Hello,

KDT 웹 개발자 양성 프로젝트

1기! 41th

with

JSX 와

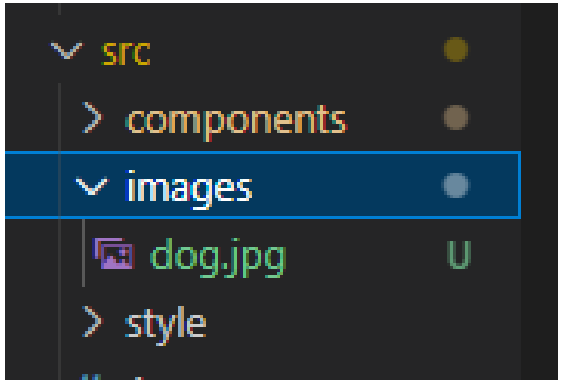
IF문!

```jsx
export default function BoardDetail() {
  const { boardID } = useParams();
  const location = useLocation();
  const articleList = [1, 2];

  if (articleList.find((el) => el === parseInt(boardID))) {
    return (
      <>
        <Header />
        <h2>{boardID} 번 게시글 내용입니다!</h2>
        <p>쿼리: {location.search}</p>
        <p>주소: {location.pathname}</p>
        <p>해쉬: {location.hash}</p>
      </>
    );
  } else {
    return (
      <>
        <NotFound />
      </>
    )
  }
}
```

Public

폴더 사용

src 폴더 내부 참조는 문제 없습니다!




```jsx
import dogImg from "../images/dog.jpg"

export default function Image() {
    return (
        <>
            <img src={dogImg} alt="강아지" />
        </>
    )
}
```

src/components/Image.js

Component Composition

Specialization

```jsx
export default function Dialog(props) {
  return (
    <div style={{ backgroundColor: props.color }}>
      <h1>{props.title}</h1>
      <p>{props.message}</p>
    </div>
  );
}
```

src/components/Dialog.js

```jsx
function App() {
  return (
    <div className="App">
      <Dialog
        color="orange"
        title="Welcome to summoner's lift"
        message="소환사의 협곡에 오신 걸 환영합니다."
      />
    </div>
  );
}
```

src/App.js

```jsx
import Dialog from "./Dialog";

export default function WelcomeDialog() {
  return (
    <Dialog
      color="orange"
      title="Welcome to sommoner's lift"
      message="소환사의 협곡에 오신 걸 환영합니다."
    />
  );
}
```

src/components/WelcomeDialog.js

```jsx
import WelcomeDialog from "./components/WelcomeDialog";

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

src/App.js

Welcome to summoner's lift

소환사의 협곡에 오신 걸 환영합니다.

Props.children

Props.children

- Props 로 값을 보내는 것은 이미 배우셨습니다!
- 다만, 이미 정해진 props에 전달하는 값이 아닌 상황에 따라 html 요소 또는 컴포넌트 자체를 보내고 싶을 땐 어떻게 하면 될까요?
- 이럴 때에는 컴포넌트의 자식 요소를 한꺼번에 전달해 주는 props.children 을 사용할 수 있습니다.

```jsx
function WelcomeDialog(props) {
    return (
        <FancyBorder color="blue">
            <h1 className="Dialog-title">
                어서오세요
            </h1>
            <p className="Dialog-message">
                우리 사이트에 방문하신 것을 환영합니다!
            </p>
        </FancyBorder>
    );
}
```

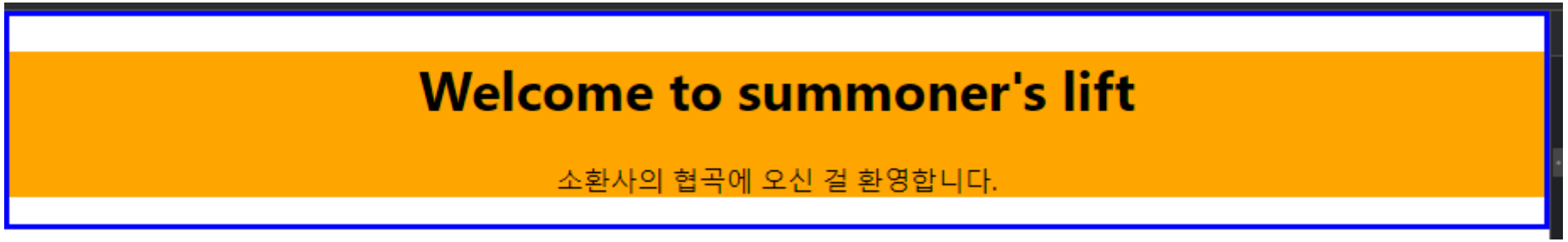
Props

Props.children

Containment

```jsx
import WelcomeDialog from "./components/WelcomeDialog";
import FancyBorder from "./components/FancyBorder";

function App() {
  return (
    <div className="App">
      <FancyBorder color="blue">
        <WelcomeDialog />
      </FancyBorder>
    </div>
  );
}
```

src/App.js



Specialization + Containment

```jsx
export default function Dialog(props) {
  return (
    <div style={{ backgroundColor: props.color }}>
      <h1>{props.title}</h1>
      <p>{props.message}</p>
      {props.children}
    </div>
  );
}
```

src/components/Dialog.js

```jsx
import Dialog from "./Dialog";

export default function SignUpDialog() {
  return (
    <Dialog
      color="skyblue"
      title="안내"
      message="회원 가입이 필요한 서비스 입니다"
    >
      <a href="#">회원 가입 페이지로 이동</a>
    </Dialog>
  );
}
```

src/components/SignUpDialog.js

```jsx
import SignUpDialog from "./components/SignUpDialog";

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

src/App.js

안내

회원 가입이 필요한 서비스 입니다

회원 가입 페이지로 이동

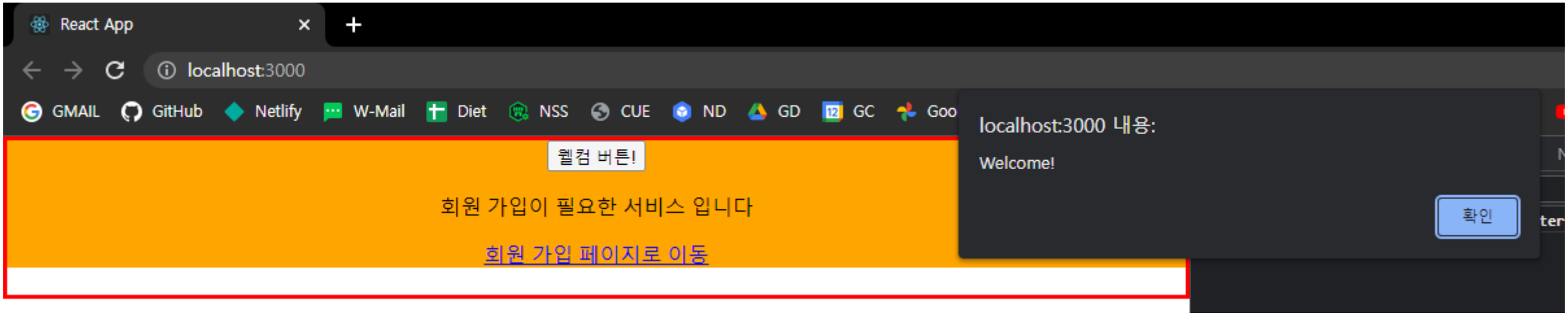
조건에 따른 처리

Props.title 처리

```
export default function Dialog(props) {
  return (
    <div style={{ backgroundColor: props.color }}>
      {typeof props.title === "string" ? <h1>{props.title}</h1> : props.title}
      <p>{props.message}</p>
      {props.children}
    </div>
  );
}
```

src/components/Dialog.js

- 전달 된 props.title 이 문자열이면 <h1> 태그 내부에 넣어서 출력하고 아니면 직접 출력하는 형태로 처리

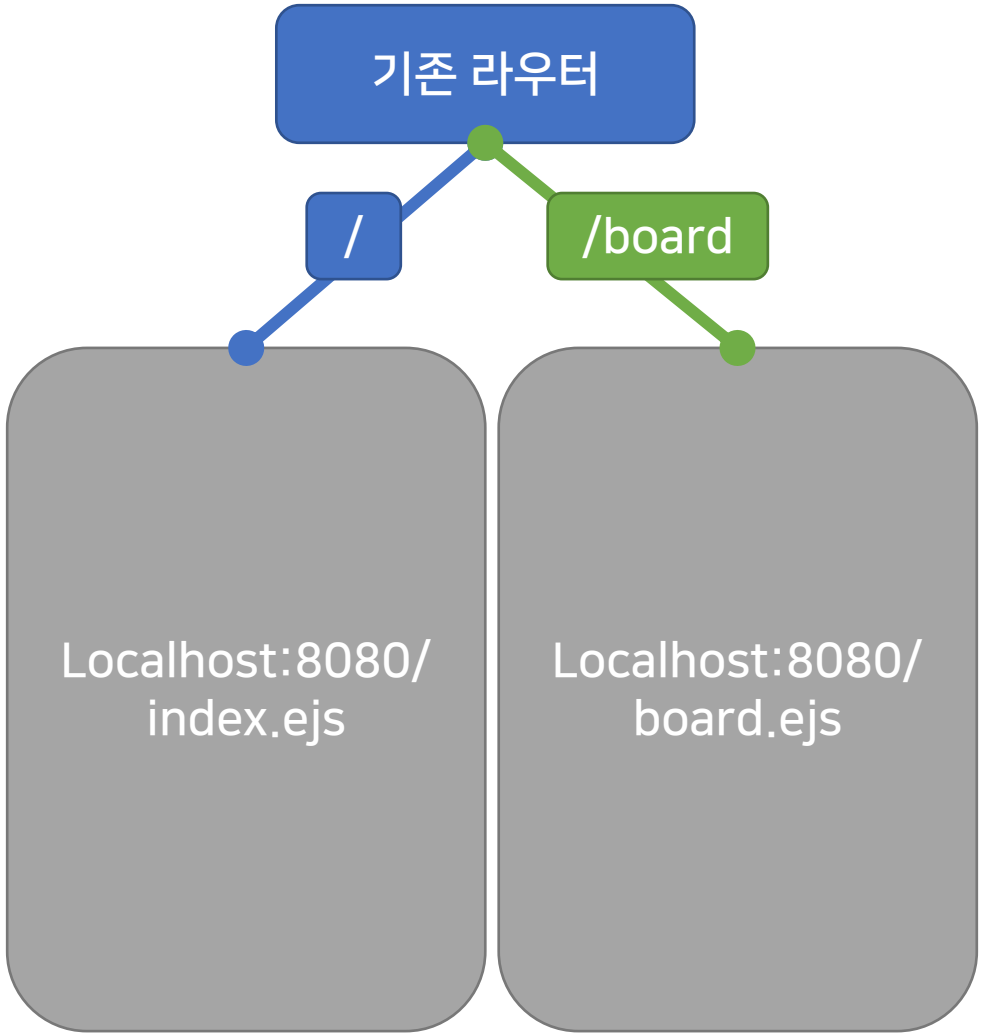

React App
localhost:3000
GMAIL
GitHub
Netlify
W-Mail
Diet
NSS
CUE
ND
GD
GC
웰컴 버튼!
회원 가입이 필요한 서비스 입니다
회원 가입 페이지로 이동
localhost:3000 내용:
Welcome!
확인

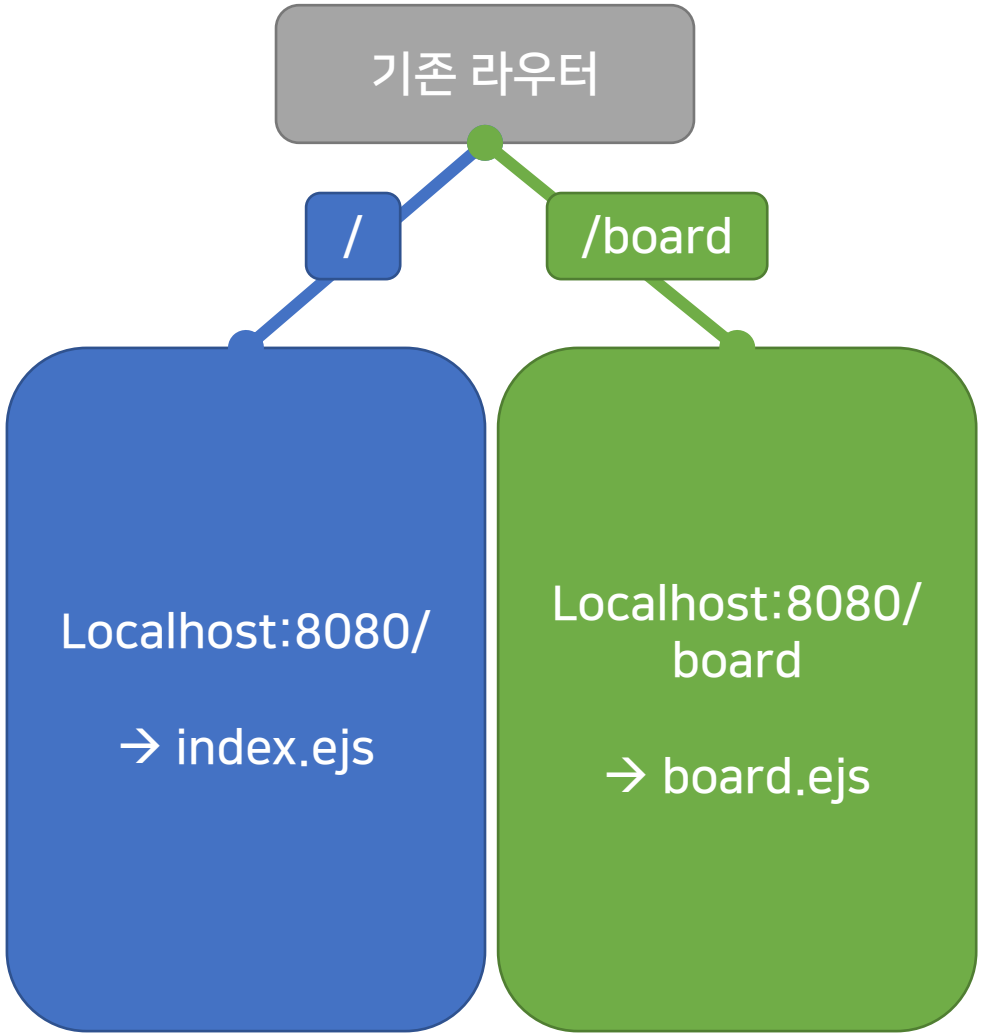
React

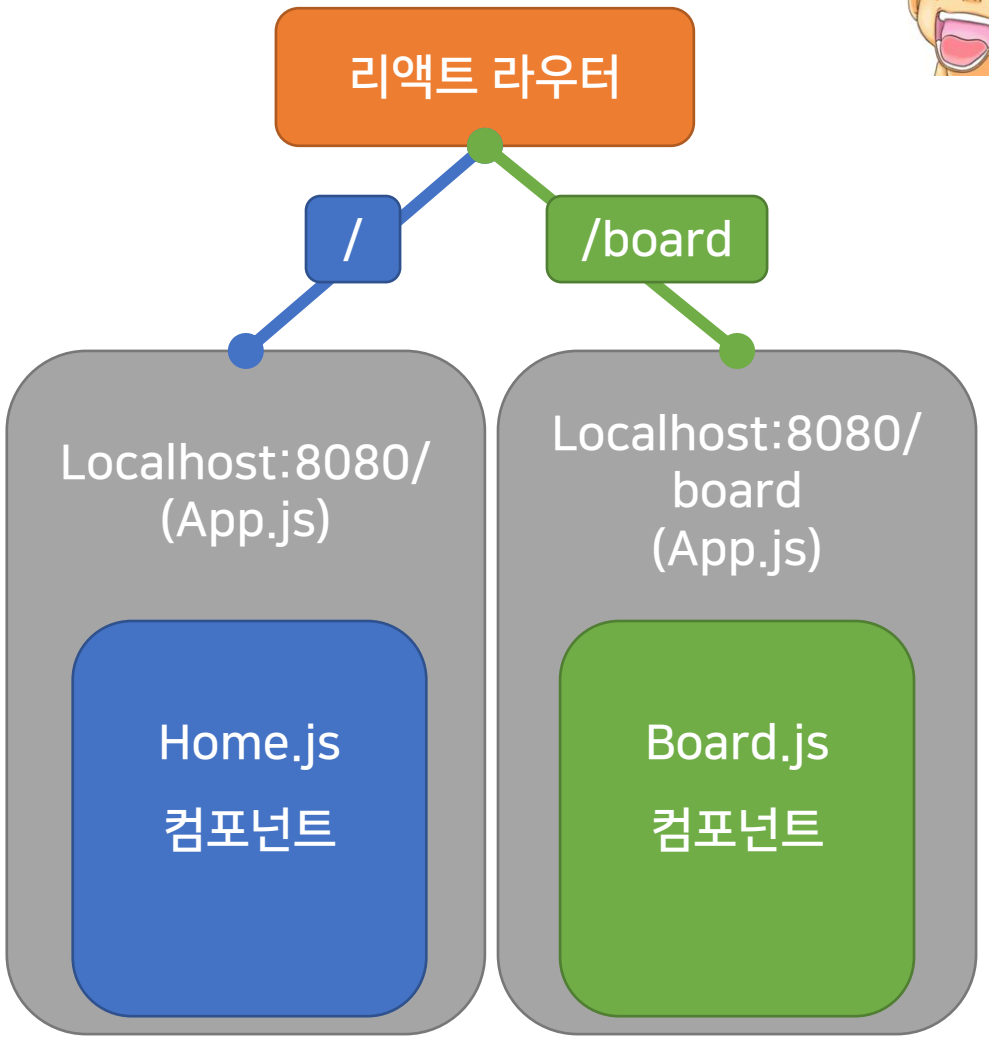
Router

React 에서 Router 활용하기

- 지금까지 Router 는 입력받은 주소에 따라 페이지를 변경해주는 역할로 많이 사용이 되었습니다.
- 하지만, React 에서는 컴포넌트별(DOM) 라우팅이 가능합니다!
- 조건부 렌더링으로 처리가 가능하지만 서비스의 경우 주소에 따른 구분을 해줘야만 서비스별 구분이 가능하므로 라우팅 기능을 활용이 필요 합니다!
- 그리고 해당 모듈을 쓰면 페이지 깜박임 없이 부드러운 브라우징 가능


기존 라우터
/
/board
Localhost:8080/
index.ejs
Localhost:8080/
board.ejs


기존 라우터
/
/board
Localhost:8080/
→ index.ejs
Localhost:8080/
board
→ board.ejs


리액트 라우터
/
/board
Localhost:8080/
(App.js)
Home.js
컴포넌트
Localhost:8080/
board
(App.js)
Board.js
컴포넌트

```
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </>
);

reportWebVitals();
```

Src/index.js

```jsx
<nav>
    <ul>
      <li>
        <Link to="/">홈 페이지 이동</Link>
      </li>
      <li>
        <Link to="/profile">프로필 페이지 이동</Link>
      </li>
      <li>
        <Link to="/board">게시판 페이지 이동</Link>
      </li>
    </ul>
</nav>
```

Src/App.js

```jsx
import { Link, Outlet, Route, Routes } from "react-router-dom";
import Profile from "./components/Profile";
import Board from "./components/Board";

function App() {
  return (
    <div className="App">
      <nav>
        <ul>
          <li>
            <Link to="/">홈 페이지 이동</Link>
          </li>
          <li>
            <Link to="/profile">프로필 페이지 이동</Link>
          </li>
          <li>
            <Link to="/board">게시판 페이지 이동</Link>
          </li>
        </ul>
      </nav>
      <Routes>
        <Route path="/profile" element={<Profile />} />
        <Route path="/board" element={<Board />} />
      </Routes>
    </div>
  );
}

export default App;
```

전체 코드

Src/App.js

React Router

심화 활용

```jsx
import { Route, Routes } from "react-router-dom";
import Profile from "./components/Profile";
import Board from "./components/Board";
import Header from "./components/Header";
import NotFound from "./components/NotFound";

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Header />} />
        <Route path="profile" element={<Profile />} />
        <Route path="board" element={<Board />} />
      </Routes>
    </div>
  );
}

export default App;
```

Src/App.js

주소 예외 처리

```jsx
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Header />} />
        <Route path="profile" element={<Profile />} />
        <Route path="board" element={<Board />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </div>
  );
}

export default App;
```

코드 처리 방향

Src/App.js

주소

Parameter 활용

Route 에 Parameter 선언

- App.js 에 선언 된 라우터 선언부에 Parameter 를 선언해 봅시다!
- 기존과 같은 방법으로 주소/:parameter 로 선언하면 됩니다!

```
<Routes>
        <Route path="/" element={<Header />} />
        <Route path="profile" element={<Profile />} />
        <Route path="board" element={<Board />} />
        <Route path="board/:boardID" element={<BoardDetail />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
```

Src/App.js

```jsx
import { useParams } from "react-router-dom"
import Header from "./Header";

export default function BoardDetail() {
    const params = useParams();
    console.log(params);
    return (
        <>
            <Header />
            <h2>{params.boardID} 번 게시글 내용입니다!</h2>
        </>
    )
}
```

Src/component/BoardDetail.js


Redux

React 입문자들이 알아야할 Redux 쉽게설명 (8분컷)

조회수 5.7만회 • 1년 전

코딩애플

React 하다보면 Redux를 필히 만나게 되는데 한 해 리덕스 포기자가 10만명이나 되기 때문에 준비했습니다 리액트 강의 ...

React 입문자들이 알아야할 Redux 쉽게설명 (8분컷)

조회수 5.7만회 • 1년 전

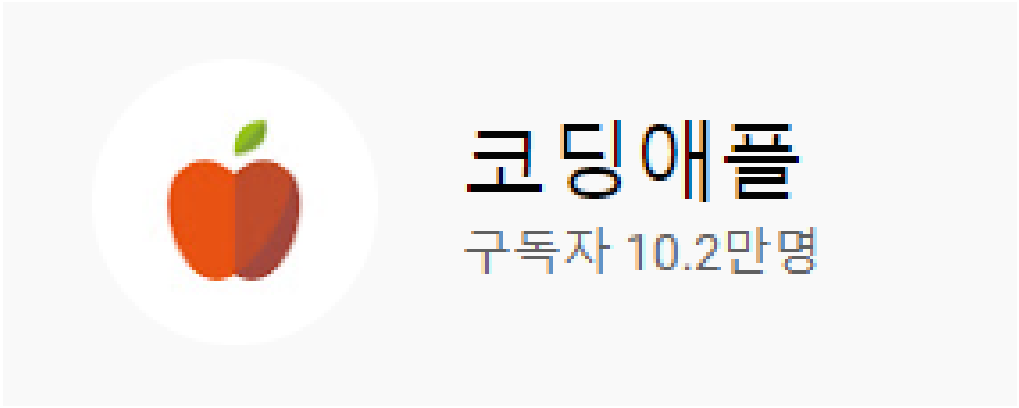
코딩애플

React 하다보면 Redux를 필히 만나게 되는데 한 해 리덕스 포기자가 10만명이나 되기 때문에 준비했습니다 리액트 강의 ...


We Can Do It!
J. Howard Miller
POST FEB. 15 TO FEB. 28
WESTINGHOUSE
WAR PRODUCTION CO-ORDINATING COMMITTEE


Redux
© Dave Stamboulis

Redux



https://www.youtube.com/watch?v=QZcYz2NrDIs&t=212s

그래서 Redux 가 뭐죠?

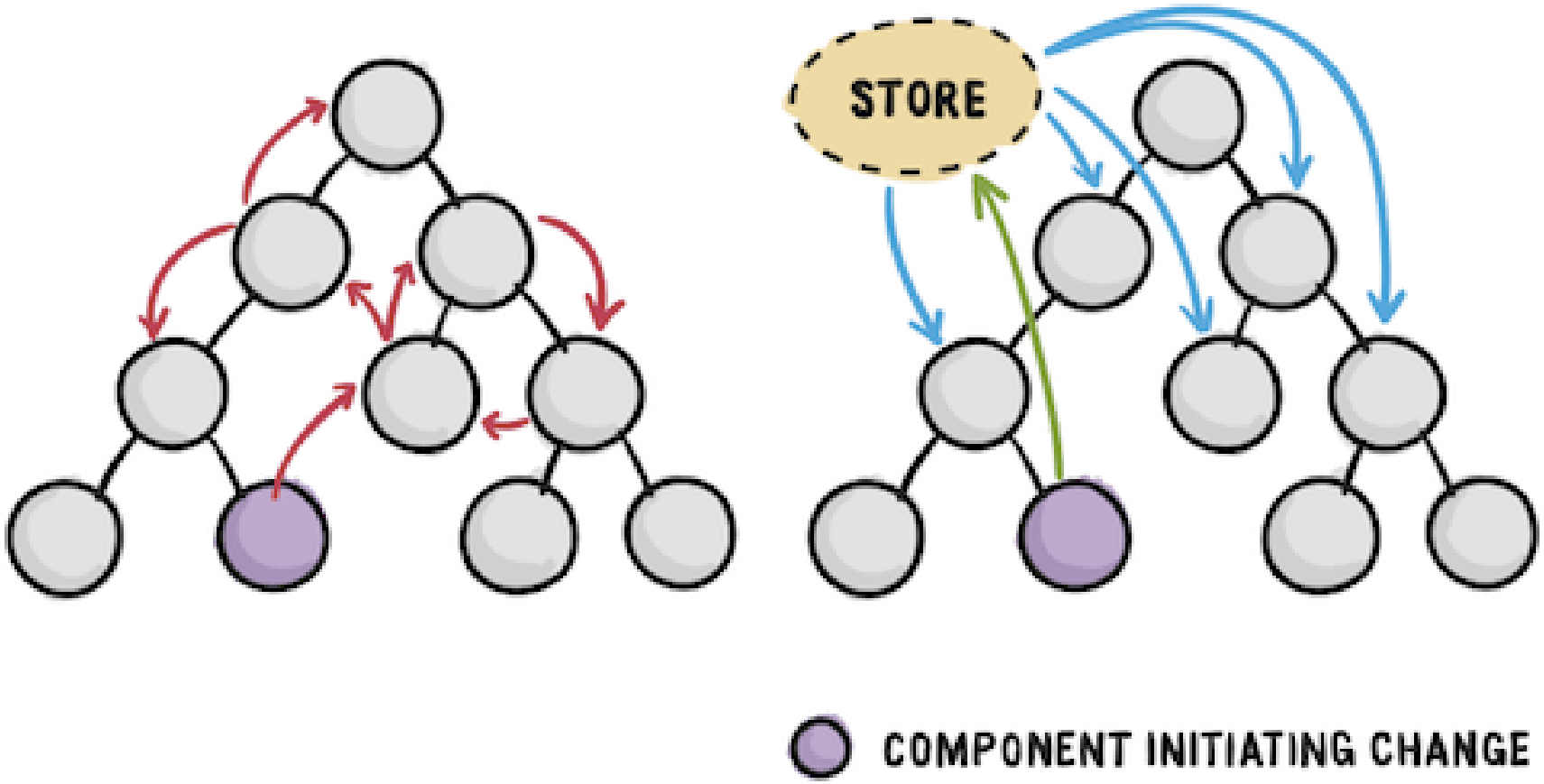
- Redux 는 상태 관리 라이브러리 입니다!
- R로 시작해서 React 랑만 쓰는 것 같지만, 상태 관리가 필요한 다른 프레임 워크(Angular.js / Vue.js / 심지어 Jquery) 에서도 사용이 가능합니다!
- 물론 Redux 하나를 전체가 공유하는 것은 아니고 Redux 의 개념을 각각의 프레임 워크에 맞춘 라이브러리를 사용합니다
- 따라서, 우리가 쓰는 건 React-Redux 입니다!

그래서 Redux 가 뭐죠?

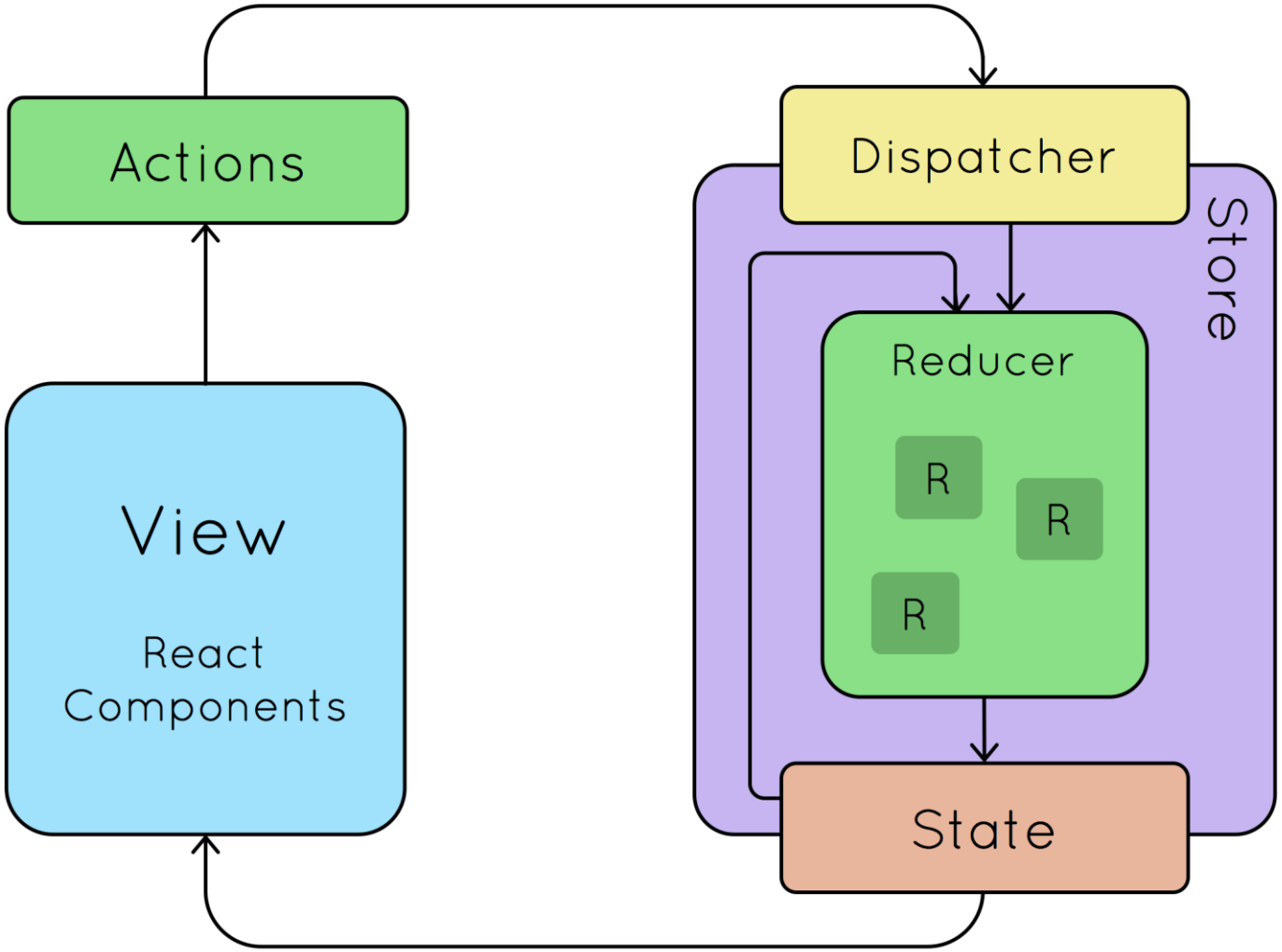
- 컴포넌트의 상태를 하나하나 Props 로 전달하면 너무 힘들기 때문에 이를 해결하고자 나온 라이브러리 입니다
- 컴포넌트의 상태를 각각 컴포넌트 별 State 에 따라 관리하는 것이 아닌 하나의 Store 라는 곳에서 관리 합니다!
- 따라서, 상태 변화 값을 중첩 된 컴포넌트 수 만큼 Props 로 전달하는 방식이 아니라 Store 에서 한번에 꺼내서 사용하는 편리함을 제공합니다!
- 물론 그 편리함을 쓰려면 어려움이 발생하죠 ㅎㅎㅎ ☺

WITHOUT REDUX

WITH REDUX

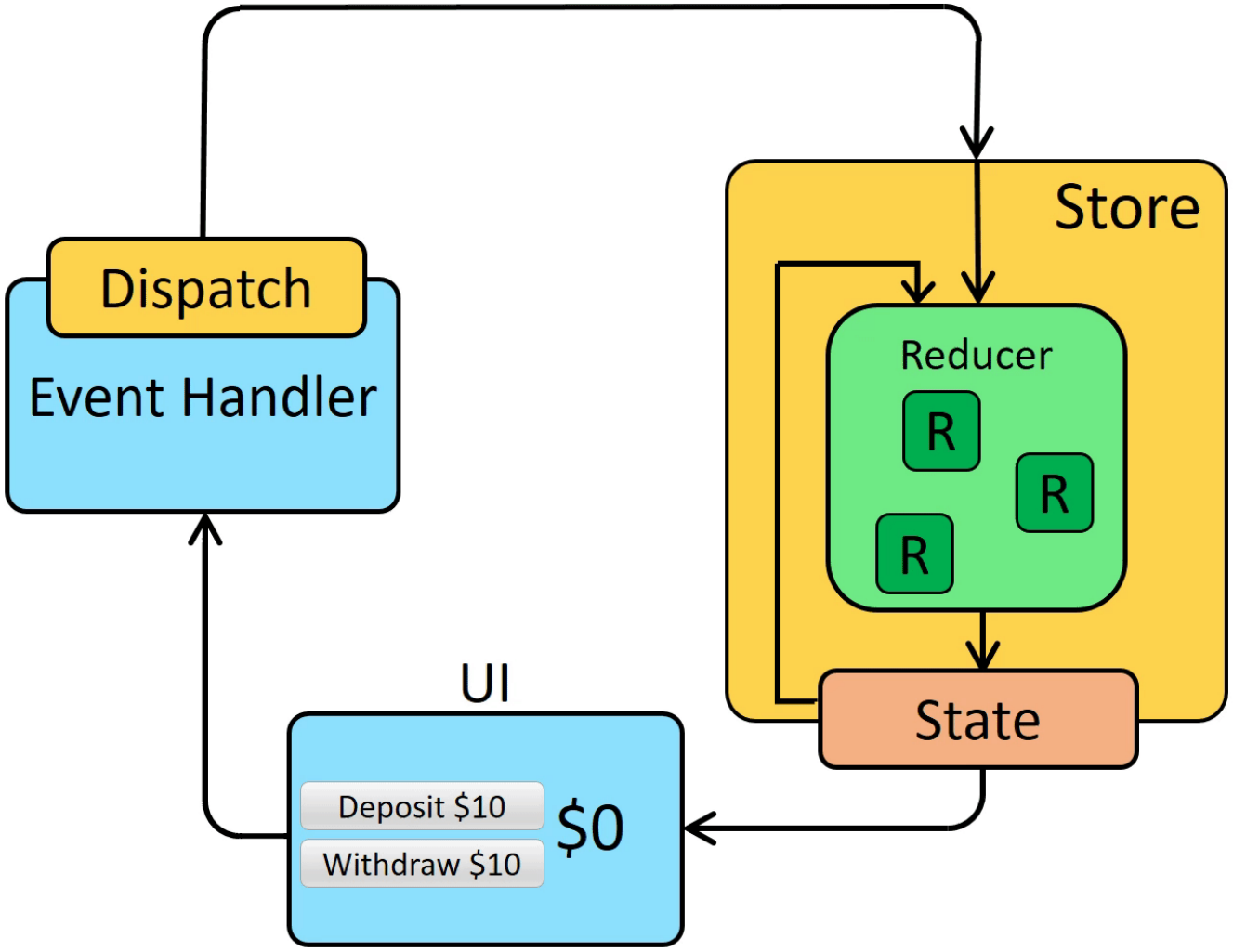


Redux 동작 순서


Actions
Dispatcher
Store
Reducer
R
R
R
View
React
Components
State

Redux 동작 순서

1. 디스패치(Dispatch) 함수를 실행하면
2. 액션(Action)이 발생합니다
3. 이 액션을 리듀서(Reducer)가 받아서
4. 상태(State)를 변경합니다
5. 상태가 변경 되면 컴포넌트를 리렌더링 됩니다!


Dispatch
Event Handler
Store
Reducer
R
R
R
State
UI
Deposit $10
Withdraw $10
$0

코딩 애플 코드로

하나하나 확인하기

Redux 기초 세팅!

- Redux 를 위한 새로운 App 을 만듭시다!
- Npx create-react-app redux-app
- 앱 생성 후, redux 관련 모듈을 설치 합시다!
- Npm install redux
- Npm install react-redux

Redux 기초 세팅!

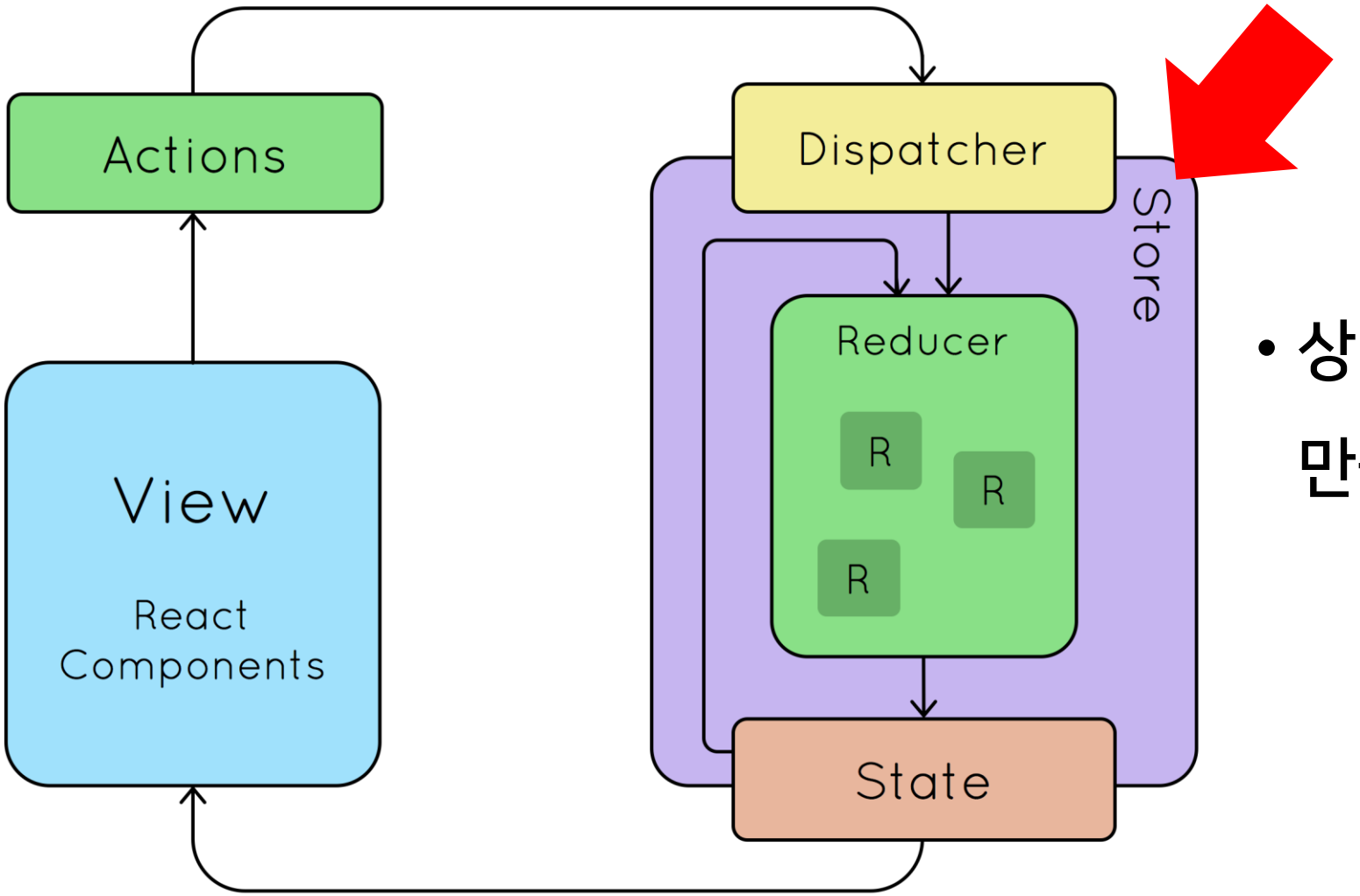
- 라우팅 처리 하던 것처럼! Redux 적용을 위해서는 <Provider> 컴포넌트를 임포트하고 해당 컴포넌트로 <App> 컴포넌트를 감싸줘야만 합니다!
- Index.js 에 가서 코드를 처리!

```
import { Provider } from 'react-redux';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <>
    <Provider>
      <App />
    </Provider>
  </>
);
```

Src/index.js

Store

Store 만들기!



- 상태를 저장할 Store 부터 만들어 봅시다!

Store 만들기!

- Redux 에서 createStore 를 임포트 한 뒤, store 를 만들어 줍니다
- 그리고 <Provider> 컴포넌트에게 상태 관리를 할 store 속성에 만들어진 store 를 부여해 줍니다!

Store

앱에는 단 하나의 스토어가 존재
현재 상태, 리듀서가 포함

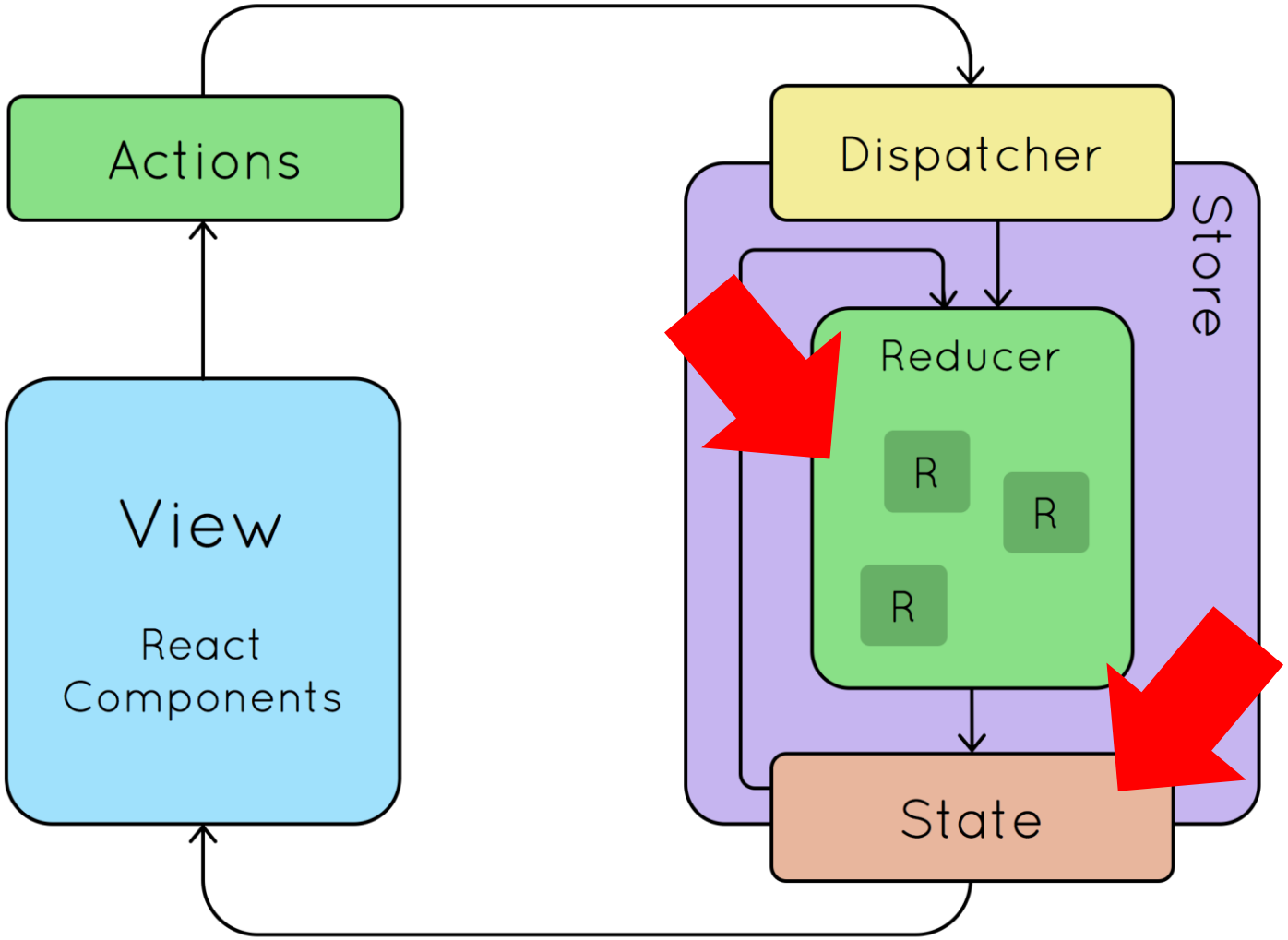
```
import { createStore } from 'redux';

let store = createStore(reducer);

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

Src/index.js

State &

Reducer

State 값 설정 및 Reducer 만들기!



- 상태 역할을 해줄 State 값 설정을 하고
- State 값을 변경해 줄 Reducer 를 만들어 봅시다!

State 설정

- State 는 하나의 변수 또는 객체를 사용하면 됩니다!
- 저희는 간단한 형태로 알아만 볼 것이기 때문에 변수 하나를 선언해서 사용해 봅시다!

```
const weight = 100;
```

Reducer 만들기!

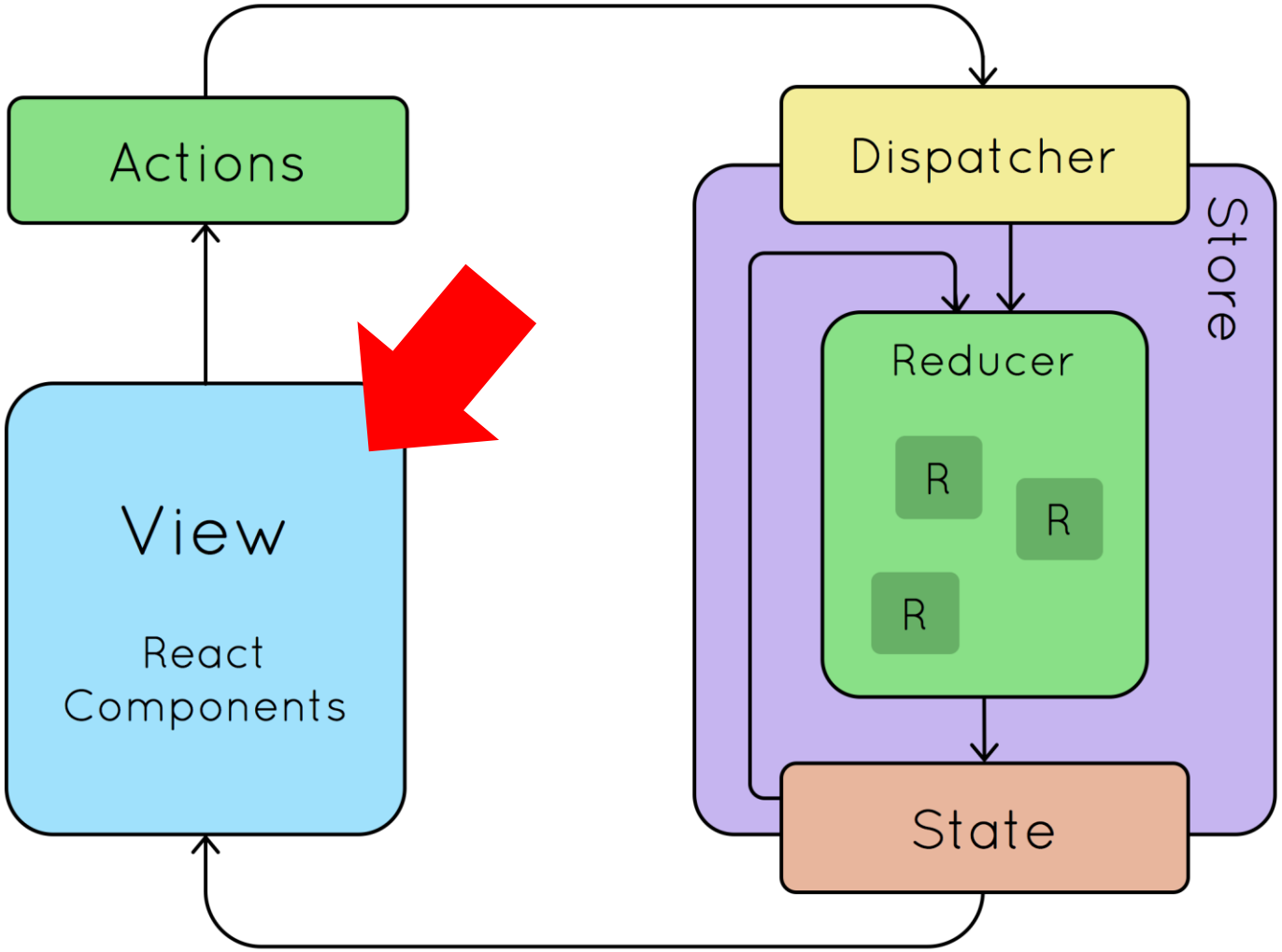
- 실제적으로 State 값을 관리하는 Reducer 를 만들어 봅시다!
- 먼저 State 로 사용할 변수를 매개 변수로 전달해 주면 됩니다!
- 지금은 Reducer 에 상태 관리 기능(Action 에 따른 동작)을 제외하고 간단하게 State 를 전달하는 기능만 만들어 봅시다!

```
function reducer(state = weight) {
  return state;
}
```
Src/index.js

- 아직은 Action 에 따라 상태 값을 변경하는 기능은 없고, index.js 에 저장된 상태 값을 전역에 있는 컴포넌트에 전달하는 기능만을 수행 합니다!

Store 에 저장 된 값 받아오기!

Store 에 저장 된 값 받아오기!



- App.js 내부의 컴포넌트에서 Store 에 저장 되어있던 State 값을 받아 옵시다!

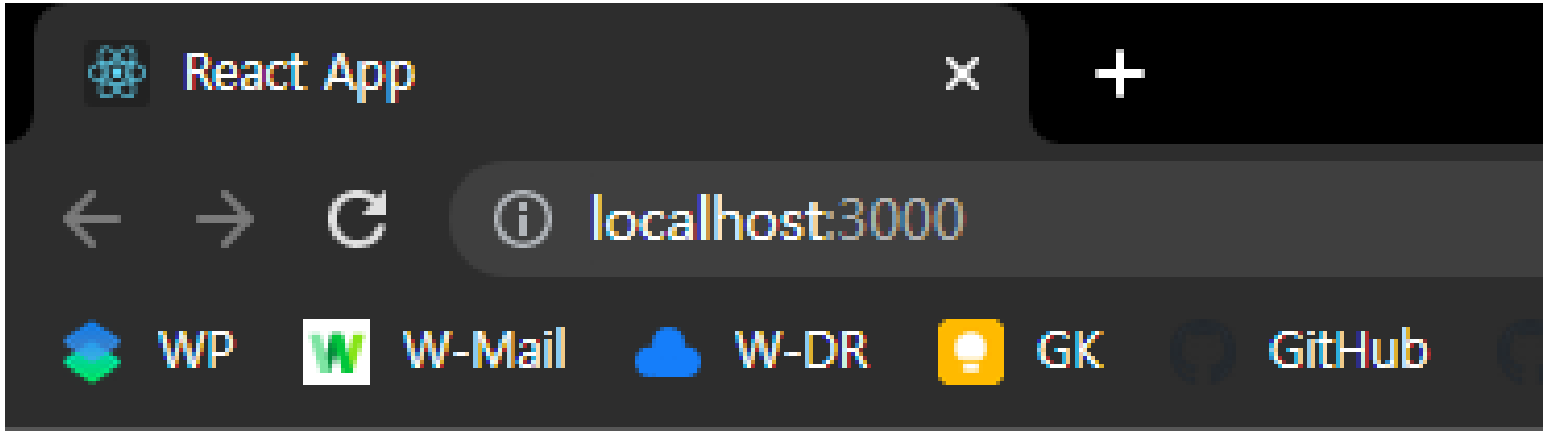
Store 의 상태 값을 받아올 컴포넌트 작성!

- Store 의 상태 값을 받아올 <Test> 컴포넌트를 작성해 봅시다!
- Store 의 상태 값을 받아올 때에는 React-redux 모듈의 useSelector 를 사용하면 됩니다!

```jsx
import React from 'react'
import { useSelector } from 'react-redux'

export default function Test() {
    const weight = useSelector((state) => state);

    return (
        <>
            <h1>당신의 몸무게는 {weight}</h1>
        </>
    )
}
```

Src/component/Test.js

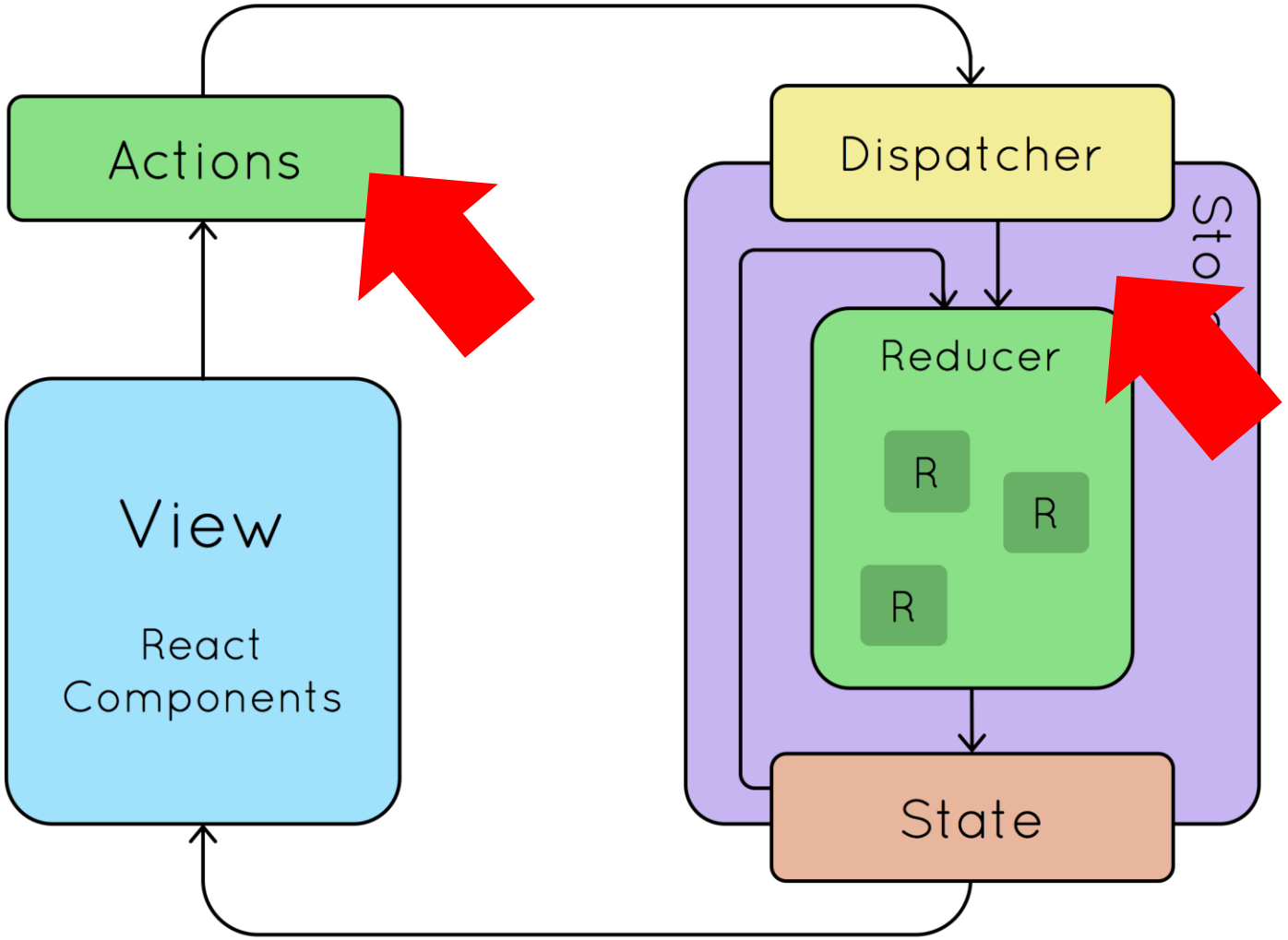
- weight 라는 변수에 Store 에 저장 되어있던 상태 값을 받고, 활용하면 됩니다!



당신의 몸무게는 100

Action & Reducer

Action 설정과 Dispatch 로 Action 보내기



- Store 의 State 값 변경을 위해서는 Action 을 설정
- Action 을 Dispatch 를 사용해서 Reducer 에 전달
- Reducer 가 State 를 변경

Action

Action & Reducer

- Action 은 Reducer 에게 어떤 처리를 해야하는지 알려주는 역할을 합니다!
- Action 은 객체 내부에 type 라는 키를 가지고 있으며, 해당 type 에는 리듀서에 전달할 액션을 “문자열“ 형태로 가지고 있습니다!
- Action 은 매개변수로 Reducer 에게 전달이 되며, Reducer 는 Action 객체 내부의 type 키 값의 문자열을 읽어서 State 를 어떤 방식으로 처리할 지 결정 합니다!

Action

```
{
    type: "some text"
}
```

Reducer

```
function reducer (state, action) {
    return changeState;
}
```

```
function reducer(state = weight, action) {
  if (action.type === "증가") {
    state++;
    return state;
  } else if (action.type === "감소") {
    state--;
    return state;
  } else {
    return state;
  }
}
```

Src/index.js

Action & Dispatch

- 컴포넌트에 있는 Action 을 index.js 에 있는 reducer 에 까지 보내려면 React-redux 에 있는 useDispatch() 를 사용해야 합니다!
- useDispatch() 를 하나의 변수에 담고 해당 변수를 통해 Action 의 값을 Reducer 로 전달해 주면 됩니다!
- Dispatch 로 전달 된 Action 의 type 의 값에 따라 Reducer 는 State 값을 변경하고, 해당 State 값은 컴포넌트에 반영 됩니다!

Dispatch

스토어 내장 함수.
액션을 발생시키는 함수.
Ex) dispatch(action)

```jsx
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'

export default function Test() {
    const weight = useSelector((state) => state);
    const dispatch = useDispatch();

    return (
        <>
            <h1>당신의 몸무게는 {weight}</h1>
            <button onClick={() => { dispatch({ type: "증가" }) }}>살 찌기</button>
            <button onClick={() => { dispatch({ type: "감소" }) }}>살 빼기</button>
        </>
    )
}
```

Src/component/Test.js

당신의 몸무게는 100

살 찌기 살 빼기

당신의 몸무게는 105

살 찌기 살 빼기

당신의 몸무게는 92

살 찌기 살 빼기

전체 코드

```jsx
import { Provider } from 'react-redux';
import { createStore } from 'redux';

const weight = 100;

function reducer(state = weight, action) {
  if (action.type === "증가") {
    state++;
    return state;
  } else if (action.type === "감소") {
    state--;
    return state;
  } else {
    return state;
  }
}

let store = createStore(reducer);

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

Src/index.js

```jsx
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'

export default function Test() {
    const weight = useSelector((state) => state);
    const dispatch = useDispatch();

    return (
        <>
            <h1>당신의 몸무게는 {weight}</h1>
            <button onClick={() => { dispatch({ type: "증가" }) }}>살 찌기</button>
            <button onClick={() => { dispatch({ type: "감소" }) }}>살 빼기</button>
        </>
    )
}
```

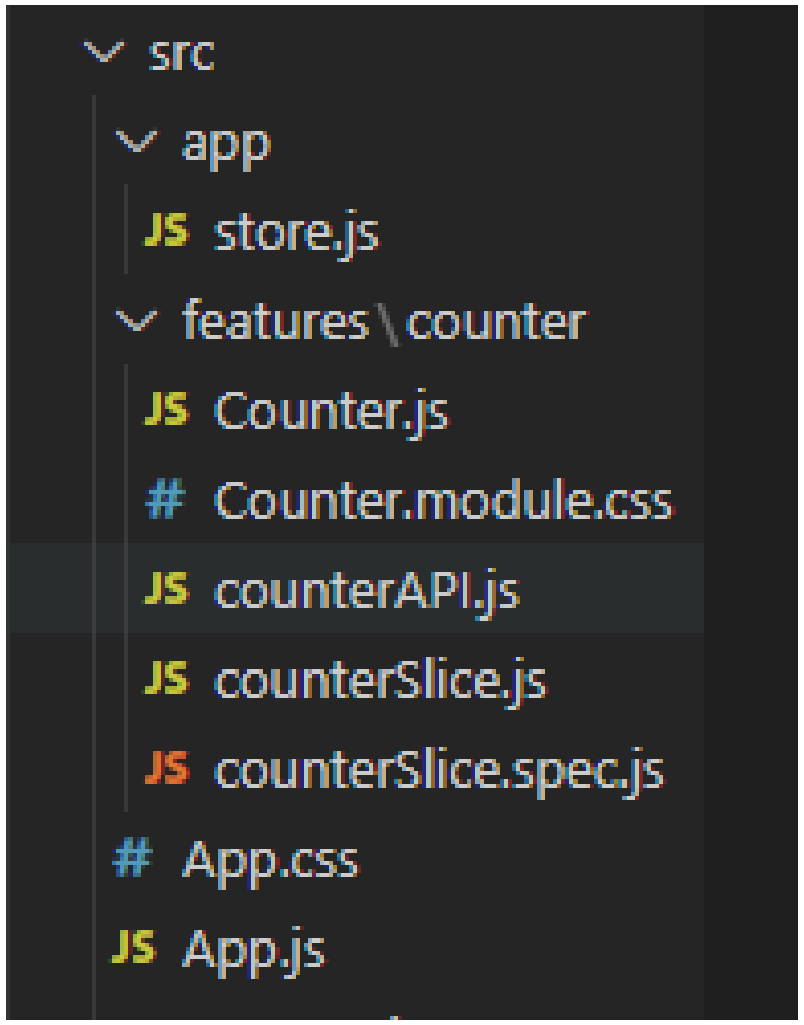
Src/component/Test.js

실전이야!

Redux Template

Redux Template

- Redux 를 빡세게 구현하는 방식은 대부분 비슷합니다!
- 따라서, 이러한 Redux 를 이용하는 React App 은 구조가 비슷하기 때문에 React App 을 만들 때 부터 Redux 를 위한 Template 을 적용하여 만들 수 있습니다!
- Npx create-react-app --template redux



- 0 +

2 Add Amount Add Async Add If Odd

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

Learn React, Redux, Redux Toolkit, and React Redux

```
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import { fetchCount } from './counterAPI';

const initialState = {
  value: 0,
  status: 'idle',
};

export const incrementAsync = createAsyncThunk(
  'counter/fetchCount',
  async (amount) => {
    const response = await fetchCount(amount);
    // The value we return becomes the `fulfilled` action payload
    return response.data;
  }
);
```

```
export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(incrementAsync.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(incrementAsync.fulfilled, (state, action) => {
        state.status = 'idle';
        state.value += action.payload;
      });
  },
});
```

React 입문자들이 알아야할 Redux 쉽게설명 (8분컷)

조회수 5.7만회 • 1년 전

코딩애플

React 하다보면 Redux를 필히 만나게 되는데 한 해 리덕스 포기자가 10만명이나 되기 때문에 준비했습니다 리액트 강의 ...


이렇게 된 이상
쌩 구현으로 간닷!
할
할
오케이!!


불현듯 떠오르는 라우터의
오싹 한 순간


어라분타
용서가됩다

그래서 우리가

만들 것은!?


What's the Plan for Today?
Add a todo
Add Todo
Like and Subscribe!
Wash the Dishes
Walk the Gorilla
Skydive over Everest
REACT
TODO LIST

Todo 리스트 만들기!

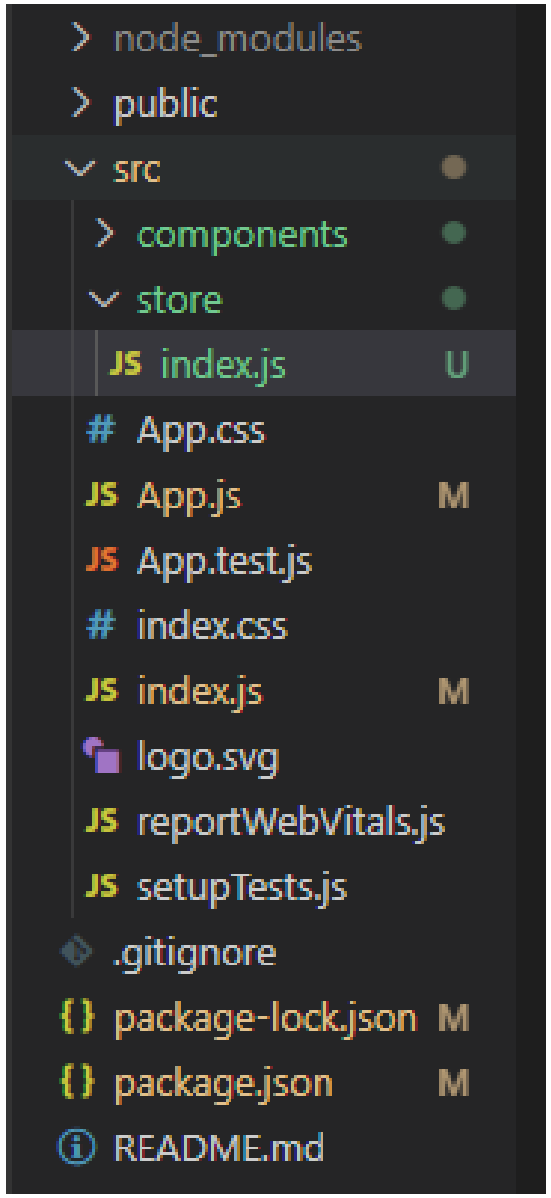
- 정말 간단하게 만들 겁니다!
- 할 일 추가, 할 일 완료를 누르면 완료 목록으로 옮기는 기능 정도만 추가해 볼게요!

Store

폴더 생성

Store 폴더 생성

- 기능 이전에 저장할 근간부터 만들어야 겠죠?
- Src 폴더 내부에 store 폴더를 만들어 주세요!
- Store 전체를 총괄하는 모듈은 index.js 가 담당할 예정입니다!

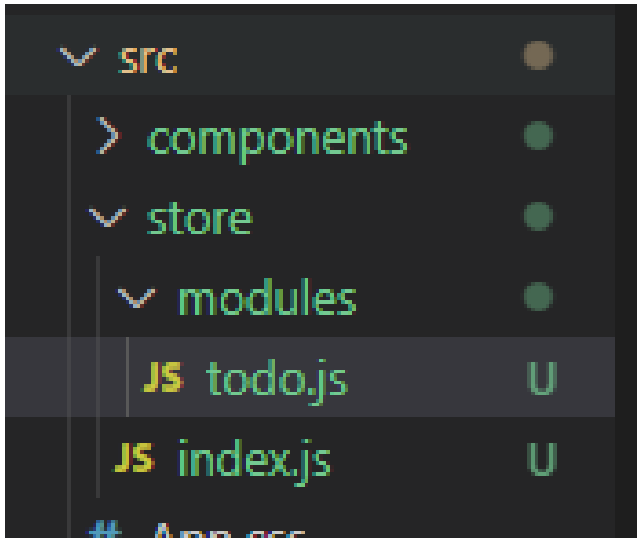

node_modules
public
src
components
store
index.js U
App.css
App.js M
App.test.js
index.css
index.js M
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json M
package.json M
README.md

Store 모듈 분할

- 모든 컴포넌트에 대한 글로벌 상태 값을 하나의 파일에서 관리한다면?
- 해당 파일이 하는 일이 너무 많겠죠?
- 당연히 코드 확장성 및 관리에도 어려움이 생깁니다! → 각 기능별 Store 모듈을 분할 합니다!

Store 모듈 분할

- 먼저 store 내부에 modules 폴더를 만들어 봅시다!
- 우리는 ToDo List 를 만들 것이므로 해당 리스트를 관리하는 모듈인 todo.js 모듈을 modules 폴더 내부에 만들어 줍시다!



초기 State 값 선언하기

최초의 State 값 설정

- 컴포넌트가 최초 렌더링 될 때 보여줘야할 최초의 State 값을 설정해 봅시다!
- 물론 DB 에서 데이터를 받아서 설정해 주는 방법이 맞지만, 이전 백엔드와 마찬가지로 편의를 위해서 변수로 설정해 봅시다!

최초의 State 값 설정

- Todo List 이므로 객체가 담긴 배열 형태로 선언할 예정입니다!
- List 객체에는 아래의 값이 구성 될 예정입니다!
  - id: 고유 id 값
  - text: 할 일 내용
  - done: 완료 여부

```
// 초기 상태 설정
const initState = {
  list: [
    {
      id: 0,
      text: '리액트 공부하기',
      done: false,
    },
    {
      id: 1,
      text: '척추의 요정이 말합니다! 척추 펴기!',
      done: false,
    },
    {
      id: 2,
      text: '취업 하기',
      done: false,
    },
  ],
};
```

Src/store/modules/todo.js

Reducer 로

값 리턴 시키기!

Reducer 를 통해 State 전달!

- 설정한 State 값을 외부에서 접근 하기 위해서는 Reducer 를 통해 값을 return 시켜줘야 합니다!
- 설정한 State 값을 바로 return 시켜주는 간단한 Reducer 를 작성해 봅시다!

```
export default function todo(state = initState, action) {
  return state;
}
```

Src/store/modules/todo.js

- 원래는 전달 된 2번째 매개 변수인 action 의 type 에 따라 다른 동작을 수행하는 것이 진짜 Reducer 입니다.
- 지금은 초기 State 값을 외부로 전달하는 목적만 달성하면 되므로 state 매개 변수에 initState 값을 넣어서 바로 return 시켜 주면 됩니다!

Store

통합 관리

Store 통합 관리!

- Store 는 모듈 별로 관리하고, 모듈 들은 Store 폴더의 index.js 에 의해서 통합 관리 됩니다!
- Store 폴더의 Index.js 파일에 가서 모듈 들을 통합 관리 해봅시다!
- 먼저 초기 값을 선언한 todo.js 를 import 해서 todo.js 의 reducer 를 불러오고(export default 로 설정 하였음) redux 의 combineReducer 를 이용하여 todo.js 의 reducer 를 하나로 합쳐서 다시 내보내 줍시다!

```
// 통합 관리 파일
import { combineReducers } from 'redux';
import todo from './modules/todo';

export default combineReducers({
  todo,
});
```

Src/store/index.js

- 각각의 Reducer 들을 합쳐주는 combineReducer 를 이용해서 각각의 store 모듈에서 export 된 reducer 를 합쳐 줍시다!
- 그리고 다시 합쳐진 reducer 를 export default 로 내보내 줍시다!

Redux 기초 세팅 및 Store 연결

Redux 기초 세팅!

- 라우팅 처리 하던 것처럼! Redux 적용을 위해서는 <Provider> 컴포넌트를 임포트하고 해당 컴포넌트로 <App> 컴포넌트를 감싸줘야만 합니다!
- Src 폴더의 최상위 Index.js 에 가서 코드를 처리!(Store 의 index.js X)
- combineReducer 를 통해 하나로 합쳐서 내보낸 Reducer 는 rootReducer 라는 값으로 받아 줍시다!

```
import { createStore } from 'redux’;
import { Provider } from 'react-redux';
import rootReducer from './store’;

const store = createStore(rootReducer);

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

Src/index.js

TodoList

컴포넌트 작성

Todo List 의 기본이 될 컴포넌트 만들기!

- 할 일 목록을 보여주고, 추가하는 기능을 가지는 → <TodoList> 컴포넌트
- 완료된 목록을 보여주는 → <DoneList> 컴포넌트
- 위의 두 컴포넌트를 “포함” 하여 전체 앱을 그려주는 <ListContainer> 컴포넌트를 제작해 봅시다!

TodoList

컴포넌트

<TodoList> 컴포넌트

- 할 일을 추가하는 Input 요소와, 추가 버튼 요소를 만들어 줍시다!
- 할 일 목록을 redux 를 통해 Store 에서 받아온 다음, 해당 목록을 <ul> 태그의 <li> 요소로 그려 줍시다!

```jsx
import { useRef } from 'react';
import { useSelector } from 'react-redux';

export default function TodoList() {
  const list = useSelector((state) => state.todo.list);
  const inputRef = useRef();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button>추가</button>
      </div>
      <ul>
        {list.map((el) => {
          return <li key={el.id}>{el.text}</li>;
        })}
      </ul>
    </section>
  );
}
```

Src/component/TodoList.js

DoneList

컴포넌트

<DoneList> 컴포넌트

- List 는 일단 useSelector 를 이용해서 state 값을 받아 옵시다!
- 완료된 List 를 받으면, 해당 List 를 ul 요소로 그려주면 도비니다!
- <TodoList> 에서 인풋 입력을 뺀 상태로 비슷하게 구현하면 됩니다!

```jsx
export default function DoneList() {
  const list = useSelector((state) => state.todo.list);
  return (
    <section>
      <h1>완료된 목록</h1>
      <ul>
        {list.map((el) => {
          return (
            <li key={el.id}>
              {el.text}
              <button>완료</button>
            </li>
          );
        })}
      </ul>
    </section>
  );
}
```

Src/component/DoneList.js

ListContainer

컴포넌트

<ListContainer> 컴포넌트

- <ListContainer> 컴포넌트는 <TodoList> 와 <DoneList> 를 순서대로 포함만 하면 되므로, 각각 컴포넌트를 Import 한 다음 자식 요소로 만들어 줍니다!

```jsx
import TodoList from './TodoList';
import DoneList from './DoneList';

export default function ListContainer() {
  return (
    <>
      <TodoList />
      <DoneList />
    </>
  );
}
```

Src/component/ListContainer.js

할일 목록

추가

- 리액트 공부하기 완료
- 척추의 요정이 말합니다 : 척추 펴기! 완료
- 취업 하기 완료

Redux 를 위한 편의 도구

편의 도구가 왜 필요하죠?

- Redux 에 저장 된 Store 의 값은 src 폴더의 index.js 파일에서 getState() 메소드를 이용하여 확인을 합니다!
- 지금은 간단한 Todo List 이기 때문에 하나의 값 만을 처리하고 있지만 프로젝트가 커지게 되면 다양한 전역 상태 값을 redux 로 관리 해야 합니다.
- 그럴때 마다 redux 의 모든 값을 하나하나 console.log 로 찍어가면서 확인하기는 매우 귀찮기 때문에 사용합니다!

```
import { createStore } from 'redux';
import rootReducer from './store';

import { Provider } from 'react-redux';

const store = createStore(rootReducer);
console.log(store.getState());

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

Src/index.js

```
▼Object i
  ▼todo:
    ▼list: Array(3)
      ▶0: {id: 0, text: '리액트 공부하기', done: false}
      ▶1: {id: 1, text: '척추의 요정이 말합니다 : 척추 펴기!', done: false}
      ▶2: {id: 2, text: '취업 하기', done: false}
        length: 3
      ▶[[Prototype]]: Array(0)
    ▶[[Prototype]]: Object
  ▶[[Prototype]]: Object
```

홈 > 확장 프로그램 > Redux DevTools

Redux DevTools

추천

★★★★★ 571 | 개발자 도구 | 사용자 1,000,000+명

Chrome에 추가

https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=ko

사용법!

- https://github.com/reduxjs/redux-devtools/tree/main/extension#installation

1.1 Basic store

For a basic Redux store simply add:

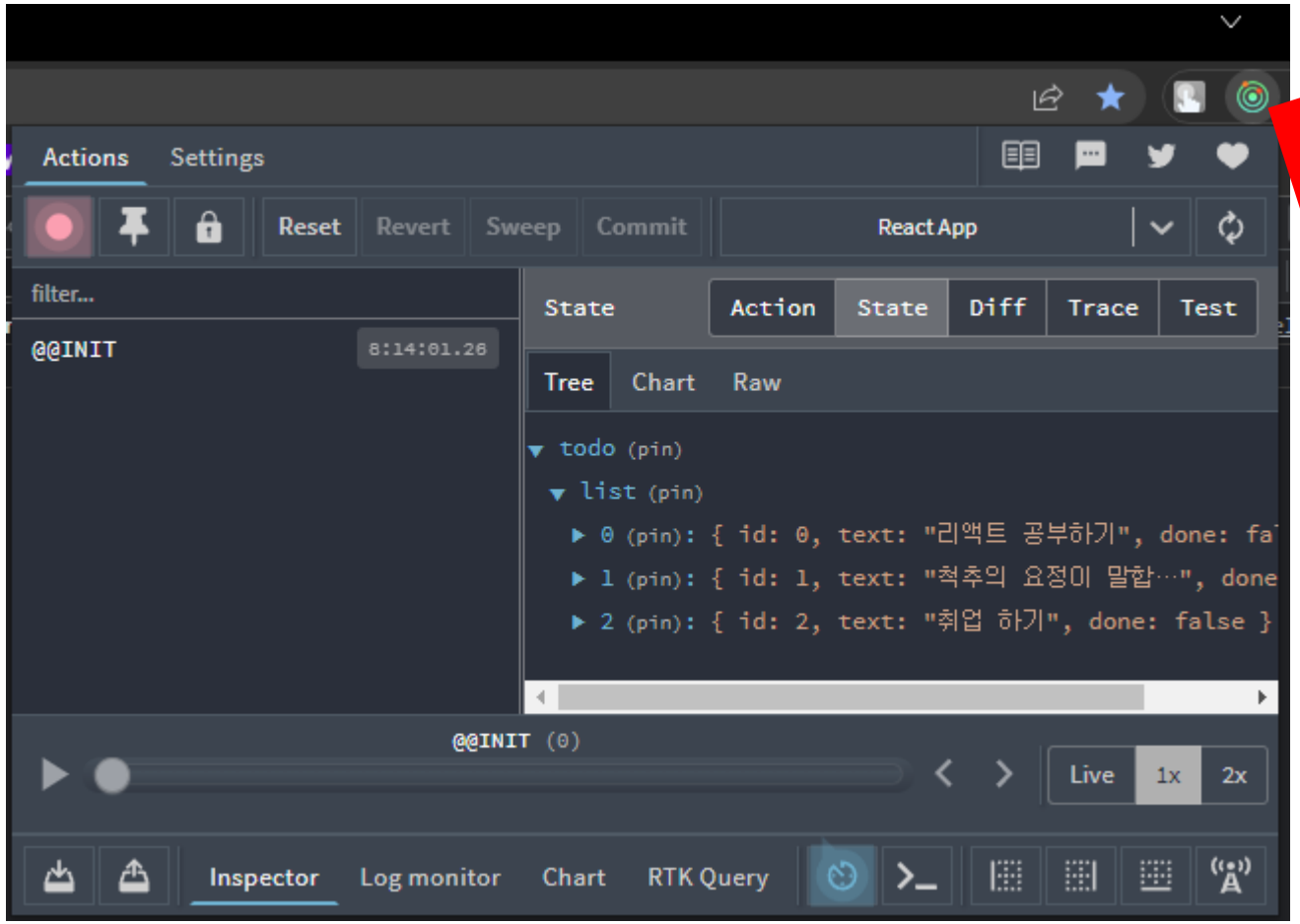
```
const store = createStore(
  reducer, /* preloadedState, */
+  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

사용법!

- redux store 를 만들 때, 약속 된 코드를 삽입해 주면 됩니다!

```
const reduxDevTool =
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__();

const store = createStore(rootReducer, reduxDevTool);
```

Src/index.js


Actions
Settings
Reset
Revert
Sweep
Commit
React App
filter...
@@INIT
8:14:01.26
State
Action
State
Diff
Trace
Test
Tree
Chart
Raw
todo (pin)
list (pin)
0 (pin): { id: 0, text: "리액트 공부하기", done: fa
1 (pin): { id: 1, text: "쳑추의 요정이 말합…", done
2 (pin): { id: 2, text: "취업 하기", done: false }
@@INIT (0)
Live
1x
2x
Inspector
Log monitor
Chart
RTK Query

Action 타입 설정

Action 타입 정의하기

- Action 타입은 “문자열”로 보통 정의합니다!
- Todo 리스트에 필요한 생성, 완료 액션을 정의합시다!

```
// 액션 타입 정의하기
const CREATE = "todo/CREATE";
const DONE = "todo/DONE";
```

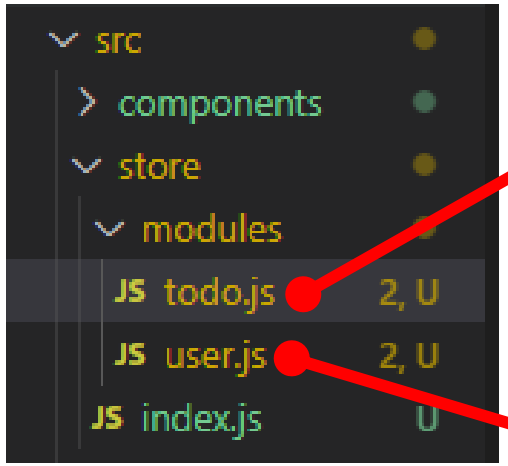
Src/store/modules/todo.js

Action 타입 정의하기

- 앞에 todo/ 는 왜 붙이는 건가요?
- 이 것은 잘못 된 사용을 막기 위한 하나의 방법입니다!
- 모듈이 달라도 Action 타입으로 CREATE, DELETE, DONE 같은 변수명은 상당히 많이 사용이 됩니다.
- 그럴 때 잘못 된 모듈 Import 로 인해, 다른 Reducer 의 기능이 호출되면 문제가 발생합니다

Action 타입 정의하기

- 이럴 때 Action 타입 앞에 지금 이 액션의 타입이 어떤 모듈의 타입인지를 알려주는 문자열을 추가하여 위와 같은 문제가 발생하는 것을 막아 줍니다!




```
// 액션 타입 정의하기
const CREATE = "todo/CREATE";
const DONE = "todo/DONE";
```

```
// 액션 타입 정의하기
const CREATE = "user/CREATE";
const DONE = "user/DONE";
```

Action 생성 함수 작성

Action 생성 함수 작성

- 외부 컴포넌트에서 Action 을 만들어주는 함수부터 작성을 해봅시다!
- Action 생성 함수는 type 정보와 전달해야 할 정보를 payload 객체에 담아서 Dispatch 를 통해 전달 합니다!
- 결과적으로 Reducer 가 Action 함수에 들어있는 type 을 확인해서 어떤 행동을 할지 정하고, payload 에 있는 데이터를 받아서 처리 합니다!

Action 생성 함수 작성 - Create

- 새로운 할 일 목록을 만드는 create 함수부터 작성해 봅시다!
- 먼저 Action type 설정 부터 CREATE 로 해줍니다!
- 그리고 전달 해야할 정보는 payload 라는 매개 변수에 담아서 전달 합니다!

```
// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}
```

Src/store/modules/todo.js

Action 생성 함수 작성 - Done

- 할 일을 완료하는 역할을 하는 Done 함수도 작성해 봅시다!
- 먼저 Action type 설정 부터 DONE 으로 해줍니다!
- 이번에는 새로운 정보를 전달 할 필요가 없이 어떤 목록이 완료 되었는지만 알면 되기 때문에 id 값만 전달 하면 됩니다!

```
export function done(id) {
  return {
    type: DONE,
    id,
  };
}
```

Src/store/modules/todo.js

```
// 액션 타입 정의하기
const CREATE = 'todo/CREATE';
const DONE = 'todo/DONE';

// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}

export function done(id) {
  return {
    type: DONE,
    id,
  };
}
```

외부에서 직접 요청하지 않고
Todo.js 의 함수를 import 해서
사용하는 이유는

이렇게 type 값 등을 외부에서는
알 수가 없기 때문에
약속 된 함수만 사용하여
접근하는 것이 편하기 때문입니다!

Src/store/modules/todo.js

전체 코드

Reducer

구조 구현

Action Type 에 따라 작동하는 Reducer

- 이제는 Action Type 에 따라 작동하는 Reducer 를 구현해 봅시다!
- 먼저, switch 문을 이용해서 action type 에 따라서 각각의 역할을 한 뒤 값을 return 하는 구조로 만들어 주시면 됩니다!

```
// 리듀서 설정(실제 작업은 이친구가 합니다!)
export default function todo(state = initState, action) {
  switch (action.type) {
    case CREATE:
      return console.log('CREATE 호출');
    case DONE:
      return console.log('DONE 호출');
    default:
      return state;
  }
}
```

Src/store/modules/todo.js

Dispatch 로 Action 함수 전달

Dispatch 로 Action 함수 전달

- 그럼 이번에는 컴포넌트에서 Dispatch 로 정의한 Action 함수를 Reducer 에 전달하여 정상적으로 호출이 되는지 확인해 봅시다!
- Dispatch 활용을 위해 useDispatch 를 dispatch 변수에 넣어주기!
- Src/store/modules/todo.js 에서 create, done 함수 불러오기!
- Dispatch 의 인자로 create, done 함수를 전달하여 호출 상태 확인!

```jsx
import { useRef } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { create, done } from './../store/modules/todo';

export default function TodoList() {
  const list = useSelector((state) => state.todo.list);

  const inputRef = useRef();
  const dispatch = useDispatch();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button
          onClick={() => {
            dispatch(create(''));
          }}
        >
          추가
        </button>
      </div>
      <ul>
        {list.map((el) => {
          return <li key={el.id}>{el.text}</li>;
        })}
      </ul>
    </section>
  );
}
```

Src/component/TodoList.js

```
Download the React DevTools for a better developmer
▶ {todo: {…}}
CREATE 호출
2 ▶ Uncaught Error: When called with an action of typ
an action, you must explicitly return the previous
    at combination (redux.js:564:1)
    at k (<anonymous>:2235:16)
    at D (<anonymous>:2251:13)
    at <anonymous>:2464:20
    at Object.dispatch (redux.js:288:1)
    at e (<anonymous>:2494:20)
    at onClick (TodoList.js:18:1)
    at HTMLUnknownElement.callCallback (react-dom.
    at Object.invokeGuardedCallbackDev (react-dom.
    at invokeGuardedCallback (react-dom.developmen
```

```
Download the React DevTools for a bet
▶ {todo: {…}}
DONE 호출
2 ▶ Uncaught Error: When called with ar
action, you must explicitly return t
    at combination (redux.js:564:1)
    at k (<anonymous>:2235:16)
    at D (<anonymous>:2251:13)
    at <anonymous>:2464:20
    at Object.dispatch (redux.js:288
    at e (<anonymous>:2494:20)
    at onClick (TodoList.js:18:1)
    at HTMLUnknownElement.callCallba
    at Object.invokeGuardedCallbackD
    at invokeGuardedCallback (react-
```

Reducer

CREATE 구현

Reducer 의 CREATE 동작 구현

- 이제 들어온 Action Type 에 따른 reducer 의 실제 동작을 구현해 봅시다!
- 먼저 CREATE 부터 구현을 해봅시다!
- 혹시 모를 다른 초기 값이 있을지 모르므로 state 를 전개 연산자로 먼저 리턴해 줍니다.
- List 의 경우는 새롭게 입력 받은 값을 list 의 배열에 넣어 주면 됩니다!

```
// 리듀서 설정(실제 작업은 이친구가 합니다!)
export default function todo(state = initState, action) {
  switch (action.type) {
    case CREATE:
      return {
        ...state,
        list: state.list.concat({
          id: action.payload.id,
          text: action.payload.text,
          done: false,
        }),
      };
    case DONE:
      return { };
    default:
      return state;
  }
}
```

Push 말고 Concat 을 사용하는 이유는?

지금은 list 라는 배열에 변경 된 값을 리턴해 줘야 하는 상황입니다!

Push 는 배열에 값을 추가하고 배열의 길이를 리턴해 주고, concat 은 값이 추가된 배열을 리턴해 줍니다!

따라서 push 를 쓰면 list 에는 숫자 값만 들어가므로 문제가 생깁니다!

Src/store/modules/todo.js

Dispatch 로

CREATE 호출

CREATE 호출

- 이번에는 CREATE 의 함수에 제대로 된 인자를 전달하여 정상적으로 기능이 작동하도록 해봅시다!
- 리듀서에서 할 일 목록 추가로 필요한 정보는 id 값과 새롭게 추가될 할 일의 text 값이 필요합니다 → 두 데이터를 객체에 담아서 인자로 전달해 봅시다!

```
<button
          onClick={() => {
            dispatch(create({ id: list.length, text: inputRef.current.value }));
            inputRef.current.value = '';
          }}
        >
          추가
        </button>
```

Src/component/TodoList.js

할일 목록

추가

- 리액트 공부하기
- 척추의 요정이 말합니다 : 척추 펴기!
- 취업 하기
- 추가가 되나요!?

Reducer

DONE 구현

Reducer 의 DONE 동작 구현

- 동일하게 list 이외의 초기 state 값은 그대로 전달이 되어야 하므로 전개 연산자를 사용!
- List 의 경우는 컴포넌트에서 전달 받은 id 값과 동일한 객체를 찾은 다음 해당 객체의 done 항목을 true 로 변경하면 됩니다!
- 이럴 때는 map() 을 쓰면 편합니다! map() 은 배열의 모든 값을 순회 하면서 배열의 값을 return 된 값으로 변경해 줍니다!

```
case DONE:
      return {
        ...state,
        list: state.list.map((el) => {
          if (el.id === action.id) {
            return {
              ...el,
              done: true,
            };
          } else {
            return el;
          }
        }),
      };
    default:
      return state;
```

Src/store/modules/todo.js

```
// 액션 타입 정의하기
const CREATE = 'todo/CREATE';
const DONE = 'todo/DONE';

// 액션 생성 함수 작성
export function create(payload) {
  return {
    type: CREATE,
    payload,
  };
}

export function done(id) {
  return {
    type: DONE,
    id,
  };
}

// 초기 상태 설정
const initState = {
  list: [
    {
      id: 0,
      text: '리액트 공부하기',
      done: false,
    },
    {
      id: 1,
      text: '척추의 요정이 말합니다 : 척추 펴기!',
      done: false,
    },
    {
      id: 2,
      text: '취업 하기',
      done: false,
    },
  ],
};
```

```
// 리듀서 설정(실제 작업은 이친구가 합니다!)
export default function todo(state = initState, action) {
  switch (action.type) {
    case CREATE:
      return {
        ...state,
        list: state.list.concat({
          id: action.payload.id,
          text: action.payload.text,
          done: false,
        }),
        nextID: action.payload.id + 1,
      };
    case DONE:
      return {
        ...state,
        list: state.list.map((el) => {
          if (el.id === action.id) {
            return {
              ...el,
              done: true,
            };
          } else {
            return el;
          }
        }),
      };
    default:
      return state;
  }
}
```

Src/store/modules/todo.js

전체 코드

Dispatch 로

DONE 호출

DONE 호출

- 이번에는 DONE 의 함수에 제대로 된 인자를 전달하여 정상적으로 기능이 작동하도록 해봅시다!
- 리듀서에서 완료 된 목록의 id 값만 받아서 해당 목록의 done 항목을 true 로 변경만 하면 됩니다!
- Done 함수에 인자로 id 값을 전달해 봅시다!

```
<ul>
    {list.map((el) => {
      return (
        <li key={el.id}>
          {el.text}
          <button
            onClick={() => {
              dispatch(done(el.id));
            }}
          >
            완료
          </button>
        </li>
      );
    })}
</ul>
```

Src/component/TodoList.js

각각 컴포넌트에

Filter 걸기!

Filter 처리!

- 지금 <TodoList> 컴포넌트와 <DoneList> 컴포넌트는 동일한 List 를 출력하고 있습니다!
- 이제 done 의 값을 통해 필터링 하여 <TodoList> 에는 할 일 목록만, <DoneList> 에는 완료 된 목록만 남겨 봅시다!

TodoList 컴포넌트

- <TodoList> 컴포넌트 List 의 항목 중에서 done 의 값이 false 인 친구들만 가져오면 됩니다!
- 배열의 filter 메소드는 조건식을 만족하는 배열만 남겨서 리턴해 주므로 해당 메소드를 사용 하면 됩니다!

```
export default function TodoList() {
  const list = useSelector((state) => state.todo.list).filter(
    (el) => el.done === false
  );
```

Src/component/TodoList.js

DoneList 컴포넌트

- <DoneList> 컴포넌트 List 의 항목 중에서 done 의 값이 true 인 친구들만 가져오면 됩니다!

```
export default function DoneList() {
  const list = useSelector((state) => state.todo.list).filter(
    (el) => el.done === true
  );
```

Src/component/DoneList.js


WE DID IT
YES
memegenerator.net


슬
조지고 올게!
하지만
언제나 조져지는 건
나였다

킹치만...

- 역시 테스트를 해보니 에러가 뜨네요!

```
▶Warning: Encountered two children with the same key,   react-dom.development.js:86
`0`. Keys should be unique so that components maintain their identity across
updates. Non-unique keys may cause children to be duplicated and/or omitted — the
behavior is unsupported and could change in a future version.
    at ul
    at section
    at DoneList (http://localhost:3000/static/js/bundle.js:103:72)
    at ListContainer
    at div
    at App
    at Provider (http://localhost:3000/static/js/bundle.js:37892:5)
```

원인은…

- List 요소의 key 값은 고유해야 하지만 고유하지 않아서 생기는 문제 입니다!
- TodoList 에서 할 일을 추가 할 때, Store 에서 받아온 list 의 length 값을 넘기고 있습니다 → 이미 완료를 몇 개 하면 list 의 길이가 짧아짐 → 새로 생성 되는 <li> 요소는 이전의 key 와 동일한 값을 가지게 됨 → 에러 발생!

해결책은…

- 할 일 목록의 id를 목록의 순번으로 부여를 하고 있으므로, 해당 순번도 store 에서 전역으로 관리하여 문제를 막아 봅시다!
- 이런 부분이 리얼 redux 실전 입니다! ☺

문제 해결하기!

ID 를 관리하기 위한 State 생성!

- Store 의 todo 모듈에 ID 관리를 위한 값을 설정해 봅시다!
- 일단 초기 List 의 길이 값을 구하고, 해당 값을 다음에 생성 될 할 일 목록의 ID 값으로 넘겨주는 구조를 그려 봅시다!

```
const initState = {
   // 초기 상태 설정
};

let counts = initState.list.length;
initState['nextID'] = counts;
```

Src/store/modules/todo.js

CREATE 리듀서에 해당 내용 추가!

- CREATE 액션이 호출 되면 nextID 의 값을 새로운 할 일 목록의 id 로 전달 되기 때문에, 그 다음에 CREATE 가 호출 되기 전에 nextID 값은 +1 상태가 되어야 합니다!
- 따라서, action 에서 받아온 id 값(이전 상태의 nextID 값)에 +1 을 해주면 됩니다!

```
export default function todo(state = initState, action) {
  switch (action.type) {
    case CREATE:
      return {
        ...state,
        list: state.list.concat({
          id: action.payload.id,
          text: action.payload.text,
          done: false,
        }),
        nextID: action.payload.id + 1,
      };
```

할 일 목록 추가 시

기능 수정

<TodoList> 컴포넌트 기능 수정

- 이제는 새롭게 만들어질 할 일 목록의 id 는 list.length 로 보내는 것이 아니라 Store 의 todo.js 모듈에서 받아오면 됩니다!
- 해당 값을 CREATE 액션 호출 시 전달해 주면, 리듀서에서 그 값을 받아 다음 할 일 목록의 id 값을 +1 시켜 주므로 논리적으로 문제 없이 구성이 가능합니다!

```
export default function TodoList() {
  const list = useSelector((state) => state.todo.list).filter(
    (el) => el.done === false
  );

  const nextID = useSelector((state) => state.todo.nextID);

  const inputRef = useRef();
  const dispatch = useDispatch();

  return (
    <section>
      <h1>할일 목록</h1>
      <div>
        <input type="text" ref={inputRef} />
        <button
          onClick={() => {
            dispatch(create({ id: nextID, text: inputRef.current.value }));
            inputRef.current.value = '';
          }}
        >
          추가
        </button>
      </div>
```


WE DID IT
YES
memegenerator.net


수고하셨습니다!