- context를 사용하기 전에 고려할 것
- API
 - React.createContext
 - Context.Provider
 - Class.contextType
 - Context.Consumer
- 예시
 - 값이 변하는 context
 - 하위 컴포넌트에서 context 업데이트하기
 - 여러 context 구독하기
- 주의사항
- 예전 API

설치 ▾

주요 개념 ▾

고급 안내서 ▲

접근성

코드 분할

Context

Error Boundary

Ref 전달하기

Fragment

고차 컴포넌트

다른 라이브러리와 통합하기

JSX 이해하기

성능 최적화

Portal

ES6 없이 사용하는 React

JSX 없이 사용하는 React

재조정 (Reconciliation)

Ref와 DOM

Render Props

정적 타입 검사

Strict Mode

PropTypes를 사용한 타입 검사

비제어 컴포넌트

웹 컴포넌트

API 참고서 ▾

Context API
useContext

```
import React from 'react';

const ReduxContext = React.createContext();

export default ReduxContext;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App2';
import * as serviceWorker from './serviceWorker';
import store from './store';
import ReduxContext from './context';
```

```
ReactDOM.render(
  <ReduxContext.Provider value={store}>
    <App />
  </ReduxContext.Provider>,
  document.getElementById('root'),
);
```

```
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

```

import React, { useContext } from 'react';
import './App.css';
import { addTodo } from './actions';
import ReduxContext from './context';
import Button from './Button';

class App extends React.Component {
  static contextType = ReduxContext;
  _unsubscribe;
  state = this.context.getState();
  componentDidMount() {
    this._unsubscribe = this.context.subscribe(() => {
      this.setState(this.context.getState());
    });
  }
  componentWillUnmount() {
    this._unsubscribe();
  }
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <p>{JSON.stringify(this.state)}</p>
          <Button />
        </header>
      </div>
    );
  }
}

export default App;

```

```

import React, { useContext } from 'react';
import { addTodo } from './actions';
import ReduxContext from './context';

export default function Button() {
  const store = useContext(ReduxContext);
  return (
    <button
      onClick={() => {
        store.dispatch(addTodo('Hello'));
      }}
    >
      추가
    </button>
  );
}

```


Redux 를 React 에 연결

react-redux 쓰고 연결하기

react-redux

- *Provider* 컴포넌트를 제공해줍니다.
- *connect* 함수를 통해 "컨테이너"를 만들어줍니다.
 - 컨테이너는 스토어의 **state** 와 **dispatch(액션)** 를 연결한 컴포넌트에 *props* 로 넣어주는 역할을 합니다.
 - 그렇다면 필요한 것은 ?
 - 어떤 **state** 를 어떤 *props* 에 연결할 것인지에 대한 정의
 - 어떤 **dispatch(액션)** 을 어떤 *props* 에 연결할 것인지에 대한 정의
 - 그 *props* 를 보낼 컴포넌트를 정의

```
npm i react-redux
```

Provider Component from react-redux

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App3';
import * as serviceWorker from './serviceWorker';
import store from './store';
import { Provider } from 'react-redux';
```

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById( 'root' ),
);
```

```
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Consumer from react-redux

```
import React, { useContext, useEffect, useState } from 'react';
import { ReactReduxContext } from 'react-redux';
import './App.css';
import { addTodo } from './actions';
import Button from './Button';
```

```
class App extends React.Component {
  render() {
    console.log(this.props);
    return (
      <div className="App">
        <header className="App-header">
          <p>{JSON.stringify(this.props.todos)}</p>
          <Button add={this.props.add} />
        </header>
      </div>
    );
  }
}
```

```
import React from 'react';
```

```
export default function Button({ add }) {
  return <button onClick={() => add('hello')}>추가</button>;
}
```

```
function AppContainer(props) {
  const { store } = useContext(ReactReduxContext);
  const [state, setState] = useState(store.getState());
  function add(text, dispatch) {
    console.log(text, dispatch);
    dispatch(addTodo(text));
  }
  useEffect(() => {
    const _unsubscribe = store.subscribe(() => {
      setState(store.getState());
    });
    return () => {
      _unsubscribe();
    };
  });
  return (
    <App
      {...props}
      todos={state.todos}
      add={text => add(text, store.dispatch)}
    />
  );
}
```

```
export default AppContainer;
```

connect function from react-redux

```
import React from 'react';
import './App.css';
import { addTodo } from './actions';
import { connect } from 'react-redux';
import Button from './Button';

class App extends React.Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <p>{JSON.stringify(this.props.todos)}</p>
          <Button add={this.props.add} />
        </header>
      </div>
    );
  }
}
```

```
import React from 'react';

export default function Button({ add }) {
  return <button onClick={() => add('hello')}>추가</button>;
}
```

```
const mapStateToProps = state => {
  return { todos: state.todos };
};

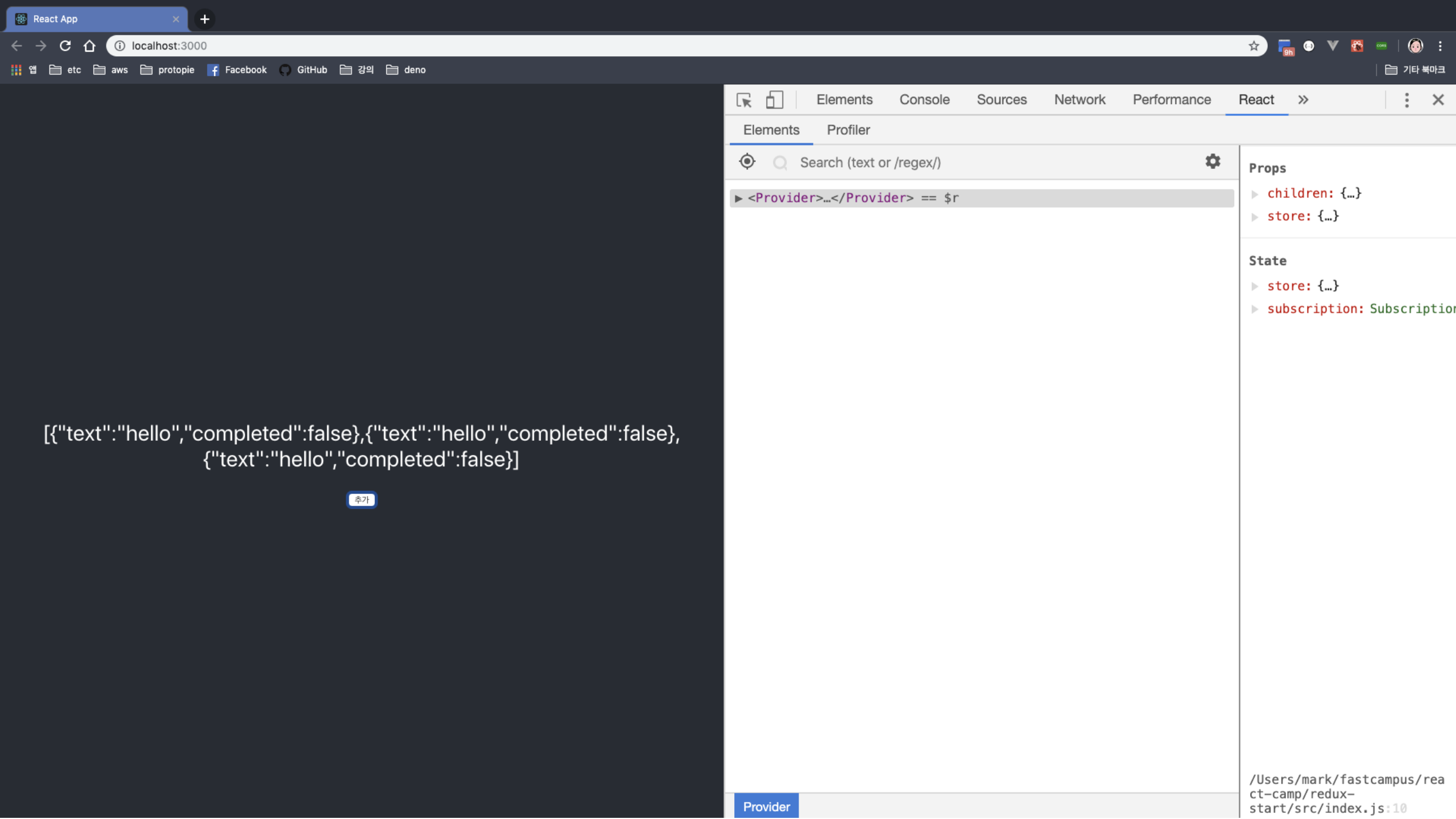
const mapDispatchToProps = dispatch => {
  return {
    add: text => {
      dispatch(addTodo(text));
    },
  };
};

const AppContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(App);

export default AppContainer;
```

mapStateToProps, mapDispatchToProps

```
const mapStateToProps = state => {  
  return { todos: state.todos };  
};  
  
const mapDispatchToProps = dispatch => {  
  return {  
    add: text => {  
      dispatch(addTodo(text));  
    },  
  };  
};
```



Git Repository

<https://github.com/xid-mark/redux-start>

2. Redux Advanced (1)

2-1) Async Action with Redux

2-2) 리덕스 미들웨어

2-3) redux-devtools

2-4) redux-thunk

2-5) redux-promise-middleware

Async Action with Redux

비동기 작업을 어디서 하느냐 ? 가 쥔 중요

- 액션을 분리합니다.
 - *Start*
 - *Success*
 - *Fail*
 - ... 등등
- ***dispatch*** 를 할때 해줍니다.
 - 당연히 리듀서는 동기적인 것 => *Pure*
 - *dispatch* 도 동기적인 것

비동기 처리를 위한 액션 추가

// 액션 정의

```
export const BOOKS_START = 'BOOKS_START';  
export const BOOKS_SUCCESS = 'BOOKS_SUCCESS';  
export const BOOKS_FAIL = 'BOOKS_FAIL';
```

// 액션 생성자 함수

```
export const booksStartAction = () => ({ type: BOOKS_START });  
export const booksSuccessAction = (books) => ({ type: BOOKS_SUCCESS, books });  
export const booksFailAction = (error) => ({ type: BOOKS_FAIL, error });
```

비동기 처리가 컴포넌트에 있는 경우

```
// src/components/Books.jsx
import React, { useEffect } from 'react';
import axios from 'axios';
import { withRouter } from 'react-router-dom';
import Book from './Book';

const Books = ({ token, books, loading, error, booksStart, booksSuccess, booksFail }) => {
  useEffect(() => {
    if (error === null) return;
  }, [error]);

  useEffect(() => {
    async function getBooks() {
      booksStart();
      await sleep(1000);
      try {
        const res = await axios.get('https://api.marktube.tv/v1/book', {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });
        booksSuccess(res.data);
      } catch (error) {
        booksFail(error);
      }
    }

    getBooks();
  }, [token, booksStart, booksSuccess, booksFail]);

  return (
    <>
      {books.map((book) => (
        <Book key={book.id} book={book} />
      ))}
    </>
  )
}
```

비동기 처리가 컴포넌트에 있는 경우

```
import React from 'react';
import Books from '../components/Books';
import { useDispatch, useSelector } from 'react-redux';
import { booksStartAction, booksSuccessAction, booksFailAction } from '../redux/actions';
import axios from 'axios';

const BooksContainer = ({ token }) => {
  // const token = useSelector((state) => state.auth.token);
  const books = useSelector((state) => state.books.books);
  const loading = useSelector((state) => state.books.loading);
  const error = useSelector((state) => state.books.error);
  const dispatch = useDispatch();
  const booksStart = dispatch(booksStartAction());
  const booksSuccess = (books) => dispatch(booksSuccessAction(books));
  const booksFail = (error) => dispatch(booksFailAction(error));
  async function getBooks() {
    booksStart();
    await sleep(1000);
    try {
      const res = await axios.get('https://api.marktube.tv/v1/book', {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });
      booksSuccess(res.data);
    } catch (error) {
      booksFail(error);
    }
  }
  return (
    <Books books={books} loading={loading} error={error} getBooks={getBooks} />
  );
};

export default BooksContainer;
```

Books.jsx

```
// src/components/Books.jsx
import React, { useEffect } from 'react';
import Book from './Book';

const Books = ({ books, loading, error, getBooks }) => {
  useEffect(() => {
    if (error === null) return;
  }, [error]);

  useEffect(() => {
    getBooks(); // 컨테이너로 로직을 옮겼음.
  }, [getBooks]);

  return (
    <>
      {books.map((book) => (
        <Book key={book.bookId} {...book} />
      ))}
    </>
  );
};

export default Books;
```


리덕스 미들웨어

Middleware

redux.js.org/advanced/middleware/

☆

2d

11

▼

🔌

🌱

🔄

👤

⋮

Redux

Getting Started

API

FAQ

GitHub

Need help?

🔍 Search

Introduction

Basic Tutorial

Advanced Tutorial

Advanced Tutorial: Intro

Async Actions

Async Flow

Middleware

Usage with React Router

Example: Reddit API

Next Steps

Recipes

FAQ

Style Guide

Other

API Reference

Redux Toolkit

Middleware

You've seen middleware in action in the [Async Actions](#) example. If you've used server-side libraries like [Express](#) and [Koa](#), you were also probably already familiar with the concept of *middleware*. In these frameworks, middleware is some code you can put between the framework receiving a request, and the framework generating a response. For example, Express or Koa middleware may add CORS headers, logging, compression, and more. The best feature of middleware is that it's composable in a chain. You can use multiple independent third-party middleware in a single project.

Redux middleware solves different problems than Express or Koa middleware, but in a conceptually similar way. **It provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.** People use Redux middleware for logging, crash reporting, talking to an asynchronous API, routing, and more.

This article is divided into an in-depth intro to help you grok the concept, and [a few practical examples](#) to show the power of middleware at the very end. You may find it helpful to switch back and forth between them, as you flip between feeling bored and inspired.

Understanding Middleware

While middleware can be used for a variety of things, including asynchronous API calls, it's really important that you understand where it comes from. We'll guide you through the thought process leading to middleware, by using logging and crash reporting as examples.

Problem: Logging

One of the benefits of Redux is that it makes state changes predictable and transparent. Every time an action is dispatched, the new state is computed and saved. The state cannot change by itself, it can only change as a consequence of a specific action.

Wouldn't it be nice if we logged every action that happens in the app, together with the state computed after it? When something goes wrong, we can look back at our log, and figure out which action corrupted the state.

```
▼ ADD_TODO
  ❶ dispatching: Object {type: "ADD_TODO", text: "Use Redux"}
    next state: ► Object {visibilityFilter: "SHOW_ALL", todos: Array[1]}
▼ ADD_TODO
  ❶ dispatching: Object {type: "ADD_TODO", text: "Learn about middleware"}
    next state: ► Object {visibilityFilter: "SHOW_ALL", todos: Array[2]}
▼ COMPLETE_TODO
  ❶ dispatching: Object {type: "COMPLETE_TODO", index: 0}
```

Understanding Middleware

Problem: Logging

Attempt #1: Logging Manually

Attempt #2: Wrapping Dispatch

Attempt #3: Monkeypatching Dispatch

Problem: Crash Reporting

Attempt #4: Hiding Monkeypatching

Attempt #5: Removing Monkeypatching

Attempt #6: Naïvely Applying the Middleware

The Final Approach

Seven Examples

리덕스 미들웨어

- 미들웨어가 "디스패치" 의 앞뒤에 코드를 추가할수 있게 해줍니다.
- 미들웨어가 여러개면 미들웨어가 "순차적으로" 실행됩니다.
- 두 단계가 있습니다.
 - 스토어를 만들때, 미들웨어를 설정하는 부분
 - {createStore, applyMiddleware} from redux
 - 디스패치가 호출될때 실제로 미들웨어를 통과하는 부분
- dispatch 메소드를 통해 store로 가고 있는 액션을 가로채는 코드

리덕스 미들웨어

```
function middleware1(store) {  
  return next => {  
    console.log('middleware1', 1);  
    return action => {  
      console.log('middleware1', 2);  
      const returnValue = next(action);  
      console.log('middleware1', 3);  
      return returnValue;  
    };  
  };  
}
```

```
function middleware2(store) {  
  return next => {  
    console.log('middleware2', 1);  
    return action => {  
      console.log('middleware2', 2);  
      const returnValue = next(action);  
      console.log('middleware2', 3);  
      return returnValue;  
    };  
  };  
}
```

applyMiddleware(함수1, 함수2, ...)

```
import { createStore, applyMiddleware } from 'redux';

function middleware1(store) {...}

function middleware2(store) {...}

const store = createStore(reducer, applyMiddleware(middleware1, middleware2));
```

React App

localhost:3000

앱etcawsprotopieFacebookGitHub강의deno

<http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791186710302&orderClick=LEB&Kc=>

delete

책 제목

감상평

지은이

[구매링크](#)

delete

책 제목

감상평

지은이

[구매링크](#)

delete

TS

ZZang

ZZang

[ZZang](#)

delete

모던 자바스크립트 입문

모던하균요1

서재원

<http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791160504439&orderClick=LET&Kc=>

delete

11th

기타 북마크

ElementsConsoleSources

top

FilterDefault levels

middleware2 1index.js:24

middleware1 1index.js:12

dispatch beforeBooksContainer.jsx:11

middleware1 2index.js:14

middleware2 2index.js:26

RECEIVE_BOOKSbooks.js:9

middleware2 3index.js:28

middleware1 3index.js:16

dispatch afterBooksContainer.jsx:13

▶ ./src/components/Book.jsxwebpackHotDevClient.js:120

Line 41: Using target="_blank" without rel="noopener

noreferrer" is a security risk: see https://mathiasbynens.gi

thub.io/rel-noopener react/jsx-no-target-blank

111 222 333 444BookAddForm.jsx:109

dispatch beforeBooksContainer.jsx:11

middleware1 2index.js:14

middleware2 2index.js:26

RECEIVE_BOOKSbooks.js:9

middleware2 3index.js:28

middleware1 3index.js:16

dispatch afterBooksContainer.jsx:13

삭제 449Book.jsx:15

middleware1 2index.js:14

middleware2 2index.js:26

DELETE_BOOKbooks.js:15

middleware2 3index.js:28

middleware1 3index.js:16

>

middleware 에서 store 접근

```
function middleware1(store) {  
  
  return next => {  
    console.log('middleware1', 1, store.getState());  
  
    return action => {  
      console.log('middleware1', 2, store.getState());  
      const returnValue = next(action);  
      console.log('middleware1', 3, store.getState());  
  
      return returnValue;  
    };  
  };  
  
};  
  
}
```

React App

localhost:3000

앱etcawsprotopieFacebookGitHub강의deno

http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791186710302&orderClick=LEB&Kc=

delete

책 제목

감상평

지은이

구매링크

delete

책 제목

감상평

지은이

구매링크

delete

TS

ZZang

ZZang

ZZang

delete

모던 자바스크립트 입문

모던하균요1

서재원

http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791160504439&orderClick=LET&Kc=

delete

Elements

Console

Sources

1

top

Filter

Default levels

middleware2 1

index.js:24

middleware1 1

{books: Array(0)}

books: []

__proto__: Object

dispatch before

BooksContainer.jsx:11

middleware1 2

{books: Array(0)}

books: []

__proto__: Object

middleware2 2

index.js:26

RECEIVE_BOOKS

books.js:9

middleware2 3

index.js:28

middleware1 3

index.js:16

{books: Array(8)}

books: Array(8)

0: {bookId: 1, ownerId: "7d26db27-168c-4c6a-bd9a-9e206..."}

1: {bookId: 2, ownerId: "7d26db27-168c-4c6a-bd9a-9e206..."}

2: {bookId: 81, ownerId: "7d26db27-168c-4c6a-bd9a-9e20..."}

3: {bookId: 101, ownerId: "7d26db27-168c-4c6a-bd9a-9e2..."}

4: {bookId: 111, ownerId: "7d26db27-168c-4c6a-bd9a-9e2..."}

5: {bookId: 112, ownerId: "7d26db27-168c-4c6a-bd9a-9e2..."}

6: {bookId: 113, ownerId: "7d26db27-168c-4c6a-bd9a-9e2..."}

7: {bookId: 144, ownerId: "7d26db27-168c-4c6a-bd9a-9e2..."}

length: 8

__proto__: Array(0)

__proto__: Object

dispatch after

BooksContainer.jsx:13

Warning

./src/components/Book.jsx

webpackHotDevClient.js:120

Line 41: Using target="_blank" without rel="noopener norereferrer" is a security risk: see https://mathiasbynens.github.io/rel-noopener react/jsx-no-target-blank

redux-devtools

<https://github.com/zalmoxisus/redux-devtools-extension>

```
npm install -D redux-devtools-extension
```

composeWithDevTools

```
import { createStore, applyMiddleware } from "redux";
import reducers from "./reducers";
import { composeWithDevTools } from "redux-devtools-extension";

const store = createStore(reducers, composeWithDevTools(applyMiddleware()));

export default store;
```

React App

zalmoxisus/redux-devtools-ext

localhost:3000

앱

etc

aws

protopie

Facebook

GitHub

강의

deno

책 추가하기

로그아웃

Home

모던 자바스크립트 입문

모던하균요

서재원

<http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791160504439&orderClick=LET&Kc=>

delete

타입스크립트 퀵 스타트

타입스크립트 짱짱짱

정진욱

<http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791186710302&orderClick=LEB&Kc=>

delete

책 제목

감상평

지은이

[구매링크](#)

delete

타입스크립트 퀵 스타트

타입스크립트 짱짱짱

정진욱

<http://www.kyobobook.co.kr/product/detailViewKor.laf?ejkGb=KOR&mallGb=KOR&barcode=9791186710302&orderClick=LEB&Kc=>

delete

Inspector

filter... Commit

@@INIT 4:58:51.03

@@router/LOCATION_CH... +00:00.01

BOOKS_PENDING +00:00.01

BOOKS_FULFILLED +00:00.08

React App

Diff Action State Diff Trace Test

Tree Raw

books (pin)

0 (pin): {bookId:1,ownerId:'7d26db27-16...edAt:nu
11}

1 (pin): {bookId:2,ownerId:'7d26db27-16...edAt:nu
11}

2 (pin): {bookId:81,ownerId:'7d26db27-1...edAt:nu
11}

3 (pin): {bookId:101,ownerId:'7d26db27-...edAt:nu
11}

4 (pin): {bookId:111,ownerId:'7d26db27-...edAt:nu
11}

5 (pin): {bookId:112,ownerId:'7d26db27-...edAt:nu
11}

6 (pin): {bookId:113,ownerId:'7d26db27-...edAt:nu
11}

7 (pin): {bookId:144,ownerId:'7d26db27-...edAt:nu
11}

8 (pin): {bookId:450,ownerId:'7d26db27-...edAt:nu
11}

1x

Pause

Lock

redux-thunk

<https://github.com/reduxjs/redux-thunk>

redux-thunk

- 리덕스 미들웨어
- 리덕스를 만든 사람이 만들었음. (*Dan*)
- 리덕스에서 비동기 처리를 위한 라이브러리
- 액션 생성자를 활용하여 비동기 처리
- 액션 생성자가 액션을 리턴하지 않고, 함수를 리턴함.

```
npm i redux-thunk
```

import thunk from 'redux-thunk';

```
import { createStore, applyMiddleware } from "redux";
import reducers from "./reducers";
import { composeWithDevTools } from "redux-devtools-extension";
import thunk from "redux-thunk"; // import

const store = createStore(
  reducers,
  composeWithDevTools(applyMiddleware(thunk)) // 미들웨어 설정
);

export default store;
```


Before Using thunk

```
const mapDispatchToProps = dispatch => ({
  requestBooks: async token => {
    dispatch(startLoading());
    dispatch(clearError());
    try {
      const res = await axios.get("https://api.marktube.tv/v1/book", {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      dispatch(setBooks(res.data));
      dispatch(endLoading());
    } catch (error) {
      console.log(error);
      dispatch(setError(error));
      dispatch(endLoading());
    }
  }
});
```

Use thunk

```
// BooksContainer.jsx
const mapDispatchToProps = dispatch => ({
  requestBooks: async token => {...},
  requestBooksThunk: token => {
    dispatch(setBooksThunk(token));
  }
});

// actions/index.js
export const setBooksThunk = token => async dispatch => {
  dispatch(startLoading());
  dispatch(clearError());
  try {
    const res = await axios.get("https://api.marktube.tv/v1/book", {
      headers: {
        Authorization: `Bearer ${token}`
      }
    });
    dispatch(setBooks(res.data));
    dispatch(endLoading());
  } catch (error) {
    console.log(error);
    dispatch(setError(error));
    dispatch(endLoading());
  }
};
```

redux-promise-middleware

<https://pburtchaell.gitbook.io/redux-promise-middleware>

```
npm i redux-promise-middleware
```

import promise from 'redux-promise-middleware';

```
import { createStore, applyMiddleware } from "redux";
import reducers from "./reducers";
import { composeWithDevTools } from "redux-devtools-extension";
import thunk from "redux-thunk";
import promise from "redux-promise-middleware"; // import

const store = createStore(
  reducers,
  composeWithDevTools(applyMiddleware(thunk, promise)) // 미들웨어 설정
);

export default store;
```

payload 7 Promise

```
// actions/index.js
export const setBooksPromise = token => ({
  type: BOOKS,
  payload: axios.get("https://api.marktube.tv/v1/book", {
    headers: {
      Authorization: `Bearer ${token}`
    }
  })
});
```

액션의 *type* 에 접미사를 붙인 액션을 자동 생성하고 자동으로 *dispatch* 시킴

```
// actions/index.js
export const BOOKS = 'BOOKS';
export const BOOKS_PENDING = 'BOOKS_PENDING';
export const BOOKS_FULFILLED = 'BOOKS_FULFILLED';
export const BOOKS_REJECTED = 'BOOKS_REJECTED';

// reducers/loading.js
export default function loading(state = initialState, action) {
  switch (action.type) {
    case BOOKS_PENDING:
      return true;

    case BOOKS_FULFILLED:
      return false;

    case BOOKS_REJECTED:
      return false;

    default:
      return state;
  }
}
```

payload 로 들어오는 데이터를 활용하여 표현

```
{                                     // reducers/books.js
  type: 'BOOKS_PENDING'
}

const books = (state = initialState, action) => {
  switch (action.type) {
    case BOOKS_FULFILLED: {
      return [...action.payload.data]
    }
    ...
  }
}

{
  type: 'BOOKS_FULFILLED'
  payload: {
    ...
  }
}

{
  type: 'BOOKS_REJECTED'
  error: true,
  payload: {
    ...
  }
}
```


3. Redux Advanced (2)

3-1) Ducks Pattern

3-2) react-router-dom 과 redux 함께 쓰기

3-3) redux-saga







3-4) redux-actions

Ducks Pattern

<https://github.com/erikras/ducks-modular-redux>

🔒 40 commits 🌿 2 branches 📦 0 packages 🏷️ 0 releases 👤 17 contributors

Clone or download ▾

 battlecry/generators/duck	Add duck battlecry generator (#74)	2 years ago
 CommonJs.md	fixed comments around commonjs example	5 years ago
 README.md	Add CodeFund sponsorship message to README  (#79)	3 months ago
 duck.jpg	first commit	5 years ago
 migrate.jpg	added migrate meme	5 years ago

 README.md

A close-up photograph of a classic yellow rubber duck. The duck is facing left, with its head slightly tilted. It has a large, bright orange beak and two small, black circular eyes. The duck's body is a solid, vibrant yellow. The background is a plain, light-colored surface.

src/redux

- *create.js*

src/redux/modules

- *module1.js*

- *module2.js*

...

- *reducer.js (or index.js)*

```
// src/redux/modules/books.js
```

```
import BookService from '../../services/BookService';
```

```
// 액션 타입 정의 ("app 이름"/"reducer 이름"/"로컬 ACTION_TYPE") => 겹치지 않게 하기 위함
```

```
const PENDING = 'reactjs-books-review/books/PENDING';
```

```
const SUCCESS = 'reactjs-books-review/books/SUCCESS';
```

```
const FAIL = 'reactjs-books-review/books/FAIL';
```

```
// 리듀서 초기값
```

```
const initialState = {
```

```
  books: [],
```

```
  loading: false,
```

```
  error: null,
```

```
};
```

```
// 액션 생성자 함수
```

```
const start = () => ({ type: PENDING });
```

```
const success = books => ({ type: SUCCESS, books });
```

```
const fail = error => ({ type: FAIL, error });
```

```
// thunk 함수
```

```
export const getBooks = token => async dispatch => {
```

```
  dispatch(start());
```

```
  try {
```

```
    await sleep(2000);
```

```
    const res = await BookService.getBooks(token);
```

```
    dispatch(success(res.data));
```

```
  } catch (error) {
```

```
    dispatch(fail(error));
```

```
  }
```

```
};
```

```
// 리듀서
```

```
const books = (state = initialState, action) => {
```

```
  switch (action.type) {
```

```
    case PENDING:
```

```
// src/redux/modules/auth.js
```

```
import UserService from '../../services/UserService';
```

```
const PENDING = 'reactjs-books-review/auth/PENDING';
```

```
const SUCCESS = 'reactjs-books-review/auth/SUCCESS';
```

```
const FAIL = 'reactjs-books-review/auth/FAIL';
```

```
const initialState = {
```

```
  token: null,
```

```
  loading: false,
```

```
  error: null,
```

```
};
```

```
// 액션 생성자 함수
```

```
const start = () => ({ type: PENDING });
```

```
const success = token => ({ type: SUCCESS, token });
```

```
const fail = error => ({ type: FAIL, error });
```

```
// thunk 함수
```

```
export const login = (email, password) => async dispatch => {
```

```
  try {
```

```
    dispatch(start());
```

```
    const res = await UserService.login(email, password);
```

```
    const { token } = res.data;
```

```
    localStorage.setItem('token', token);
```

```
    dispatch(success(token));
```

```
  } catch (error) {
```

```
    dispatch(fail(error));
```

```
  }
```

```
};
```

```
export const logout = token => async dispatch => {
```

```
  // 서버에 알리기
```

```
  try {
```

```
    await UserService.logout(token);
```

```
  } catch (error) {
```

```
// src/redux/modules/reducer.js
```

```
import { combineReducers } from 'redux';  
import auth from './auth';  
import books from './books';
```

```
const reducer = combineReducers({  
  auth,  
  books,  
});
```

```
export default reducer;
```

```
// src/redux/create.js
```

```
import { createStore, applyMiddleware } from 'redux';  
import reducer from './modules/reducer';  
import { composeWithDevTools } from 'redux-devtools-extension';  
import thunk from 'redux-thunk';
```

```
export default function create(token) {  
  const initialState = {  
    books: undefined,  
    auth: {  
      token,  
      loading: false,  
      error: null,  
    },  
  };  
  
  const store = createStore(  
    reducer,  
    initialState,  
    composeWithDevTools(applyMiddleware(thunk)),  
  );  
  
  return store;  
}
```



```
// src/index.js
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'antd/dist/antd.css';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import create from './redux/create';
import { Provider } from 'react-redux';
```

```
const token = localStorage.getItem('token');
const store = create(token);
```

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root'),
);
```

```
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

```
// src/containers/BooksContainer.jsx
```

```
import { connect } from 'react-redux';  
import Books from '../components/Books';  
import { getBooks } from '../redux/modules/books';
```

```
const mapStateToProps = state => ({  
  token: state.auth.token,  
  books: state.books.books,  
  loading: state.books.loading,  
  error: state.books.error,  
});
```

```
const mapDispatchToProps = dispatch => ({  
  getBooks: token => {  
    dispatch(getBooks(token));  
  },  
});
```

```
export default connect(mapStateToProps, mapDispatchToProps)(Books);
```

```
// src/components/Books.jsx
```

```
import React from 'react';
```

```
import { useEffect } from 'react';
```

```
const Books = ({ token, books, loading, error, getBooks }) => {
```

```
  useEffect(() => {
```

```
    getBooks(token);
```

```
  }, [token, getBooks]);
```

```
  if (error !== null) {
```

```
    return <div>에러다</div>;
```

```
  }
```

```
  return (
```

```
    <>
```

```
      {loading && <p>로딩 중...</p>}
```

```
      <ul>
```

```
        {books.map(book => (
```

```
          <li key={book.bookId}>{book.title}</li>
```

```
        )))
```

```
      </ul>
```

```
    </>
```

```
  );
```

```
};
```

```
export default Books;
```

react-router 와 *redux* 함께 쓰기

<https://github.com/supasate/connected-react-router>

React Router: Declarative Routing

+

reacttraining.com/react-router/web/guides/redux-integration/deep-integration

REACT TRAINING / REACT ROUTER

CORE

WEB

NATIVE

ANNOUNCEMENTS

The Future of React Router

EXAMPLES

Basic

URL Parameters

Nesting

Redirects (Auth)

Custom Link

Preventing Transitions

No Match (404)

Recursive Paths

Sidebar

Animated Transitions

Route Config

Modal Gallery

StaticRouter Context

Query Parameters

GUIDES

Quick Start

Primary Components

Server Rendering

Code Splitting

Scroll Restoration

Philosophy

Testing

Redux Integration

Static Routes

API

Hooks

useHistory

useLocation

useParams

useRouteMatch

from the router. This is straightforward to fix. Find where you connect your component and wrap it in withRouter.

Deep integration

Some folks want to:

1. Synchronize the routing data with, and accessed from, the store.

2. Be able to navigate by dispatching actions.

3. Have support for time travel debugging for route changes in the Redux devtools.

All of this requires deeper integration.

Our recommendation is **not to keep your routes in your Redux store at all**. Reasoning:

1. Routing data is already a prop of most of your components that care about it. Whether it comes from the store or the router, your component's code is largely the same.

2. In most cases, you can use Link, NavLink and Redirect to perform navigation actions. Sometimes you might also need to navigate programmatically, after some asynchronous task that was originally initiated by an action. For example, you might dispatch an action when the user submits a login form. Your **thunk**, **saga** or other async handler then authenticates the credentials, *then* it needs to somehow navigate to a new page if successful. The solution here is simply to include the **history** object (provided to all route components) in the payload of the action, and your async handler can use this to navigate when appropriate.

3. Route changes are unlikely to matter for time travel debugging. The only obvious case is to debug issues with your router/store synchronization, and this problem goes away if you don't synchronize them at all.

But if you feel strongly about synchronizing your routes with your store, you may want to try **Connected React Router**, a third party binding for React Router v4 and Redux.

// after

import { withRouter } from 'react-router-dom'

export default withRouter(connect(mapStateToProps)(Something))

```
npm install connected-react-router
```

reducer 에 router 라는 state 를 combine

```
// src/redux/modules/reducer.js
```

```
import { combineReducers } from 'redux';  
import auth from './auth';  
import books from './books';  
import { connectRouter } from 'connected-react-router';
```

```
const reducer = history =>  
  combineReducers({  
    auth,  
    books,  
    router: connectRouter(history),  
  });
```

```
export default reducer;
```

store 에/ routerMiddleware 를 추가

```
// src/redux/create.js

import { createStore, applyMiddleware } from 'redux';
import reducer from './modules/reducer';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import { createBrowserHistory } from 'history';
import { routerMiddleware } from 'connected-react-router';

export const history = createBrowserHistory();

export default function create(token) {
  const initialState = {
    books: undefined,
    auth: {
      token,
      loading: false,
      error: null,
    },
  },
  };

  const store = createStore(
    reducer(history),
    initialState,
    composeWithDevTools(applyMiddleware(thunk, routerMiddleware(history))),
  );

  return store;
}
```


Router => ConnectedRouter

```
import React from 'react';
import { Switch, Route } from 'react-router-dom';
import Home from './pages/Home';
import Signin from './pages/Signin';
import NotFound from './pages/NotFound';
import ErrorBoundary from 'react-error-boundary';
import { ConnectedRouter } from 'connected-react-router';
import { history } from './redux/create';

const ErrorFallbackComponent = ({ error }) => <div>{error.message}</div>;

const App = () => (
  <ErrorBoundary FallbackComponent={ErrorFallbackComponent}>
    <ConnectedRouter history={history}>
      <Switch>
        <Route exact path="/signin" component={Signin} />
        <Route exact path="/" component={Home} />
        <Route component={NotFound} />
      </Switch>
    </ConnectedRouter>
  </ErrorBoundary>
);

export default App;
```

history.push() 대신 dispatch(push())

```
// src/redux/modules/auth.js
```

```
export const login = (email, password) => async dispatch => {  
  try {  
    dispatch(start());  
    const res = await UserService.login(email, password);  
    const { token } = res.data;  
    localStorage.setItem('token', token);  
    dispatch(success(token));  
    dispatch(push('/'));  
  } catch (error) {  
    dispatch(fail(error));  
  }  
};
```

redux-saga

<https://redux-saga.js.org>

리덕스 사가

- 미들웨어 입니다.
- 제너레이터 객체를 만들어 내는 제네레이터 생성 함수를 이용합니다.
- 리덕스 사가 미들웨어를 설정하고,
- 내가 만든 사가 함수를 등록한 후
- 사가 미들웨어를 실행합니다.
- 그리고 등록된 사가 함수를 실행할 액션을 디스패치하면 됩니다.

```
npm i redux-saga
```

사가 미들웨어를 리덕스 미들웨어로 설정

```
// src/redux/create.js
import { createStore, applyMiddleware } from 'redux';
import reducer from './modules/reducer';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import { createBrowserHistory } from 'history';
import { routerMiddleware } from 'connected-react-router';
import createSagaMiddleware from 'redux-saga'; // 1. import

export const history = createBrowserHistory();
const sagaMiddleware = createSagaMiddleware(); // 2. saga 미들웨어 생성

export default function create(token) {
  const initialState = {
    books: undefined,
    auth: {
      token,
      loading: false,
      error: null,
    },
  },
};

  const store = createStore(
    reducer(history),
    initialState,
    composeWithDevTools(
      applyMiddleware(thunk, routerMiddleware(history), sagaMiddleware), // 3. 리덕스 미들웨어에 saga 미들웨어 추가
    ),
  );

  return store;
}
```

나의 사가 함수 만들기

```
// src/redux/modules/books.js
```

```
import { delay, put, call } from 'redux-saga'; // 사가 이펙트 추가
```

```
// saga 함수
```

```
function* getBooksSaga(action) {  
  const token = action.payload.token;  
  yield put(start());  
  try {  
    yield delay(2000);  
    const res = yield call(BookService.getBooks, token);  
    yield put(success(res.data));  
  } catch (error) {  
    yield put(fail(error));  
  }  
}
```

나의 사가 함수를 실행하는 사가 만들기

```
// src/redux/modules/books.js
import { delay, put, call, takeEvery } from 'redux-saga/effects'; // 사가 이펙트 추가

// saga 함수
function* getBooksSaga(action) {
  const token = action.payload.token;
  yield put(start());
  try {
    yield delay(2000);
    const res = yield call(BookService.getBooks, token);
    yield put(success(res.data));
  } catch (error) {
    yield put(fail(error));
  }
}

// getBooksSaga 를 시작하는 액션 타입 정의
const START_SAGA = 'START_SAGA';

// getBooksSaga 를 시작하는 액션 생성 함수
export const startSaga = token => ({ type: START_SAGA, payload: { token } });

// saga 함수를 등록하는 saga
export function* booksSaga() {
  yield takeEvery(START_SAGA, getBooksSaga);
}
```


나의 여러 사가 모듈을 합친 *rootSaga* 만들기

```
// src/redux/modules/saga.js
```

```
import { all } from 'redux-saga/effects';  
import { booksSaga } from './books';
```

```
export default function* rootSaga() {  
  yield all([booksSaga()]);  
}
```

rootSaga 를 사가 미들웨어로 실행

```
// src/redux/create.js
import { createStore, applyMiddleware } from 'redux';
import reducer from './modules/reducer';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import { createBrowserHistory } from 'history';
import { routerMiddleware } from 'connected-react-router';
import createSagaMiddleware from 'redux-saga';
import rootSaga from './modules/saga'; // 나의 사가 가져오기

export const history = createBrowserHistory();
const sagaMiddleware = createSagaMiddleware();

export default function create(token) {
  const initialState = {
    books: undefined,
    auth: {
      token,
      loading: false,
      error: null,
    },
  };

  const store = createStore(
    reducer(history),
    initialState,
    composeWithDevTools(
      applyMiddleware(thunk, routerMiddleware(history), sagaMiddleware),
    ),
  );

  sagaMiddleware.run(rootSaga); // 나의 사가들을 실행

  return store;
}
```

나의 사가 함수를 시작하게 할 액션을 디스패치

```
// src/containers/BooksContainer.jsx

import React, { useCallback } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import Books from '../components/Books';
import { startSaga } from '../redux/modules/books';

const BooksContainer = props => {
  const token = useSelector(state => state.auth.token);
  const { books, loading, error } = useSelector(state => state.books);

  const dispatch = useDispatch();

  const getBooks = useCallback(() => {
    dispatch(startSaga(token));
  }, [token, dispatch]);

  return (
    <Books
      {...props}
      books={books}
      loading={loading}
      error={error}
      getBooks={getBooks}
    />
  );
};

export default BooksContainer;
```

select 이펙트 활용하기 (1)

```
// src/containers/BooksContainer.jsx

import React, { useCallback } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import Books from '../components/Books';
import { startSaga } from '../redux/modules/books';

const BooksContainer = props => {
  const { books, loading, error } = useSelector(state => state.books);

  const dispatch = useDispatch();

  const getBooks = useCallback(() => {
    dispatch(startSaga());
  }, [dispatch]);

  return (
    <Books
      {...props}
      books={books}
      loading={loading}
      error={error}
      getBooks={getBooks}
    />
  );
};

export default BooksContainer;
```

select 이펙트 활용하기 (2)

```
// src/redux/modules/books.js
```

```
import { delay, put, call, takeEvery, select } from 'redux-saga/effects'; // select 추가
```

```
// saga 함수
```

```
function* getBooksSaga() {  
  const token = yield select(state => state.auth.token); // 여기서 가져오기  
  yield put(start());  
  try {  
    yield delay(2000);  
    const res = yield call(BookService.getBooks, token);  
    yield put(success(res.data));  
  } catch (error) {  
    yield put(fail(error));  
  }  
}
```

takeEvery, takeLatest, takeLeading

```
// src/redux/modules/books.js
```

```
import { delay, put, call, takeEvery, takeLatest, takeLeading, select } from 'redux-saga/effects';
```

```
// saga 함수를 등록하는 saga
```

```
export function* booksSaga() {  
  yield takeEvery(START_SAGA, getBooksSaga);  
  // yield takeLatest(START_SAGA, getBooksSaga);  
  // yield takeLeading(START_SAGA, getBooksSaga);  
}
```

redux-actions

<https://github.com/redux-utilities/redux-actions>

Read Me

Introduction

API Reference

createAction(s)

handleAction(s)

combineActions

External Resources

Changelog

Contributors

Read Me

redux-actions

build error | codecov 96% | npm v2.6.5 | downloads 1.2M/month

Flux Standard Action utilities for Redux

Table of Contents

- Getting Started
 - Installation
 - Usage
- Documentation

Edit on GitHub

CONTENTS

redux-actions

Getting Started

Installation

Usage

Documentation

Getting Started

Installation

```
$ npm install --save redux-actions
```

or

```
$ yarn add redux-actions
```



Powered by GitBook


```
npm i redux-actions
```

- *createAction, createActions*
- *handleAction, handleActions*
- *combineActions*

createAction

```
// src/redux/modules/books.js
```

```
import { createAction } from 'redux-actions';
```

```
const start = createAction('START');
```

```
const success = createAction('SUCCESS', books => ({ books }));
```

```
const fail = createAction('FAIL');
```

```
console.log(start());
```

```
console.log(success(['book']));
```

```
console.log(fail(new Error()));
```

createActions

```
// src/redux/modules/books.js

import { createActions } from 'redux-actions';

const { start, success, fail } = createActions(
  {
    SUCCESS: books => ({ books }),
  },
  'START',
  'FAIL',
  {
    prefix: 'reactjs-books-review/books',
  },
);

console.log(start());
console.log(success(['book']));
console.log(fail(new Error()));
```

handleActions

```
// src/redux/modules/books.js

import { handleActions } from 'redux-actions';

const books = handleActions(
  {
    START: () => ({
      books: [],
      loading: true,
      error: null,
    }),
    SUCCESS: (state, action) => ({
      books: action.payload.books,
      loading: false,
      error: null,
    }),
    FAIL: (state, action) => ({
      books: [],
      loading: false,
      error: action.payload,
    }),
  },
  initialState,
  {
    prefix: 'reactjs-books-review/books',
  },
);
```