03장 중급 - React.js 주요 개념 이해하기

066 click 이벤트 사용하기(onClick)

- react에서는 html과 달리리 이벤트에 camelCase를 사용한다. onClick 이벤트는 특정 element가 클릭됐을 때 정의된 함수를 호출하는 방식으로 사용한다. html에서는 onclick으로 모두 소문자로 나타낸다.

- App.js 파일

[066/App.js]

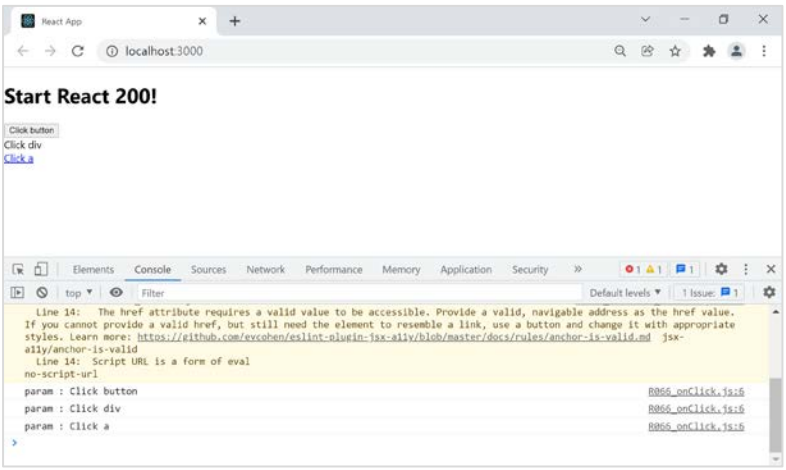
```
01  import React from 'react';
02  import ReactonClick from './R066_onClick'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ReactonClick/>
09      </div>
10    );
11  }
12
13  export default App;
```

- R066_onClick.js 파일

[066/R066_onClick.js]

```
01  import React, { Component } from 'react';
02
03  class R065_Promise extends Component {
04    buttonClick = (param) => {
05      if(typeof param != 'string') param = "Click a"
06      console.log('param : '+ param);
07    }
08
09    render() {
10      return (
11        <>
12          <button onClick={e => this.buttonClick("Click button")}>Click button</button>
13          <div onClick={e => this.buttonClick("Click div")}>Click div</div>
14          <a href="javascript:" onClick={this.buttonClick}>Click a</a>
15        </>
16      )
17    }
18  }
19
20  export default R065_Promise;
```

- 실행 결과



067 change 이벤트 사용하기(onChange)

- react에서는 onChange 이벤트도 camelCase 형식의 명칭을 사용한다. onChange 이벤트는 특정 element에 변화가 생겼을 때 정의된 함수를 호출하는 방식으로 사용한다.

- App.js 파일

[067/App.js]

```
01  import React from "react";
02  import MyComponent from "./067/src/App";
03
04  class App extends React.Component {
05    render() {
06      return (
07        <div className="body">
08          <MyComponent />
09        </div>
10      );
11    }
12  }
13
14  export default App;
```

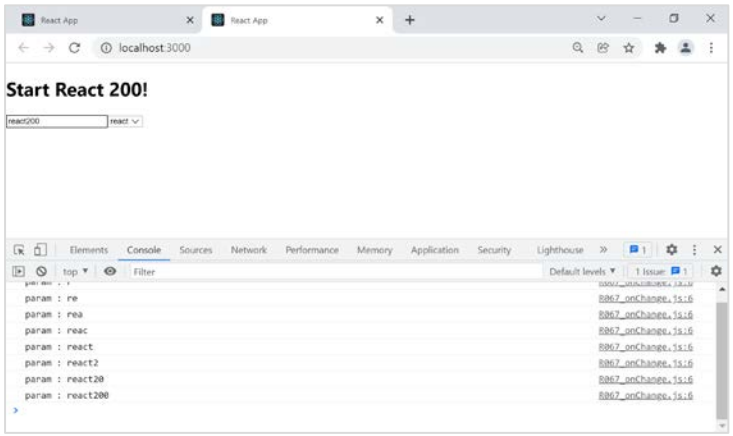
- R067_onChange.js 파일

[067/R067_onChange.js]

```
01  import React, { Component } from 'react';
02
03  class R067_onChange extends Component {
04    change = (e) => {
05      var val = e.target.value;
06      console.log('param : '+ val);
07    }
08
09    render() {
10      return (
11        <>
12          <input type="text" onChange={this.change}/>
13          <select onChange={this.change}>
```

```
14              <option value="react">react</option>
15              <option value="200">200</option>
16          </select>
17        </>
18      )
19    }
20  }
21
22  export default R067_onChange;
```

■ 실행 결과



068 mousemove 이벤트 사용하기(onMouseMove)

■ react에서는 onChange 이벤트도 camelCase 형식의 명칭을 사용한다. onChange 이벤트는 특정 element에 변화가 생겼을 때 정의된 함수를 호출하는 방식으로 사용한다.

■ App.js 파일

[068/App.js]

```
01  import React from 'react';
02  import ReactMouseMove from './R068_onMouseMove'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ReactMouseMove/>
09      </div>
10    );
11  }
12
13  export default App;
```

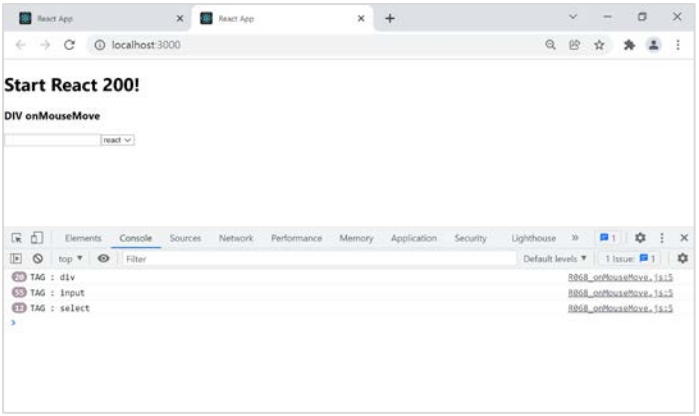
■ R068_onMouseMove.js 파일

[068/R068_onMouseMove.js]

```
01  import React, { Component } from 'react';
02
```

```
03  class R068_onMouseMove extends Component {
04    MouseMove(tag) {
05      console.log('TAG : '+tag);
06    }
07
08    render() {
09      return (
10        <>
11          <div onMouseMove={e => this.MouseMove("div")}>
12            <h3>DIV onMouseMove</h3>
13          </div>
14          <input type="text" onMouseMove={e => this.MouseMove("input")}/>
15          <select onMouseMove={e => this.MouseMove("select")}>
16              <option value="react">react</option>
17              <option value="200">200</option>
18          </select>
19        </>
20      )
21    }
22  }
23
24  export default R068_onMouseMove;
```

■ 실행 결과



069 mouseover 이벤트 사용하기(onMouseOver)

070 mouseout 이벤트 사용하기(onMouseOut)
071 key 이벤트 사용하기(onKeyDown, onKeyPress, onKeyUp)
072 submit 이벤트 사용하기(onSubmit)

073 Ref 사용하기

■ Ref는 ‘reference’의 약자로 ‘참조’라는 뜻이다. react에서 element의 값을 얻거나 수정할 때 보통 javascript나 jquery를 사용한다. 이때 id나 class와 같은 속성으로 element에 접근한다. Ref를 사용하면 element가 참조하는 변수에 접근해 변경하고 element를 제어할 수 있다.

■ App.js 파일

[073/App.js]

```
01  import React from 'react';
02  import ReactRef from './R073_ReactRef'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ReactRef/>
09      </div>
10    );
11  }
12
13  export default App;
```

■ R073_ReactRef.js 파일

- 06: createRef() 함수로 Ref 변수 InputRef를 생성한다.
- 20: element에 ref 속성을 추가하고 Ref 변수에 InputRef를 할당해 참조하도록 한다. 이때 참조에 대한 정보가 ref의 current라는 속성에 할당된다.
- 22: 이 버튼을 클릭하면 JavascriptFocus 함수가 실행된다. Javascript를 사용해 id 값으로 <input> 태그에 접근해 focus 이벤트를 발생시킨다.

[073/R073_ReactRef.js]

```
01  import React, { Component } from 'react';
02
03  class R073_ReactRef extends Component {
04    constructor(props) {
05      super(props);
06      this.InputRef = React.createRef();
07    }
08
09    RefFocus = (e) => {
10      this.InputRef.current.focus();
11    }
12
13    JavascriptFocus() {
14      document.getElementById('id').focus();
15    }
16
17    render() {
18      return (
19        <>
20          <input id="id" type="text" ref={this.InputRef} />
21          <input type="button" value="Ref Focus" onClick={this.RefFocus}/>
22          <input type="button" value="Javascript Focus" onClick={this.JavascriptFocus}/>
23        </>
24      )
25    }
26  }
27
28  export default R073_ReactRef;
```

■ 실행 결과

074 커링 함수 구현하기

- 커링(Currying)은 함수의 재사용성을 높이기 위해 함수 자체를 return하는 함수다. 함수를 하나만 사용할 때는 필요한 모든 파라미터를 한 번에 넣어야 한다. 커링을 사용하면 함수를 분리할 수 있으므로 파라미터도 나눠 전달할 수 있다.

- App.js 파일

[074/App.js]

```
01  import React from 'react';
02  import Currying from './R074_ReactCurrying'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <Currying/>
09      </div>
10    );
11  }
12
13  export default App;
```

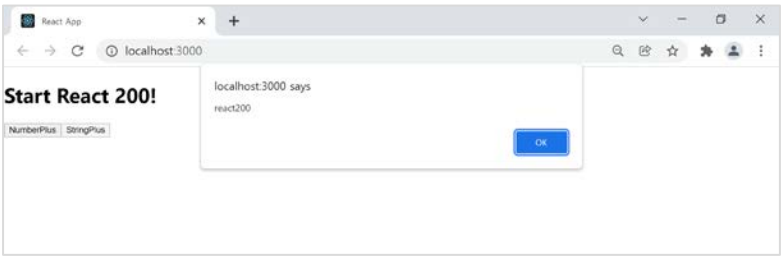
- R074_ReactCurrying.js 파일
  - 05~07: 파라미터로 전달된 c, d를 더해주는 함수를 생성한다. + 연산자는 숫자와 문자열에 사용할 수 있다.
  - 09~13: PlusFunc1 함수는 this.plusNumorString(a, b)를 return하는 또 다른 함수를 return한다.
  - 15: PlusFunc2 함수는 PlusFunc1 함수를 화살표 함수로 간단하게 표현한 것이다.
  - 17~19: PlusFunc 함수는 파라미터를 1개(a)만 받는다. PlusFunc1 함수를 호출할 때 변수 a와 상수 200을 파라미터로 함께 전달한다.

[074/R074_ReactCurrying.js]

```
01  import React, { Component } from 'react';
02
03  class R074_ReactCurrying extends Component {
04
05    plusNumOrString(c, d){
06      return c + d;
07    }
08
09    PlusFunc1(a){
10      return function(b){
11        return this.plusNumOrString(a, b)
12      }.bind(this)
```

```
13    }
14
15    PlusFunc2 = a => b => this.plusNumOrString(a, b)
16
17    PlusFunc(a){
18      return this.PlusFunc1(a)(200)
19    }
20
21    render() {
22      return (
23        <>
24          <input type="button" value="NumberPlus" onClick={e => alert(this.PlusFunc(100))}/>
25          <input type="button" value="StringPlus"
26          onClick={e => alert(this.PlusFunc("react"))}/>
27        </>
28      )
29    }
30  }
31
32  export default R074_ReactCurrying;
```

■ 실행 결과



075 하이오더 컴포넌트 구현하기

■ 커링과 같이 함수 자체를 인자로 받거나 반환하는 함수를 '고차 함수'라고 한다. 이와 비슷하게 컴포넌트를 인자로 받거나 반환하는 함수를 '고차 컴포넌트(HOC, Higher-Order Component)'라고 한다.

■ App.js 파일

[075/src/App.js]

```
01  import React from 'react';
02  import ReactHoc from './Hoc/R075_ReactHoc'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ReactHoc name='React200'/>
09      </div>
10    );
11  }
12
13  export default App;
```

■ R075_ReactHoc.js 파일

[075/src/Hoc/R075_ReactHoc.js]

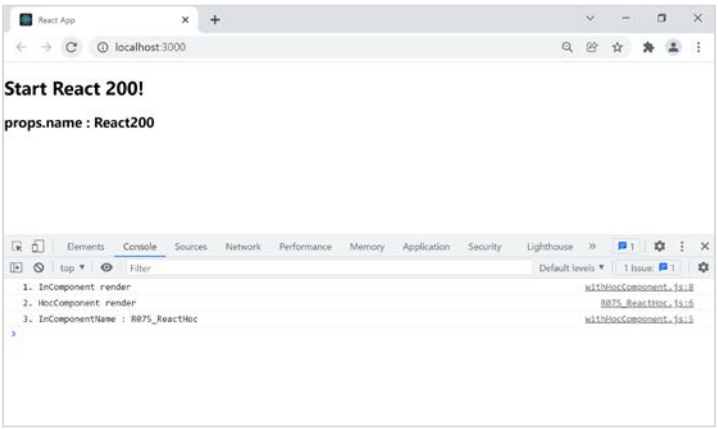
```
01  import React from 'react'
02  import withHocComponent from "./withHocComponent";
03
04  class R075_ReactHoc extends React.Component {
05      render () {
06        console.log('2. HocComponent render')
07        return (
08          <h2>props.name : {this.props.name}</h2>
09        )
10      }
11  }
12  export default withHocComponent(R075_ReactHoc, 'R075_ReactHoc')
```

■ withHocComponent.js 파일

[075/src/Hoc/R075_ReactHoc.js]

```
01  import React from "react";
02  export default function withHocComponent(InComponent, InComponentName) {
03      return class OutComponent extends React.Component {
04          componentDidMount () {
05              console.log(`3. InComponentName : ${InComponentName}`)
06          }
07          render () {
08              console.log('1. InComponent render')
09              return (<InComponent {...this.props}/>)
10          }
11      }
12  }
```

■ 실행 결과



076 컨텍스트 api 사용하기

■ 컨텍스트는 데이터의 공급자와 소비자를 정의하고 데이터가 필요한 컴포넌트만 사용할 수 있게 구현할 수 있다.

■ App.js 파일

[076/src/App.js]

```
01  import React from 'react';
02  import ContextApi from './Context/R076_ContextApi'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ContextApi/>
09      </div>
10    );
11  }
12
13  export default App;
```

■ R076_ContextApi.js 파일

- 04: 리액트 기본 제공 함수인 createContext를 호출하고 공급자 Provider와 소비자 Consumer를 받아 사용할 수 있도록 한다.
- 05: 하위 컴포넌트에서 소비자를 사용할 수 있도록 익스포트한다.
- 10~12: 자식 컴포넌트를 <Provider> 태그로 감싸고 전달할 데이터를 value 값으로 할당한다.

[076/src/Context/R076_ContextApi.js]

```
01  import React from 'react'
02  import Children  from "./contextChildren";
03
04  const {Provider, Consumer} = React.createContext()
05  export {Consumer}
06
07  class R076_ContextApi extends React.Component {
08    render () {
09      return (
10        <Provider value='React200'>
11          <Children />
12        </Provider>
13        )
14      }
15    }
16  export default R076_ContextApi
```

■ contextChildren.js 파일

[076/src/Context/contextChildren.js]

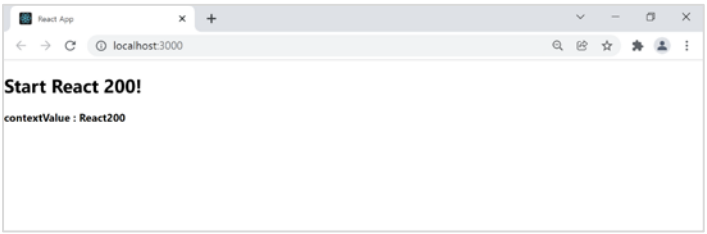
```
01  import React from 'react'
02  import Children2  from "./contextChildren2";
03
04  class contextChildren extends React.Component {
05      render () {
06        return (
07          <Children2 />
08        )
09      }
10  }
11  export default contextChildren
```

■ contextChildren2.js 파일
- 02: 부모 컴포넌트의 데이터를 사용하기 위해 R076_ContextApi 컴포넌트에서 익스포트했던 Consumer를 임포트해 사용할 수 있게 한다.
- 07~09: <Consumer> 태그로 출력할 element를 감싸고 R076_ContextApi 컴포넌트에서 value에 할당했던 데이터를 contextValue 변수로 받아 출력한다. 컨텍스트를 사용하면 몇 번째 하위 컴포넌트인지와는 상관없이 필요한 하위 컴포넌트에서 소비자를 임포트해 필요한 데이터를 사용할 수 있다.

[076/src/Context/contextChildren2.js]

```
01  import React from 'react'
02  import {Consumer}  from "./R076_ContextApi";
03
04  class contextChildren2 extends React.Component {
05      render () {
06        return (
07          <Consumer>
08              {contextValue=> <h3>{`contextValue : ${contextValue}`}</h3>}
09          </Consumer>
10        )
11      }
12  }
13  export default contextChildren2
```

■ 실행 결과



077 컨텍스트로 부모 데이터 변경하기

■ Props는 데이터가 부모에서 자식 컴포넌트로 단방향으로만 이동할 수 있다. 컨텍스트를 사용하면 자식 컴포넌트에서 부모 컴포넌트의 데이터를 변경할 수 있다.

■ App.js 파일

[077/src/App.js]

```
01  import React from 'react';
02  import ContextApi from './Context/R077_ContextApi'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <ContextApi/>
09      </div>
10    );
```

```
11  }
12
13  export default App;
```

■ R077_ContextApi.js 파일
- 12~14: state 변수 name에 파라미터 value를 할당하는 함수를 선언한다.
- 17~20: content 변수에 R0&&_ContextApi 컴포넌트의 state와 setStateFunc 함수를 할당한다.
- 22~24: 자식 컴포넌트를 <Provider> 태그로 감싸고 전달할 데이터인 content를 value 값으로 할당한다.

[077/src/Context/R077_ContextApi.js]

```
01  import React from 'react'
02  import Children  from "./contextChildren";
03
04  const {Provider, Consumer} = React.createContext()
05  export {Consumer}
06
07  class R077_ContextApi extends React.Component {
08    constructor (props) {
09      super(props);
10      this.setStateFunc = this.setStateFunc.bind(this)
11  }
12    setStateFunc(value) {
13      this.setState({name : value});
14    }
15
16    render () {
17      const content = {
18        ...this.state,
19        setStateFunc: this.setStateFunc
20      }
21      return (
22        <Provider value={content}>
23          <Children />
24        </Provider>
25        )
26      }
27    }
28  export default R077_ContextApi
```

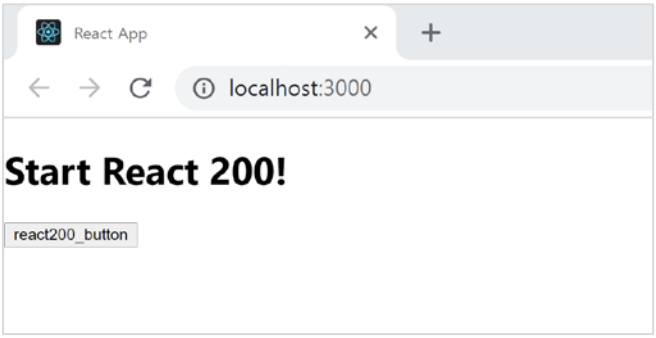
■ contextChildren.js 파일
- 07~13: <Consumer> 태그로 출력할 element를 감싸고 R077_ContextApi 컴포넌트에서 value에 할당했던 데이터를 contextValue 변수로 받아 사용한다.
- 09: 버튼을 클릭하면, 파라미터로 전달받은 R077_ContextApi 컴포넌트의 setStateFunc(" react200 ")을 호출한다. 애때 R077_ContextApi 컴포넌트의 state 변수 name 값을 reat200으로 할당한다.
- 10: 버튼을 누르기 전 contextValue.name 값이 없기 때문에 버튼명이 _button으로 표시된다. 버튼을 누르면 변경된 R077_ContextApi 컴포넌트의 state 변수 name 값인 react200을 가져오고 버튼명이 react200_button으로 표시된다.

[077/src/Context/contextChildren.js]

```
01  import React from 'react'
02  import {Consumer}  from "./R077_ContextApi";
03
04  class contextChildren extends React.Component {
```

```
05      render () {
06        return (
07          <Consumer>
08              {contextValue=>
09                <button onClick={e => contextValue.setStateFunc("react200")}>
10                  {contextValue.name}_button
11                </button>
12              }
13          </Consumer>
14        )
15      }
16  }
17  export default contextChildren
```

■ 실행 결과



078 redux 리듀서로 스토어 생성하기

■ Redux는 컨텍스트와 마찬가지로 데이터를 필요한 컴포넌트에서만 요청해 사용할 수 있다. 컨텍스트는 부모 컴포넌트에서 생성한 데이터에 모든 자식 컴포넌트에서 접근할 수 있다. 하지만 redux에서는 컴포넌트 외부의 스토어라는 곳에서 관리한다. 그래서 컴포넌트의 위치에 상관없이 스토어에 접근해 데이터를 사용하고 변경할 수 있다.

```
# 1. Create-react-app 설치
D:\dev\workspace\react>yarn create react-app client

// 2. redux를 설치한다.
D:\dev\workspace\react\client>yarn add redux
```

■ Index.js 파일
- 05: redux 패키지에서 스토어 생성 함수 createStore를 임포트해 사용할 수 있도록 한다.
- 08: createStore 함수의 파라미터로 reducers 폴더 경로를 넘긴다. reducers 폴더의 index.js에는 데이터 초기값을 설정하고 데이터를 변경해주는 함수가 있는데, 이 함수를 리듀서라고 한다.
- 10~15: 생성한 store를 App 컴포넌트에 전달한다. render 함수를 listener라는 함수 내부에 위치시킨다.
- 17: store를 구독하면 store 데이터에 변화가 있을 때 listener 함수 내부의 render 함수를 실행하고 변경된 데이터를 렌더링한다.
- 18: render 함수를 listener 함수로 감쌌기 때문에 초기 렌더링을 위해 수동으로 render 함수를 실해시켜준다.

[client/src/index.js]

```
01  import React from 'react';
02  import ReactDOM from 'react-dom';
03  import './index.css';
04  import App from './App';
05  import {createStore} from 'redux';
06  import reducers from './reducers';
07
08  const store = createStore(reducers);
09
10  const listener = ()=> {
11      ReactDOM.render(
12          <App store={store}/>,
13          document.getElementById('root')
14      );
15  };
16
17  store.subscribe(listener);
18  listener();
```

079 redux 스토어 상속과 디스패치 함수 실행하기

- props에 스토어를 담아 하위 컴포넌트로 전달하면, 전달받은 컴포넌트에서 스토어에 접근할 수 있다. 컴포넌트에서 dispatch 함수를 사용하면 스토어 데이터를 변경할 수 있다.

- App.js 파일
  - 02: 버튼 컴포넌트인 StrAddButton를 임포트해 사용할 수 있도록 한다.
  - 09: props를 통해 index.js에서 전달받은 store에 접근한다. 스토어 state 데이터에서 str 변수값을 가져온다.
  - 10: 버튼 컴포넌트 StrAddButton에 store를 전달한다.

[client/src/App.js]

```
01  import React, {Component} from 'react'
02  import StrAddButton from './StrAddButton'
03
04  class App extends Component {
05      render() {
06          return (
07              <div>
08                  <h1>Start React 200!</h1>
09                  <span>{this.props.store.getState().data.str}</span><br/>
10                  <StrAddButton store={this.props.store}/>
11              </div>
12          );
13      }
14  }
15
16  export default App;
```

- StrAddButton.js 파일
  - 02: actions 폴더 경로를 임포트한다. action 폴더의 index.js 파일에는 add라는 함수가 있다.
  - 07: 버튼을 클릭하면 addString 함수를 실행한다.

- 12: dispatch 함수를 통해 add 함수(actions 폴더의 index.js 파일)의 반환 값을 스토어에 전달한다.

[client/src/StrAddButton.js]

```
01  import React, {Component} from 'react';
02  import {add} from './actions'
03
04  class StrAddButton extends Component {
05      render() {
06          return (
07              <input value='Add200' type="button" onClick={this.addString}/>
08          )
09      }
10
11      addString = () => {
12          this.props.store.dispatch(add());
13      }
14  }
15
16  export default StrAddButton;
```

080 redux 리듀서에서 스토어 데이터 변경하기

■ 컴포넌트에서 dispatch 함수가 실행되면 리듀서 함수는 액션 값을 참조해 작업을 실행한다. 이때 액션 값에 따라 조건을 분기할 수 있다.

■ index.js 파일

[client/src/actions/index.js]

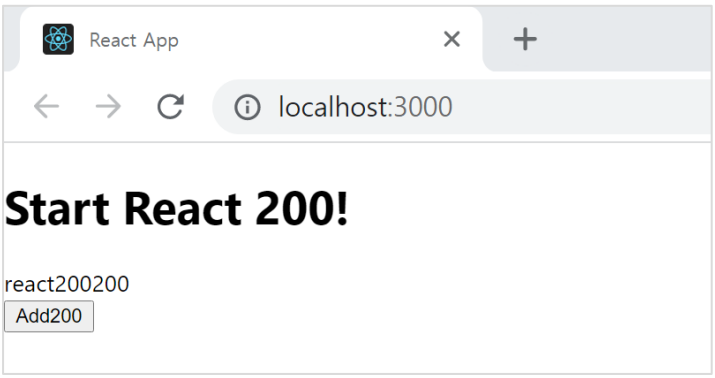
```
01  export const ADD = 'ADD';
02  export const add = () => {
03      return {
04          type: ADD
05      }
06  };
```

■ reducers 폴더의 index.js 파일을 다음과 같이 수정한다.
- 01: actions 폴더 경로를 임포트한다. actions 폴더의 index.js에서 ADD 변수값을 가져온다.
- 02: 리듀서를 스토어에 넘겨주기 위해 combineReducers 함수를 임포트한다.
- 04~06: 리듀서 데이터의 초기값을 선언, 할당한다.
- 08: state 변수에 line 5에서 할당한 초기값이 할당된다.
- 10~13: action.type 값이 ADD 값과 같은 경우, state 변수 str에 200을 붙인다. 반환 값은 line 8의 data 변수에 할당된다.
- 14~15: action.type 값이 ADD 값과 같지 않은 경우, state 변수를 그대로 반환한다.
- 19~23: 리듀스 함수 data를 combineReducers 함수를 이용해 하나의 리듀싱 함수로 변환하고 익스포트한다. 이 함수는 src의 index.js에 있는 createStore 함수의 파라미터로 넘겨진다. 스토어 state 값에 변경이 발생했기 때문에 subscribe 함수가 동작해 화면이 렌더링된다.

[client/src/reducers/index.js]

```
01  import {ADD} from '../actions'
02  import {combineReducers} from 'redux'
03
04  const initState = {
05      str: 'react',
06  };
07
08  const data = (state = initState, action) => {
09      switch (action.type) {
10          case ADD:
11              return state, {
12                  str: state.str + '200'
13              };
14          default:
15              return state;
16      }
17  };
18
19  const App = combineReducers({
20      data
21  });
22
23  export default App;
```

■ 실행 결과



081 react-redux로 스토어 상속하기

■ redux만 사용해도 충분히 스토어 데이터를 사용하고 변경할 수 있다. react-redux는 redux를 react와 연동해서 사용하기 편리하도록 만든 라이브러리다. react-redux의 장점은 크게 두 가지가 있다. 첫 번째는 store를 하위 컴포넌트에 매번 상속하지 않고 사용할 수 있다. 두 번째는 스토어 데이터를 사용, 변경하는 코드를 모듈화해 컴포넌트 내에 중복된 코드 사용을 최소화할 수 있다.

```
// react-redux를 설치한다.
D:\dev\workspace\react\client>yarn add react-redux
```

■ src 폴더의 index.js 파일을 다음과 같이 수정한다.
- 06: store 상속을 위해 react-redux의 Provider를 임포트해 사용할 수 있도록 한다.
- 13~15: <Provider> 태그로 App 컴포넌트를 감싸는 부분이 변경됐다. Provider에 데이터를 넘겨주면 중간 컴포넌트에서 props 값을 다시 전달해줄 필요 없이 모든 하위 컴포넌

트에서 데이터를 사용할 수 있다. 컨텍스트 app에서 사용했던 Provider와 동일한 기능을 한다.
- 14: App 컴포넌트에서 사용할 변수 indexProp에 react 문자열을 할당해 props로 전달한다.

[client/src/actions/index.js]

```
01  import React from 'react';
02  import ReactDOM from 'react-dom';
03  import './index.css';
04  import App from './App';
05  import {createStore} from 'redux';
06  import {Provider} from 'react-redux'
07  import reducers from './reducers';
08
09  const store = createStore(reducers);
10
11  const listener = ()=> {
12      ReactDOM.render(
13          <Provider store={store}>
14              <App indexProp="react"/>
15          </Provider>,
16          document.getElementById('root')
17      );
18  };
19
20  store.subscribe(listener);
21  listener();
```

082 react-redux로 스토어 데이터 사용하기

■ react-redux 패키지의 connect 함수는 파라미터를 4개까지 받을 수 있는데, 파라미터 위치에 따라 미리 정의된 함수나 object를 사용할 수 있다. 예제에서는 2개의 파라미터를 사용한다. 첫 번째 위치의 파라미터(mapStateToProps)는 스토어의 상태 값을 컴포넌트 props에 할당하는 함수이고, 두 번째 파라미터(mapDispatchToProps)는 dispatch 함수를 컴포넌트 함수에 바인딩하는 함수다.

■ src 폴더의 App.js 파일을 다음과 같이 수정한다.
- 02: react-redux 패키지에서 conect 함수를 임포트해 사용할 수 있도록 한다.
- 11: line 22에서 str 변수로 할당한 값을 화면에 출력한다. redux를 사용하면 line 10과 비교하면 컴포넌트 내에 코드가 짧아진 것을 확인할 수 있다. 접근하는 스토어 변수가 많아질수록 코드 효율이 더 좋아진다.
- 13: 기존 코드에서는 하위 버튼 컴포넌트에 store를 props로 다시 전달했지만, index.js에서 Provider를 사용했기 때문에 전달하지 않아도 된다. StrAddButton 컴포넌트에서 사용할 변수 AppProp에 200 문자열을 할당해 props로 전달한다.
- 19: mapStateToProps 함수는 첫 번째 파라미터로 스토어의 state 변수를, 두 번째 파라미터로 부모 컴포넌트에서 전달한 props 변수를 받는다.
- 20: index.js에서 전달한 props 변수 indexProp를 콘솔에 출력한다.
- 21~23: 스토어의 state 변수 str 값을 App 컴포넌트 props의 str 변수로 할당한다.
- 26: connect 함수의 첫 번째 파라미터는 mapStateToProps 함수로, 스토어의 state 값에 접근할 수 있다.

[client/src/App.js]

```
01  import React, {Component} from 'react'
02  import {connect} from 'react-redux'
03  import StrAddButton from './StrAddButton'
04
05  class App extends Component {
06      render() {
07          return (
08              <div>
09                  <h1>Start React 200!</h1>
10                  {/* <span>{this.props.store.getState().data.str}</span><br/> */}
11                  <span>{this.props.str}</span><br/>
12                  {/* <StrAddButton store={this.props.store}/> */}
13                  <StrAddButton AppProp="200"/>
14              </div>
15          );
16      }
17  }
18
19  let mapStateToProps = (state, props) => {
20      console.log('Props from indes.js : ' + props.indexProp)
21      return {
22          str: state.data.str,
23      };
24  };
25
26  App = connect(mapStateToProps, null)(App);
27
28  export default App;
```

083 react-redux로 스토어 데이터 변경하기

- connect 함수의 두 번째 파라미터 mapDispatchToProps 함수로 dispatch 함수를 컴포넌트 함수에 바인딩할 수 있다. 즉, 컴포넌트 함수가 실행되면 바인딩된 dispatch 함수가 실행된다.

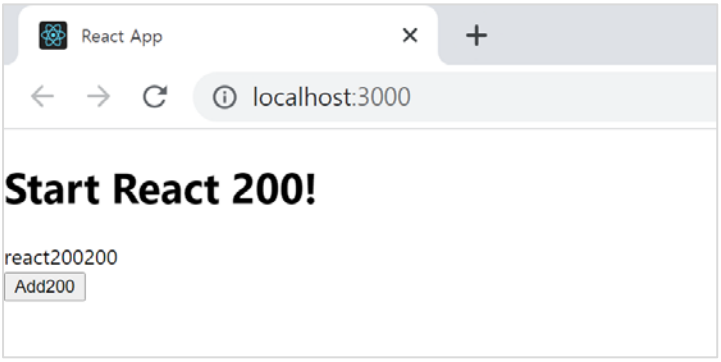
- src 폴더의 StrAddButton.js 파일을 다음과 같이 수정한다.

[client/src/StrAddButton.js]

```
01  import React, {Component} from 'react';
02  import {connect} from 'react-redux';
03  import {add} from './actions'
04
05  class StrAddButton extends Component {
06      render() {
07          return (
08              // <input value='Add200' type="button" onClick={this.addString}/>
09              <input value='Add200' type="button" onClick={this.props.addString}/>
10          )
11      }
12
13      // addString = () => {
14      //     this.props.store.dispatch(add());
15      // }
16  }
17
18  let mapDispatchToProps = (dispatch, props) => {
19      console.log('Props from App.js : ' + props.AppProp)
20      return {
21          addString: () => dispatch(add())
22      };
```

```
23  };
24
25  StrAddButton = connect(null, mapDispatchToProps)(StrAddButton);
26
27  export default StrAddButton;
```

■ 실행 결과



084 redux 미들웨어 사용하기

■ redux 미들웨어는 액션을 dispatch 함수로 전달하고 리듀서가 실행되기 전과 실행된 후에 처리되는 기능을 말한다. redux 패키지에서 지원하는 applyMiddleware 함수를 사용하면 미들웨어를 간단하게 구현할 수 있다.

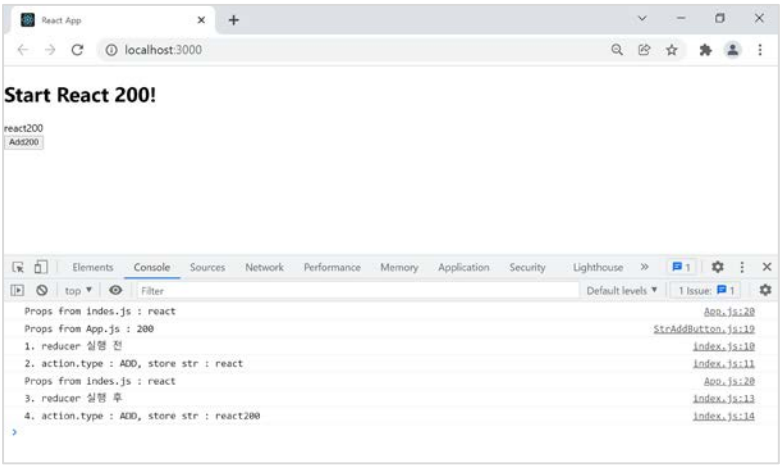
■ src 폴더의 index.js 파일을 다음과 같이 수정한다.

[client/src/index.js]

```
01  import React from 'react';
02  import ReactDOM from 'react-dom';
03  import './index.css';
04  import App from './App';
05  import {createStore, applyMiddleware} from 'redux';
06  import {Provider} from 'react-redux'
07  import reducers from './reducers';
08
09  const CallMiddleware = store => nextMiddle => action => {
10      console.log('1. reducer 실행 전');
11      console.log('2. action.type : '+action.type+', store str : '+store.getState().data.str);
12      let result = nextMiddle(action);
13      console.log('3. reducer 실행 후');
14      console.log('4. action.type : '+action.type+', store str : '+store.getState().data.str);
15      return result;
16  }
17
18  const store = createStore(reducers, applyMiddleware(CallMiddleware));
19
20  const listener = ()=> {
21      ReactDOM.render(
22          <Provider store={store}>
23              <App indexProp="react"/>
24          </Provider>,
25          document.getElementById('root')
26      );
27  };
```

```
28
29  store.subscribe(listener);
30  listener();
```

■ 실행 결과



085 react-cookies save 사용하기

■ redux 미들웨어는 액션을 dispatch 함수로 전달하고 리듀서가 실행되기 전과 실행된 후에 처리되는 기능을 말한다. redux 패키지에서 지원하는 applyMiddleware 함수를 사용하면 미들웨어를 간단하게 구현할 수 있다.

```
// react-cookies를 설치한다.
D:\dev\workspace\react\react200>yarn add react-cookies
```

■ src 폴더의 index.js 파일을 다음과 같이 수정한다.

[085/App.js]

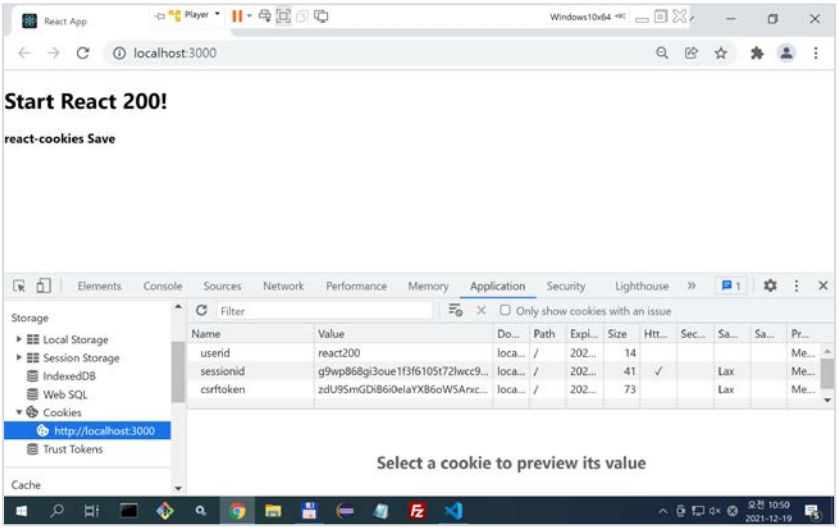
```
01  import React from 'react';
02  import CookieSave from './R085_cookieSave'
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <CookieSave/>
09      </div>
10    );
11  }
12
13  export default App;
```

■ src 폴더의 R085_cookieSave.js 파일을 다음과 같이 수정한다.

[085/R085_cookieSave.js]

```
01  import React, { Component } from 'react';
02  import cookie from 'react-cookies';
03
04  class R085_cookieSave extends Component {
05    componentDidMount() {
06      const expires = new Date()
07      expires.setMinutes(expires.getMinutes() + 60)
08      cookie.save('userid', "react200"
09          , {
10              path: '/',
11              expires,
12              // secure: true,
13              // httpOnly: true
14          }
15      );
16    }
17
18    render() {
19      return (
20        <><h3>react-cookies Save</h3></>
21      )
22    }
23  }
24
25  export default R085_cookieSave;
```

■ 실행 결과



086 react-cookies load 사용하기

087 react-cookies remove 사용하기

088 react-router-dom BrowserRouter 사용하기

■ 라우팅(routing)이란 호출되는 url에 따라 페이지(view) 이동을 설정하는 것이다. react에서 view는 component를 사용한다. react에서 라우팅을 지원하는 패키지는 여러 개 있지만, 웹 개발을 위한 패키지로 적합한 react-router-dom을 설치한다.

```
// react-router-dom을 설치한다.
D:\dev\workspace\react\client>yarn add react-router-dom@5.0.0
```

■ src 폴더의 index.js 파일을 다음과 같이 수정한다.

[client/src/index.js]

```
01  import React from 'react';
02  import ReactDOM from 'react-dom';
03  import { BrowserRouter } from 'react-router-dom';
04  import './index.css';
05  import App from './components/App';
06  import * as serviceWorker from './serviceWorker';
07
08  ReactDOM.render((
09    <BrowserRouter>
10        <App />
11    </BrowserRouter>
12  ), document.getElementById('root'));
13
14  // If you want your app to work offline and load faster, you can change
15  // unregister() to register() below. Note this comes with some pitfalls.
16  // Learn more about service workers: https://bit.ly/CRA-PWA
17  serviceWorker.unregister();
```

089 react-router-dom Route 사용하기

■ Route는 서버에 호출된 url의 path에 따라 연결할 component를 정의한다.

■ App.js 파일을 다음과 같이 수정한다.

[client/src/components/App.js]

```
01  import React, { Component } from 'react';
02  import { Route } from "react-router-dom";
03  import reactRouter from './R089_reactRouter'
04  import reactRouter2 from './R089_reactRouter2'
05
06  class App extends Component {
07    render () {
08      return (
09        <div className="App">
10          <Route exact path='/' component={reactRouter} />
11          <Route exact path='/reactRouter2' component={reactRouter2} />
12        </div>
13      );
14    }
15  }
16
17  export default App;
```

■ components 폴더에 R089_reactRouter.js 파일을 생성한 후 다음과 같이 입력한다.

[client/src/components/R089_reactRouter.js]

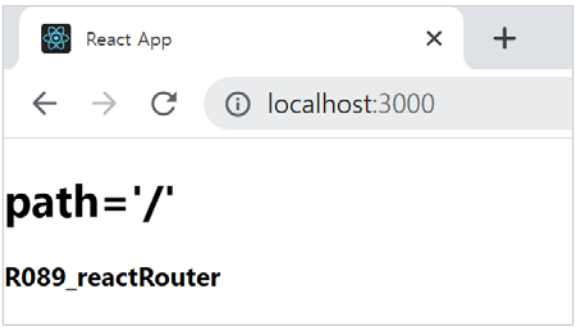
```
01  import React, { Component } from 'react';
02
03  class R089_reactRouter extends Component {
04    render() {
05      return (
06        <>
07          <h1>path='/'</h1>
08          <h3>R089_reactRouter</h3>
09        </>
10      )
11    }
12  }
13
14  export default R089_reactRouter;
```

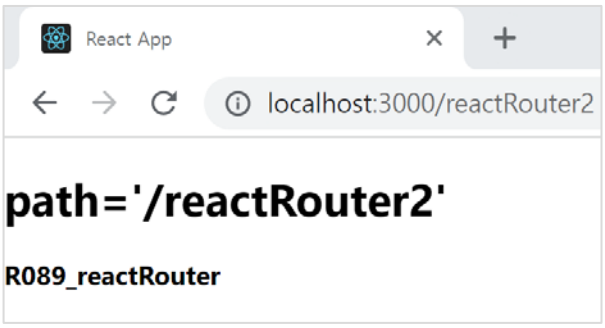
■ components 폴더에 R089_reactRouter2.js 파일을 생성한 후 다음과 같이 입력한다.

[client/src/components/R089_reactRouter2.js]

```
01  import React, { Component } from 'react';
02
03  class R089_reactRouter2 extends Component {
04    render() {
05      return (
06        <>
07          <h1>path='/reactRouter2'</h1>
08          <h3>R089_reactRouter</h3>
09        </>
10      )
11    }
12  }
13
14  export default R089_reactRouter2;
```

■ 실행 결과





090 react-router-dom Link 사용하기

■ Link는 <a> 태그와 동일하게 동작한다. <Route> 태그에 정의한 path를 Link 속성에 연결해 놓으면 링크를 클릭했을 때 라우팅된 컴포넌트로 이동한다.

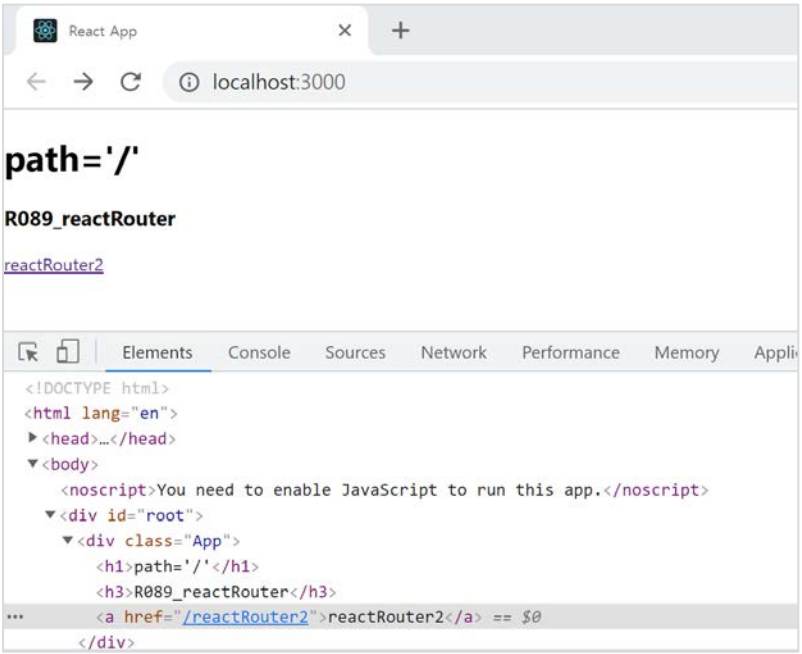
■ components 폴더에 R089_reactRouter.js 파일을 생성한 후 다음과 같이 입력한다.

- <Link> 태그를 추가하고 to 속성에 연결할 path를 입력한다. 이때 path은 App.js의 <Route> 태그에서 특정 컴포넌트로 라우팅 처리가 돼 있어야 한다.

[client/src/components/R089_reactRouter.js]

```
01  import React, { Component } from 'react';
02  import { Link } from 'react-router-dom'
03
04  class R089_reactRouter extends Component {
05    render() {
06      return (
07        <>
08          <h1>path='/'</h1>
09          <h3>R089_reactRouter</h3>
10          <Link to={'/reactRouter2'}>reactRouter2</Link>
11        </>
12      )
13    }
14  }
15
16  export default R089_reactRouter;
```

■ 실행 결과
- 개발자 도구의 [Element] 탭에서 <Link> 태그의 코드를 보면, 다음과 같이 <a> 태그인 것을 확인할 수 있다.



091 header, footer 구현하기

■ react-router-dom 패키지의 Route 기능을 사용하면, 호출되는 url에 따라 서로 다른 컴포넌트를 표시할 수 있다. 하지만 header와 footer는 라우팅과 상관없이 항상 표시돼야 하는 영역이다.

■ App.js 파일을 다음과 같이 수정한다.

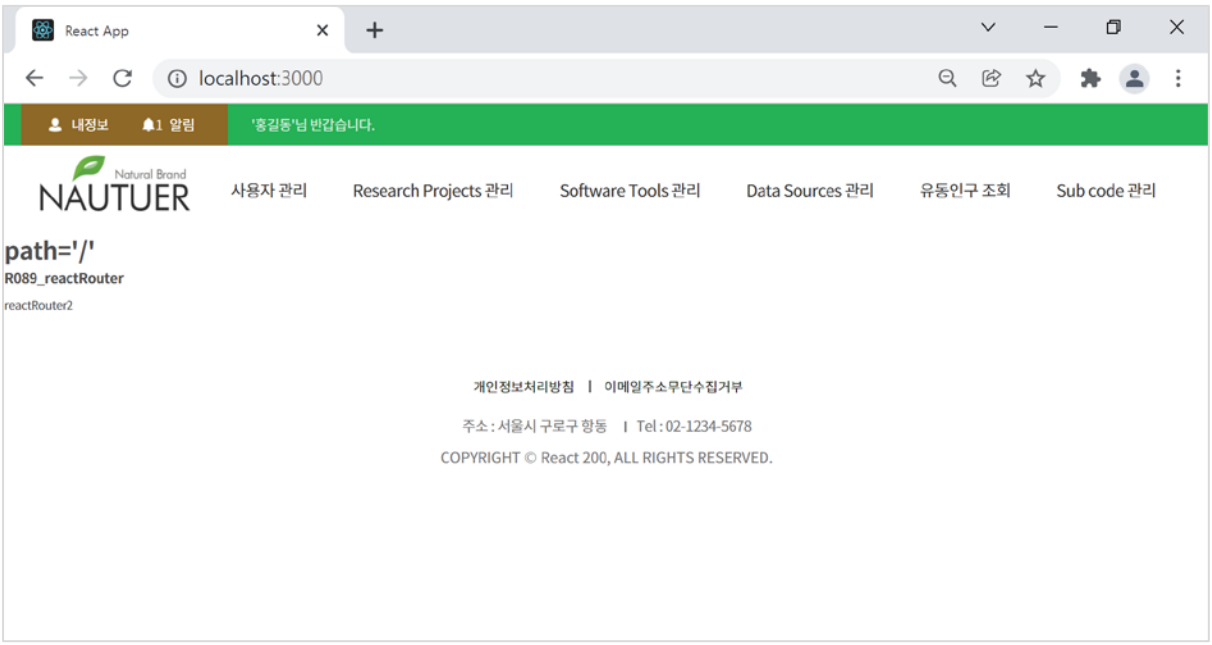
- 19: line 10에서 임포트한 header 컴포넌트를 <Route> 태그보다 위쪽에 위치시킨다.
- 22: line 13 에서 임포트한 header 컴포넌트를 <Route> 태그보다 아래쪽에 위치시킨다.

[client/src/components/App.js]

```
01  import React, { Component } from 'react';
02  import { Route } from "react-router-dom";
03  import reactRouter from './R089_reactRouter'
04  import reactRouter2 from './R089_reactRouter2'
05
06  // css
07  import '../css/new.css';
08
09  // header
10  import HeaderAdmin from './Header/Header admin';
11
12  // footer
13  import Footer from './Footer/Footer';
14
15  class App extends Component {
16    render () {
17      return (
18        <div className="App">
19          <HeaderAdmin/>
20          <Route exact path='/' component={reactRouter} />
21          <Route exact path='/reactRouter2' component={reactRouter2} />
22          <Footer/>
23        </div>
24      );
25    }
26  }
27
28  export default App;
```

■ 실행 결과

- 상단에 header가 하단에 footer 영역이 고정으로 위치한다. 그리고 중간 영역에 라우팅 되는 컴포넌트가 표시된다. 루트 경로(/)에서 /reactRouter2 경로로 url을 이동해도 중간에 표시되는 컴포넌트 영역만 변경된다.



092 home 페이지 만들기

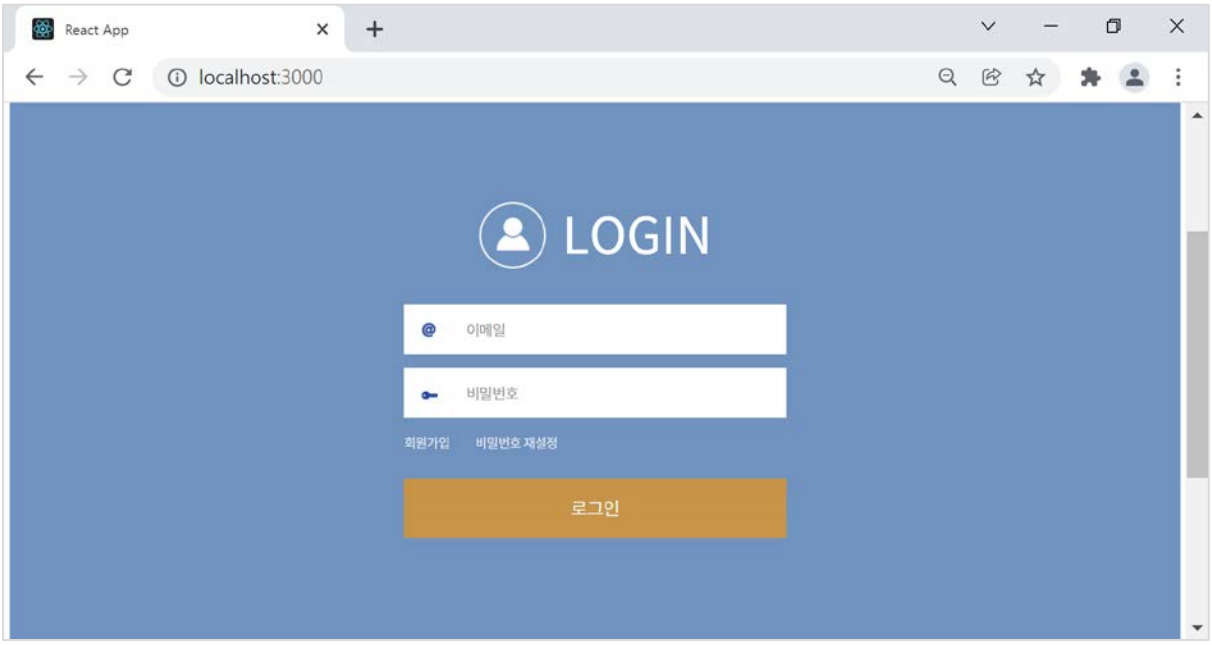
- home 페이지는 루트 경로(/)로 url을 호출했을 대 접속하게 되는 페이지다. home 화면을 구성하는 방법은 다양한데 예제에서는 로그인 페이지를 표시한다.

- App.js 파일을 다음과 같이 수정한다.

[client/src/components/App.js]

```
01  import React, { Component } from 'react';
02  import { Route } from "react-router-dom";
03
04  // css
05  import '../css/new.css';
06
07  // header
08  import HeaderAdmin from './Header/Header admin';
09
10  // footer
11  import Footer from './Footer/Footer';
12
13  // login
14  import LoginForm from './LoginForm';
15
16  class App extends Component {
17    render () {
18      return (
19        <div className="App">
20          <HeaderAdmin/>
21          <Route exact path='/' component={LoginForm} />
22          <Footer/>
23        </div>
24      );
25    }
26  }
27
28  export default App;
```

■ 실행 결과



093 <react img> 태그 사용하기

■ html에서 <img> 태그에 src 속성 값으로 이미지 경로를 직접 할당하면 화면에 이미지를 표시할 수 있다. react에서도 동일한 구조로 사용하지만 require 문법을 사용해 이미지 경로를 할당해야 한다.

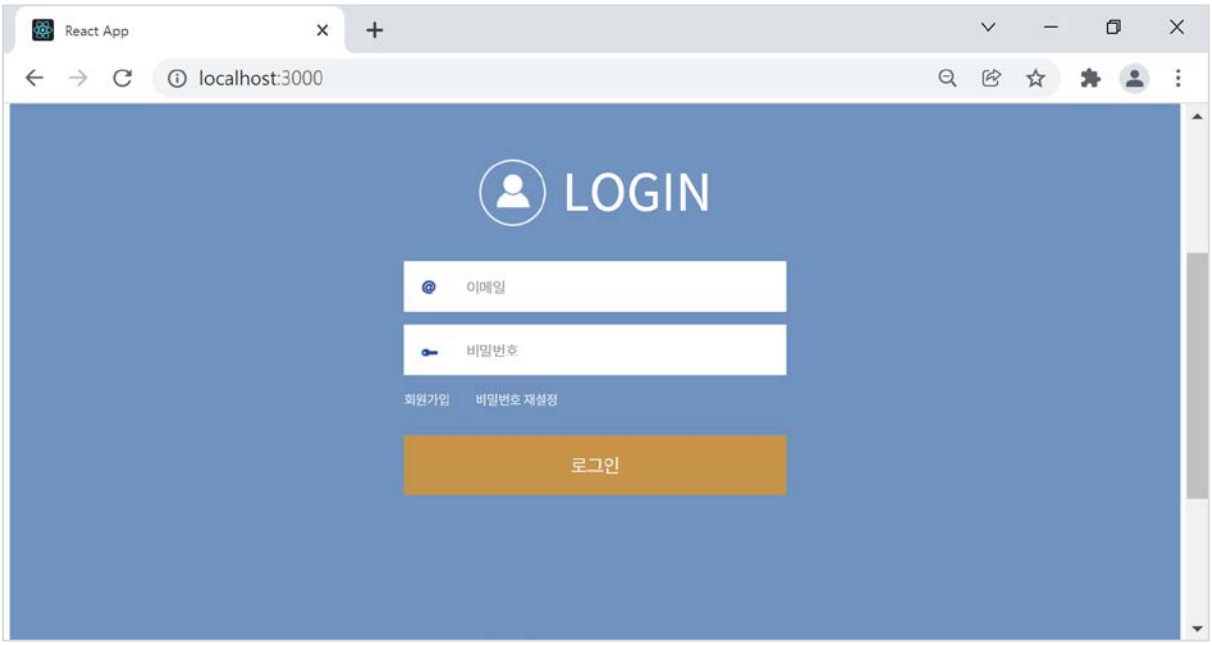
■ LoginForm.js 파일에서 <img> 태그 사용 방법을 확인한다.
- 09,13,18: <img 태그의 src 속성 값을 require 문법을 사용해 할당한다.

[client/src/components/LoginForm.js]

```
01  import React, { Component } from 'react';
02  import { Link } from 'react-router-dom'
03
04  class LoginForm extends Component {
05      render () {
06          return (
07              <section className="main">
08                  <div className="m_login">
09                  <h3><span><img src={require("../img/main/log_img.png")} alt="" /></span>LOGIN</h3>
10                  <div className="log_box">
11                      <form onSubmit={this.handleSubmit}>
12                      <div className="in_ty1">
13                          <span><img src={require("../img/main/m_log_i3.png")} alt="" /></span>
14                          <input type="text" id="email_val" name="email" placeholder="이메일"
15  onChange={this.handleChange} />
16                      </div>
17                      <div  className="in_ty1">
18                          <span className="ic_2"><img src={require("../img/main/m_log_i2.png")} alt=""
19  /></span>
20                          <input type="password" id="pwd_val" name="password" placeholder="비밀번호"
```

```
21  onChange={this.handleChange} />
22                      </div>
23                      <ul className="af">
24                          <li><Link to={'/register_check'}>회원가입</Link></li>
25                          <li className="pwr_b" onClick={this.pwdResetClick}><a href="#n">비밀번호
26  재설정</a></li>
27                      </ul>
28                      <button className="s_bt" type="submit" onClick={this.submitClick}>로그인</button>
29                      </form>
30                  </div>
31                  </div>
32              </section>
33          );
34      }
35  }
36
37  export default LoginForm;
```

■ 실행 결과



094 lodash 디바운스 사용하기

■ debounce는 연속된 이벤트 호출이 일어나는 상화에 사용한다. 마지막 이벤트가 실행되고 일정 시간 동안 추가 이벤트가 발생하지 않을 때 실행되는 함수다. debounce는 꼭 필요한 시점에만 함수를 실행해 서버 자원을 효율적으로 사용할 수 있게 해준다.

```
// lodash를 설치한다.
D:\dev\workspace\react\client>yarn add lodash
```

■ App.js 파일을 다음과 같이 수정한다.
- 24: App.js에서 라우팅돼 있는 /Debounce 경로를 호출한다.

[client/src/components/App.js]

```
01  import React, { Component } from 'react';
02  import { Route } from "react-router-dom";
03
04  // css
05  import '../css/new.css';
06
07  // header
08  import HeaderAdmin from './Header/Header admin';
09
10  // footer
11  import Footer from './Footer/Footer';
12
13  // login
14  import LoginForm from './LoginForm';
15
16  import reactDebounce from './R094_reactDebounce';
17
18  class App extends Component {
19    render () {
20      return (
21        <div className="App">
22          <HeaderAdmin/>
23          <Route exact path='/' component={LoginForm} />
24          <Route exact path='/Debounce' component={reactDebounce} />
25          <Footer/>
26        </div>
27      );
28    }
29  }
30
31  export default App;
```

■ R094_reactDebounce.js 파일

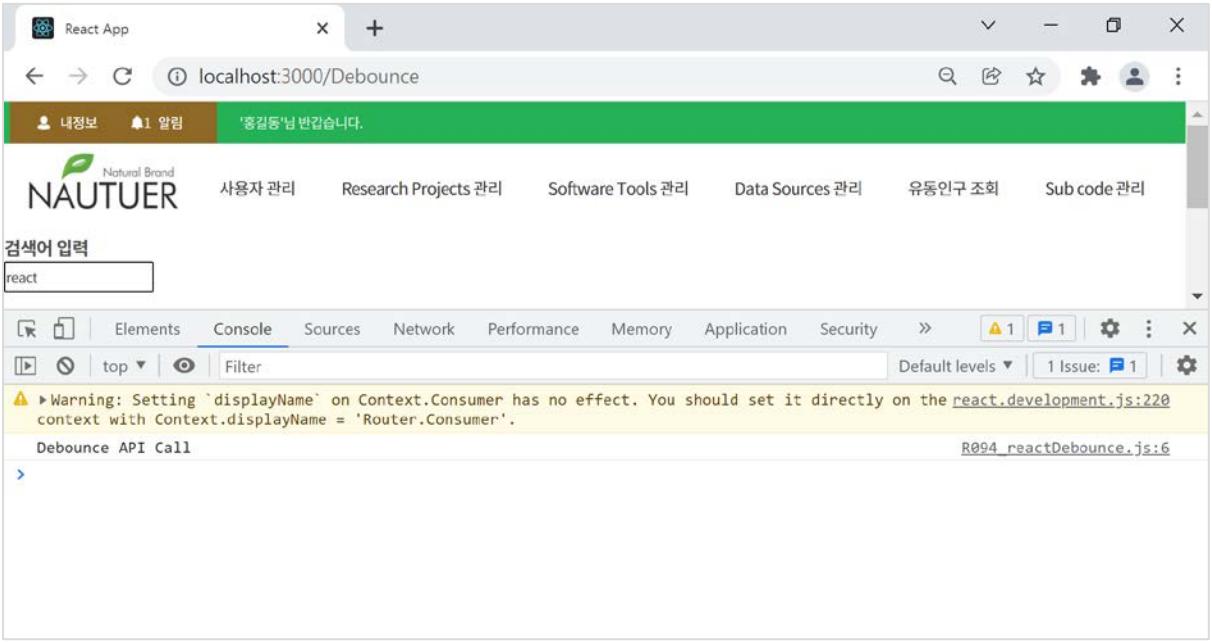
- 02: lodash 패키지를 임포트해 debounce 함수를 사용할 수 있도록 한다.
- 05~07: debounce 함수에 1초의 지연 시간을 할당한다. debounceFunc 함수는 글자가 입력될 때마다 호출되지만 debounce 함수는 마지막 호출이 끝나고 1초 후에 콘솔 로그를 출력한다.
- 13: <input> 태그에 텍스트를 입력할 때마다 debounceFunc 함수를 호출한다. 실제로 사용자가 입력한 텍스트가 포함된 검색어를 불러와야 할 때 line 6에 검색어 데이터를 호출하는 코드가 위치해야 한다. 이때 디바운스를 사용하지 않고 react라는 글자를 입력한다면 r, re, rea, reac, reat가 입력되는 시점에 모두 데이터를 호출해야 한다. 디바운스를 사용하면 react라는 글자가 모두 작성되고 지연 시간 1초가 지난 후 데이터를 한 번만 호출한다.

[client/src/components/R094_reactDebounce.js]

```
01  import React, { Component } from 'react';
02  import { debounce } from "lodash";
03
04  class R094_reactDebounce extends Component {
05    debounceFunc = debounce(() => {
06      console.log("Debounce API Call");
07    }, 1000);
08
09    render() {
10      return (
11        <>
12          <h2>검색어 입력</h2>
```

```
13          <input type="text" onChange={this.debounceFunc} />
14        </>
15      )
16    }
17  }
18
19  export default R094_reactDebounce;
```

■ 실행 결과



095 lodash 스로틀 사용하기

■ throttle도 debounce와 동일하게 시간 조건을 추가해 실행 횟수를 제한한다. 차이점은 debounce가 연속된 이벤트 중 마지막 이벤트가 발생한 시점부터 특정 시간이 지났을 때 동작한다면, throttle은 발생한 이벤트 수와 관계없이 지정한 시간 단위당 최대 한 번만 동작한다는 것이다.

■ R095_reactThrottle.js 파일
- 05~07: throttle 함수에 1초의 실행 시간 간격을 할당한다. throttleFunc 함수는 글자가 입력될 때마다 호출되지만 throttle 함수는 1초마다 0개 또는 1개의 로그를 출력한다. 만약 react라는 검색어를 0초(r), 0.3초(e), 0.5초(a), 1.1초(c), 2초(t)에 입력했다면, throttle는 0초(r), 1초(rea), 2초(react)에 1초 단위로 데이터 호출 코드를 실행한다.

[client/src/components/R095_reactThrottle.js]

```
01  import React, { Component } from 'react';
02  import { throttle } from "lodash";
03
04  class R095_reactThrottle extends Component {
05    throttleFunc = throttle(() => {
06      console.log("Throttle API Call");
07    }, 1000);
08
```

```
09    render() {
10      return (
11        <>
12          <h2>검색어 입력</h2>
13          <input type="text" onChange={this.throttleFunc} />
14        </>
15      )
16    }
17  }
18
19  export default R095_reactThrottle;
```

■ 실행 결과