02장 초급 - React.js 기초 다지기

017 props 사용하기

- 학습 내용: props 사용 방법을 이해한다.
- 힌트 내용: props에 데이터를 넣는 부분과 받아오는 부분을 확인한다.
- props는 부모 컴포넌트가 자식 컴포넌트에 데이터를 전달할 대 사용한다. props를 전달받은 자식 컴포넌트에서는 데이터를 수정할 수 없다. 데이터를 변경하기 위해서는 컴포넌트 내부에서만 사용하는 변수에 값을 넣어 사용해야 한다.
- App.js 파일
  - 10: line 3에서 임포트한 하위 컴포넌트 R017_Props에 전달할 props 변수(pros_val)에 값을 저장한다.

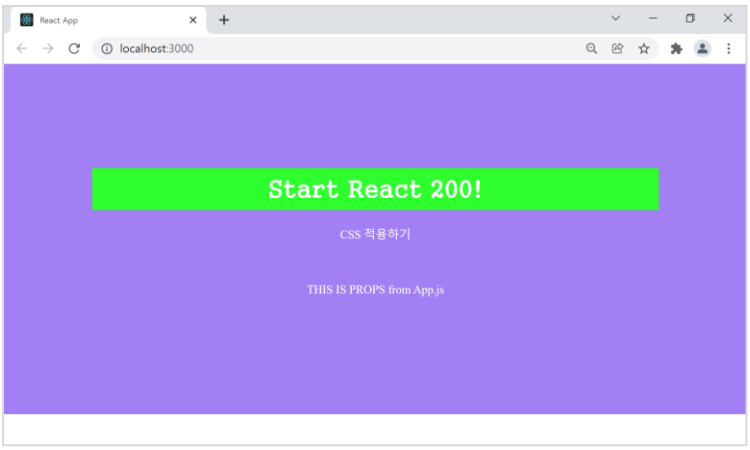
[react200/src/017/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import Props from './R017_Props'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <Props props_val="THIS IS PROPS"/>
11      </div>
12    );
13  }
14
15  export default App;
```

- R017_Props.js 파일

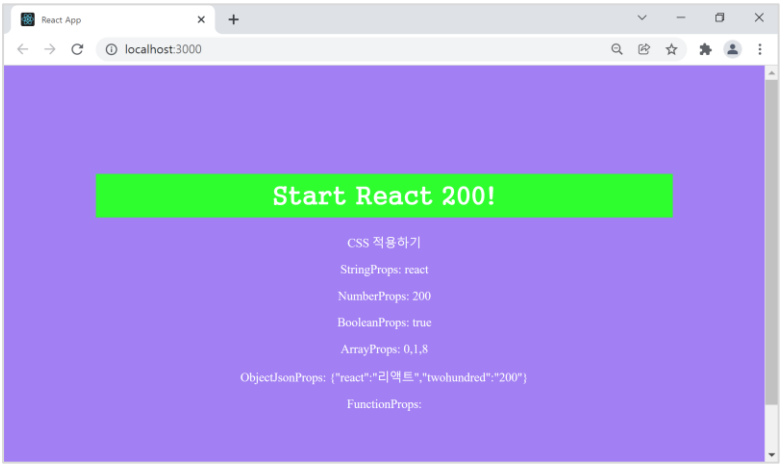
[react200/src/017/R017_Props.js]

```
01  import React, { Component } from 'react';
02
03  class R017_Props extends Component {
04    render() {
05      let props_value = this.props.props_val;
06      props_value += ' from App.js'
07      return (
08      <div>{props_value}</div>
09      )
10    }
11  }
12
13  export default R017_Props;
```



018 props 자료형 선언하기

ccccccccccccccc



019 props Boolean으로 사용하기

- 학습 내용: props Boolean의 사용 방법을 이해한다.
- 힌트 내용: props의 다른 자료형들이 갖고 있지 않은 Boolean 변수만의 특징을 확인한다.
- props 값을 Boolean 형으로 하위 컴포넌트에 전달할 경우, true나 false 중 하나를 할당한다. 추가 문법으로 props 변수를 선언한 후 값을 할당하지 않고 넘기면 true가 기본값으로 할당된다.
- App.js 파일

[react200/src/019/App.js]

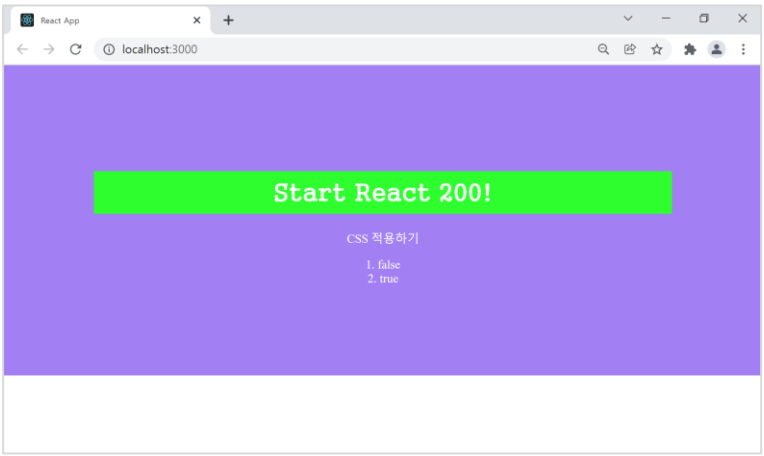
```
01  import React from 'react';
02  import './App.css';
03  import PropsBoolean from './R019_PropsBoolean'
04
05  function App() {
```

```
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PropsBoolean BooleanTrueFalse={false}/>
11        <PropsBoolean BooleanTrueFalse/>
12      </div>
13    );
14  }
15
16  export default App;
```

■ R019_PropsBoolean.js 파일

[react200/src/019/R019_PropsBoolean.js]

```
01  import React, { Component } from 'react';
02
03  class R019_PropsDatatype extends Component {
04    render() {
05      let {
06        BooleanTrueFalse
07      } = this.props
08      return (
09      <div style={{padding: "0px"}}>
10        {BooleanTrueFalse ? '2. ' : '1. '}
11        {BooleanTrueFalse.toString()}
12      </div>
13      )
14    }
15  }
16
17  export default R019_PropsDatatype;
```



020 props 객체형으로 사용하기

■ 학습 내용: props 객체형 사용 방법을 이해한다.

■ 힌트 내용: props 객체 자체의 자료형이 아닌 객체 내부 변수들의 자료형 선언 방법을 확인한다.

■ props 값을 객체로 하위 컴포넌트에 전달할 경우, 자료형을 object로 선언한다. 하지만 객체 형태(객체 내부 변수들)의 자료형을 선언할 때는 shape이라는 유형을 사용한다.

■ App.js 파일
- 10: ObjectJson 변수에 key와 value를 할당한 후 props에 담아 하위 컴포너트로 전달한다.

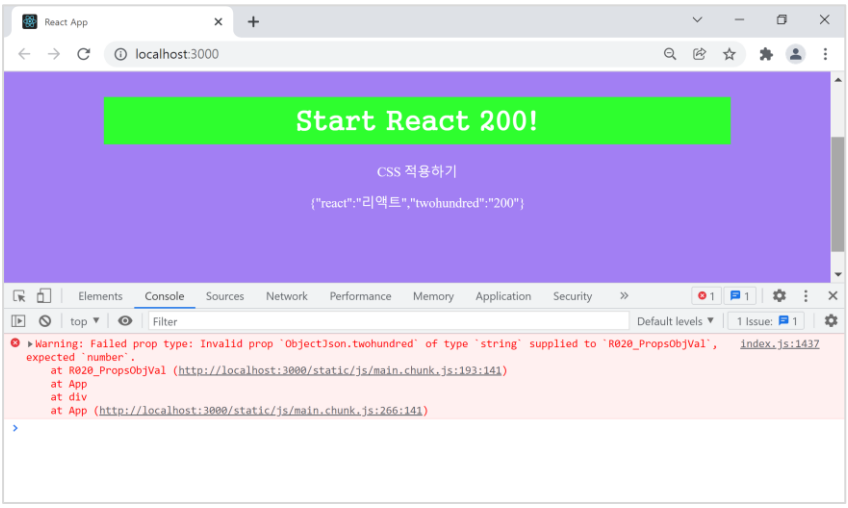
[react200/src/020/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import PropsObjVal from './R020_PropsObjVal'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PropsObjVal ObjectJson={{react:"리액트", twohundred:"200"}}/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R020_PropsObjVal.js 파일
- 06~08: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값을 할당한다.
- 11: ObjectJson 객체의 key와 value 값들을 화면에 출력한다.
- 18~21: shape 유형을 사용해 객체 변수 OjectJson의 내부 key 값에 대해 자료형을 선언한다. twohundred가 문자열(" 200 ")로 전달됐지만, 자료형이 number로 선언됐다. 자료형이 일치하지 않아 경고 메세지가 발생한다. 경고 메시지는 개발자 도구 콘솔 창에서 확인할 수 있다.

[react200/src/020/R020_PropsObjVal.js]

```
01    import React, { Component } from 'react';
02    import datatype from 'prop-types';
03
04    class R020_PropsObjVal extends Component {
05      render() {
06        let {
07          ObjectJson
08        } = this.props
09        return (
10        <div style={{padding: "0px"}}>
11          {JSON.stringify(ObjectJson)}
12        </div>
13        )
14      }
15    }
16
17    R020_PropsObjVal.propTypes = {
18      ObjectJson: datatype.shape({
19        react: datatype.string,
20        twohundred: datatype.number
21      })
22    }
23
24    export default R020_PropsObjVal;
```



021 props를 필수 값으로 사용하기

- 학습 내용: props를 필수 값으로 사용하는 방법을 이해한다.
- 힌트 내용: props를 필수 값으로 지정하는 문법을 확인한다.
- props의 자료형을 선언할 때 prop-types을 사용한다. 자료형 설정 대신 isRequired를 조건으로 추가하면, 변수값이 없는 경우 경고 메시지가 발생할 수 있다.
- App.js 파일

[react200/src/021/App.js]

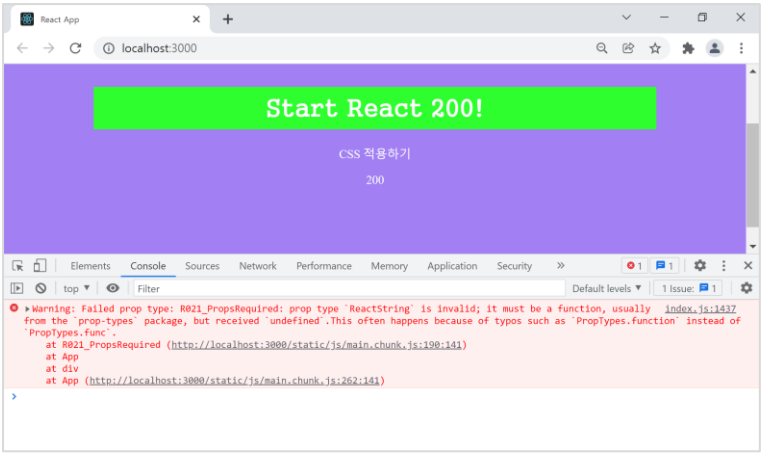
```
01  import React from 'react';
02  import './App.css';
03  import PropsRequired from './R021_PropsRequired'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PropsRequired ReactNumber={200}/>
11      </div>
12    );
13  }
14
15  export default App;
```

- R021_PropsRequired.js 파일
  - 18~20: ReactString이라는 props 값을 필수 값으로 지정한다. 하지만 상위 컴포넌트에서 ReactString이라는 변수를 전달하지 않았기 때문에 경고 메시지가 발생한다. 경고 메시지는 개발자 도구 콘솔 창에서 확인할 수 있다.

[react200/src/021/R021_PropsRequired.js]

```
01    import React, { Component } from 'react';
02    import datatype from 'prop-types';
03
04    class R021_PropsRequired extends Component {
05      render() {
06        let {
07          ReactString,
08          ReactNumber
09        } = this.props
10        return (
11        <div style={{padding: "0px"}}>
12          {ReactString}{ReactNumber}
13        </div>
14        )
15      }
16    }
17
18    R021_PropsRequired.propTypes = {
19      ReactString: datatype.isRequired,
20    }
21
22    export default R021_PropsRequired;
```

■ 실행 결과



022 props를 기본값으로 정의하기

■ 학습 내용: props를 기본값으로 사용하는 방법을 이해한다.

■ 힌트 내용: props를 기본값으로 지정하는 문법을 확인한다.

■ props의 기본값은 부모 컴포넌트에서 값이 넘어 오지 않았을 때 사용한다. defaultProps라는 문법을 사용한다.

■ App.js 파일

[react200/src/022/App.js]

```
01   import React from 'react';
02   import './App.css';
03   import PropsDefault from './R022_PropsDefault'
```

```
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PropsDefault ReactNumber={200}/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R022_PropsDefault.js 파일

- 17~20: 상위 컴포넌트에서 값이 전달될 것이라 기대되는 ReactString과 ReactNumber 변수에 각각 기본값을 할당했다.

[react200/src/022/R022_PropsDefault.js]

```
01    import React, { Component } from 'react';
02
03    class R022_PropsDefault extends Component {
04      render() {
05        let {
06          ReactString,
07          ReactNumber
08        } = this.props
09        return (
10        <div style={{padding: "0px"}}>
11          {ReactString}{ReactNumber}
12        </div>
13        )
14      }
15    }
16
17    R022_PropsDefault.defaultProps = {
18      ReactString: "리액트",
19      ReactNumber: 400
20    }
21
22    export default R022_PropsDefault;
```

■ 실행 결과

023 props의 자식 Component에 node 전달하기

■ 학습 내용: props에 node를 사용하는 방법을 이해한다.

■ 힌트 내용: props에 node를 넣어 전달하는 문법을 확인한다.

■ props를 하위 컴포넌트 태그 안쪽에 선언해 전달하는 것 이외에도 하위 컴포넌트 태그 사이에 작성된 node에 전달할 수 있다.

■ App.js 파일
- 10~12: 하위 컴포넌트 태그 사이에 태그를 추가하면 props에 담아 하위 컴포넌트에 전달한다.

[react200/src/023/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import PropsNode from './R023_PropsNode'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PropsNode>
11          <span>node from App.js</span>
12        </PropsNode>
13      </div>
14    );
15  }
16
17  export default App;
```

■ R023_PropsNode.js 파일
- 07: 상위 컴포넌트에서 전달한 노드는 this.props.children이라는 문법으로 접근해 사용할 수 있다.

[react200/src/023/R023_PropsNode.js]

```
01    import React, { Component } from 'react';
02
03    class R023_PropsNode extends Component {
04      render() {
05        return (
06        <div style={{padding: "0px"}}>
07          {this.props.children}
08        </div>
09        )
10      }
11    }
12
13    export default R023_PropsNode;
```

■ 실행 결과

024 state 사용하기

- 학습 내용: state 사용 방법을 이해한다.
- 힌트 내용: state 값을 화면에 표시하는 문법을 확인한다.
- props를 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달할 때 사용했다면, state는 하나의 컴포넌트 안에서 전역 변수처럼 사용한다.
- App.js 파일

[react200/src/024/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ReactState from './R024_ReactState'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ReactState reactString={"react"}/>
11      </div>
12    );
13  }
14
15  export default App;
```

- R024_ReactState.js 파일
  - 06~09: 가장 먼저 실행되는 생성자 함수 constructor 안에서 state 변수의 초기값을 정의해야 한다. SteString 변수에는 props로 전달된 reactString 값을 저장하고 StateNumber 변수에는 숫자 200을 저장한다.

[react200/src/024/R024_ReactState.js]

```
01    import React, { Component } from 'react';
02
03    class R024_ReactState extends Component {
04      constructor (props) {
05        super(props);
06        this.state = {
07            StateString: this.props.reactString,
08            StateNumber: 200,
09        }
10      }
11
```

```
12      render() {
13        return (
14        <div style={{padding: "0px"}}>
15          {this.state.StateString}{this.state.StateNumber}
16        </div>
17        )
18      }
19    }
20
21    export default R024_ReactState;
```

■ 실행 결과

025 setState() 함수 사용하기

■ 학습 내용: setState() 함수를 사용하는 방법을 이해한다.

■ 힌트 내용: state를 직접 변경했을 때와 setState() 함수를 사용했을 때의 차이를 확인한다.

■ this.state.변수명=value와 같이 state를 직접 변경하면 render() 함수를 호출하지 않으므로 화면에 보이는 state 값은 바뀌기 전 상태로 남게 된다. setState() 함수로 state로 변경해야 render() 함수를 호출해 변경된 값을 화면에 보여줄 수 있다.

■ App.js 파일

[react200/src/025/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import SetState from './R025_SetState'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <SetState/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R025_SetState.js 파일

[react200/src/025/R025_SetState.js]

```
01    import React, { Component } from 'react';
02
03    class R025_SetState extends Component {
04      constructor (props) {
05        super(props);
06        this.state = {
07            StateString: 'react',
08        }
09      }
10
11      StateChange = (flag) => {
12        if(flag == 'direct') this.state.StateString = '리액트';
13        if(flag == 'setstate') this.setState({StateString : '리액트'});
14      }
15
16      render() {
17        return (
18        <div style={{padding: "0px"}}>
19          <button onClick={(e) => this.StateChange('direct', e)}>state 직접 변경</button>
20          <button onClick={(e) => this.StateChange('setstate', e)}>setState로 변경</button><br/>
21          [state 변경하기] StateString : {this.state.StateString}
22        </div>
23        )
24      }
25    }
26
27    export default R025_SetState;
```

■ 실행 결과

026 state를 직접 변경한 후 forceUpdate() 함수 사용하기

■ 학습 내용: setState() 함수를 사용하는 방법을 이해한다.

■ 힌트 내용: state를 직접 변경했을 때와 setState() 함수를 사용했을 때의 차이를 확인한다.

■ this.state.변수명=value와 같이 state를 직접 변경하면 render() 함수를 호출하지 않으므로

화면에 보이는 state 값은 바뀌기 전 상태로 남게 된다. setState() 함수로 state로 변경해야 render() 함수를 호출해 변경된 값을 화면에 보여줄 수 있다.

■ App.js 파일

[react200/src/026/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ForceUpdate from './R026_ForceUpdate'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ForceUpdate/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R026_ForeUpdate.js 파일

- 13: forceUpdate() 함수는 화면을 강제로 새로 고침하기 때문에 render() 함수를 다시 실행시켜 화면에 변경된 state 값을 표시할 수 있다.

[react200/src/026/R026_ForceUpdate.js]

```
01    import React, { Component } from 'react';
02
03    class R026_ForceUpdate extends Component {
04      constructor (props) {
05        super(props);
06        this.state = {
07            StateString: 'react',
08        }
09      }
10
11      StateChange = () => {
12        this.state.StateString = '리액트';
13        this.forceUpdate();
14      }
15
16      render() {
17        return (
18        <div style={{padding: "0px"}}>
19          <button onClick={(e) => this.StateChange('direct', e)}>state 직접 변경</button><br/>
20          [state 변경하기] StateString : {this.state.StateString}
21        </div>
22        )
23      }
24    }
25
26    export default R026_ForceUpdate;
```

■ 실행 결과

027 Component 사용하기(class형 컴포넌트)

- 학습 내용: Class형 컴포넌트 중 Component를 사용하는 방법을 이해한다.
- 힌트 내용: Component와 render() 함수와의 관계를 확인한다.
- class형 컴포넌트에는 Component와 PureComponent가 있다. 두 컴포넌트 모두 props와 state의 변경에 따라 render() 함수를 호출하는데, 변경에 대한 기준이 다르다. Component에서는 비교 대상이 완전히 동일하지 않으면 변경이 발생했다고 본다.
- App.js 파일

[react200/src/027/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ComponentClass from './R027_ComponentClass'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ComponentClass/>
11      </div>
12    );
13  }
14
15  export default App;
```

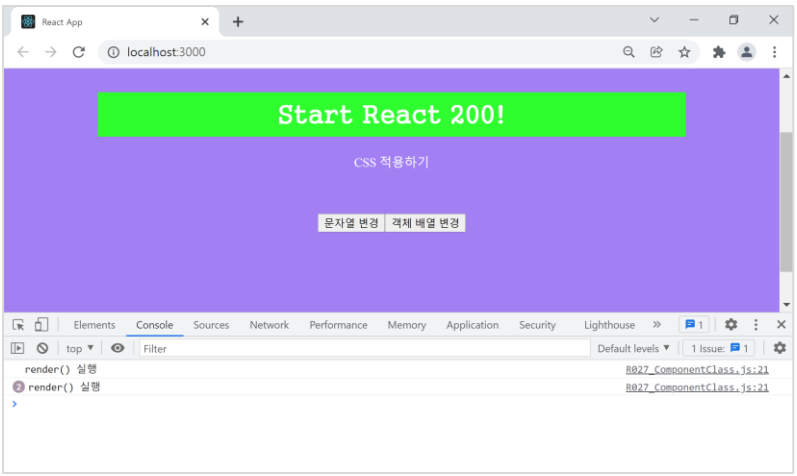
- R027_ComponentClass.js 파일
  - 13: forceUpdate() 함수는 화면을 강제로 새로 고침하기 때문에 render() 함수를 다시 실행시켜 화면에 변경된 state 값을 표시할 수 있다.

[react200/src/027/R027_ComponentClass.js]

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

```
02
03    class R027_ComponentClass extends Component {
04      constructor (props) {
05        super(props);
06        this.state = {
07          StateString: 'react',
08          StateArrayObj: ['react', { react: '200' }]
09        }
10      }
11
12      buttonClick = (type) => {
13        if(type === 'String'){
14          this.setState({ StateString: 'react' });
15        }else{
16          this.setState({ StateArrayObj: ['react', { react: '200' }] });
17        }
18      }
19
20      render() {
21        console.log('render() 실행')
22        return (
23          <div>
24            <button onClick={e => this.buttonClick('String')}>문자열 변경</button>
25            <button onClick={e => this.buttonClick('ArrayObject')}>객체 배열 변경</button>
26          </div>
27        )
28      }
29    }
30
31    export default R027_ComponentClass;
```

■ 실행 결과



028 PureComponent 사용하기(class형 컴포넌트)

■ class형 컴포넌트에는 Component와 PureComponent가 있다. 두 컴포넌트 모두 props와 state의 변경에 따라 render() 함수를 호출하는데, 변경에 대한 기준이 다르다. PureComponent에서는 비교 대상의 값을 비교해 동일하지 않으면 변경이 발생했다고 본다. 불필요한 render() 함수 실행을 줄이면 페이지 성능을 향상시킬 수 있다.

■ App.js 파일

[react200/src/028/App.js]

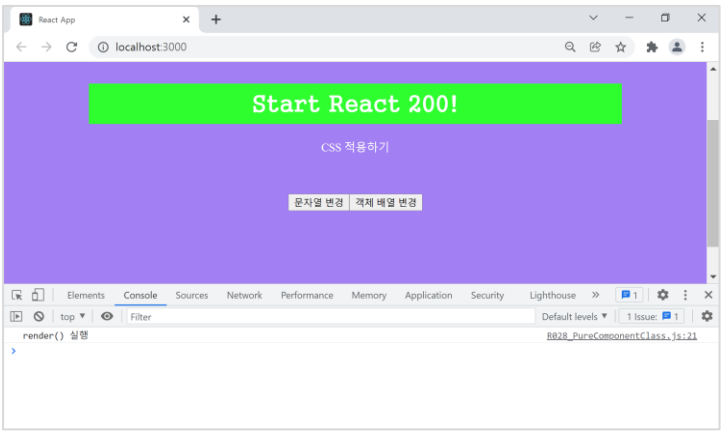
```
01  import React from 'react';
02  import './App.css';
03  import PureComponentClass from './R028_PureComponentClass'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <PureComponentClass/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R028_PureComponentClass.js 파일

[react200/src/028/R028_PureComponentClass.js]

```
01  import React, { PureComponent } from 'react';
02
03  class R028_PureComponentClass extends PureComponent {
04    constructor (props) {
05      super(props);
06      this.state = {
07        StateString: 'react',
08        StateArrayObj: ['react', { react: '200' }]
09      }
10    }
11
12    buttonClick = (type) => {
13      if(type === 'String'){
14        this.setState({ StateString: 'react' });
15      }else{
16        this.setState({ StateArrayObj: ['react', { react: '200' }] });
17      }
18    }
19
20    render() {
21      console.log('render() 실행')
22      return (
23        <div>
24          <button onClick={e => this.buttonClick('String')}>문자열 변경</button>
25          <button onClick={e => this.buttonClick('ArrayObject')}>객체 배열 변경</button>
26        </div>
27      )
28    }
29  }
30
31  export default R028_PureComponentClass;
```

■ 실행 결과



029 shallow-equal 사용하기(class형 컴포넌트)

■ shallow-equal 패키지는 PureComponent에서 state 값의 변경을 비교하는 것과 동일한 기능을 하는 함수를 제공한다. shallowEqualArrays() 함수를 사용하면 문자열과 배열은 값만 비교한다. 객체는 PureComponent와 동일하게 참조 값을 (얕은) 비교한다.

```
// shallow-equal 패키지를 설치한다.
D:\dev\workspace\react\react200-lab>yarn add shallow-equal
...(생략)...
info Direct dependencies
└── shallow-equal@1.2.1
info All dependencies
└── shallow-equal@1.2.1
Done in 16.99s.
```

■ App.js 파일

[react200/src/029/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ShallowEqual from './R029_ShallowEqual'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ShallowEqual/>
11      </div>
12    );
13  }
14
15  export default App;
```

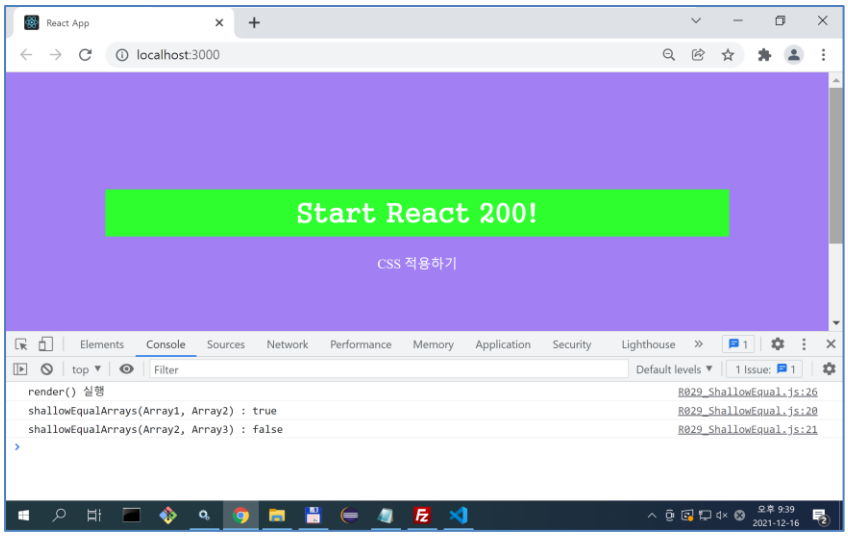
■ R029_ShallowEqual.js 파일

[react200/src/029/R029_ShallowEqual.js]

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

```
02  import { shallowEqualArrays} from "shallow-equal";
03
04  class R029_ShallowEqual extends Component  {
05    constructor (props) {
06      super(props);
07      this.state = { StateString: 'react' }
08    }
09
10    shouldComponentUpdate(nextProps, nextState){
11      return !shallowEqualArrays(this.state, nextState)
12    }
13
14    componentDidMount(){
15      const object = { react : '200'};
16      const Array1 = ['리액트', object];
17      const Array2 = ['리액트', object];
18      const Array3 = ['리액트', { react : '200'}];
19
20      console.log('shallowEqualArrays(Array1, Array2) : ' + shallowEqualArrays(Array1, Array2));
21      console.log('shallowEqualArrays(Array2, Array3) : ' + shallowEqualArrays(Array2, Array3));
22      this.setState({StateString : 'react'})
23    }
24
25    render() {
26      console.log('render() 실행')
27      return (<div></div>)
28    }
29  }
30
31  export default R029_ShallowEqual;
```

■ 실행 결과



030 함수형 컴포넌트 사용하기

■ 함수형 컴포넌트는 클래스형 컴포넌트와 달리 state가 없고 생명주기 함수를 사용할 수 없다. 상위 컴포넌트에서 props와 context를 파라미터로 전달받아 사용하고 render() 함수가 없으므로 return만 사용해 화면을 그려준다.

■ App.js 파일

[react200/src/030/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import FunctionComponent from './R030_FunctionComponent'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <FunctionComponent contents="[ THIS IS FunctionComponent ]"/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R030_FunctionComponent.js 파일

- 04: 상위 컴포넌트에서 전달받은 props를 지역 변수에 할당한다. 클래스형 컴포넌트와 달리 props 앞에 this가 붙지 않는다.

[react200/src/030/R030_FunctionComponent.js]

```
01  import React from 'react';
02
03  function R030_FunctionComponent(props){
04    let { contents } = props;
05    return (
06      <h2>{contents}</h2>
07    )
08  }
09
10  export default R030_FunctionComponent;
```

■ 실행 결과

031 hook 사용하기

■ 함수형 컴포넌트에서 클래스형 컴포넌트와 같이 state와 생명주기 함수와 같은 기능을 사용하기 위해 hook을 이용한다. 대표적인 hook 함수에는 usestate()와 useEffect()가 있다.

■ App.js 파일

[react200/src/031/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ReactHook from './R031_ReactHook'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ReactHook/>
11      </div>
12    );
13  }
14
15  export default App;
```

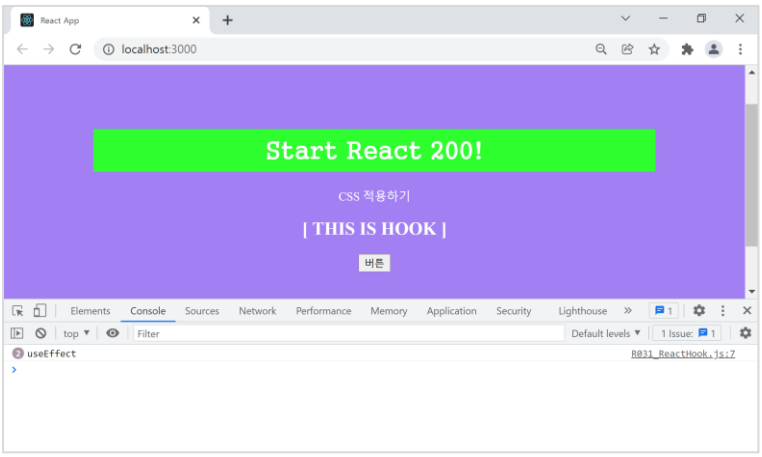
■ R031_ReactHook.js 파일

- 04: useState() 함수로 state 변수값을 선언 및 할당한다. 이때 두 가지 인자가 선언된 배열이 반환된다. 첫 번째 인자 contents는 state 변수명, 두 번째 인자 setContents는 state 변수값을 변경해주는 함수다.
- 06~08: useEffect() 함수는 생명주기 함수 componentDidMount()와 같이 return되는 html 코드들이 화면에 그려진 이후에 실행된다. 최초 페이지가 로딩될 때 한 번 실행되고 setContents() 함수로 state 값이 변경되면 한 번 더 실행된다.
- 13: 버튼을 클릭하면 line 4에서 선언한 setContents() 함수로 contents 값을 수정한다. 이때 state 값이 변경되면 화면을 다시 그려주는데, “ THIS IS REACT ” 라는 글자가 “ THIS IS HOOK ” 로 변경되는 것을 확인할 수 있다.

[react200/src/031/R031_ReactHook.js]

```
01  import React, { useState, useEffect } from 'react';
02
03  function R031_ReactHook(props){
04    const [contents, setContents] = useState('[ THIS IS REACT ]');
05
06    useEffect(() => {
07      console.log('useEffect');
08    });
09
10    return (
11      <div style={{padding: "0px"}}>
12        <h2>{contents}</h2>
13        <button onClick={() =>setContents('[ THIS IS HOOK ]')}>버튼</button>
14      </div>
15    )
16  }
17
18  export default R031_ReactHook;
```

■ 실행 결과



032 Fragments 사용하기

- 컴포넌트 단위로 element를 return할 때 하나의 <html> 태그로 전체를 감싸지 않으면 에러가 발생한다. 이때 <Fragments> 태그로 감싸면 불필요한 <html> 태그를 추가하지 않고 사용할 수 있다.

- App.js 파일

[react200/src/032/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import Fragments from './R032_Fragments'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <Fragments/>
11      </div>
12    );
13  }
14
15  export default App;
```

- R032_Fragments.js 파일
  - 06~09: <React.Fragment> 태그로 사용하지 않았다면 <p> 태그와 태그가 하나의 태그로 감싸져 있지 않기 때문에 에러가 발생한다. <></>는 <React.Fragment> 태그의 약식 표현이다.

[react200/src/032/R032_Fragments.js]

```
01  import React, { Component } from 'react';
02
03  class R032_Fragments extends Component {
04    render() {
05      return (
06        <>
07          <p>P TAG</p>
08          <span>SPAN TAG</span>
```

```
09        </>
10      )
11    }
12  }
13
14  export default R032_Fragments;
```

■ 실행 결과

033 map()으로 element 반환하기

■ 반복해서 출력해야 하는 element들을 배열에 넣어두고 map() 함수로 순서대로 나열해 컴포넌트를 return할 수 있다.

■ App.js 파일

[react200/src/033/App.js]

```
01  import React from 'react';
02  // import './App.css';
03  import ReturnMap from './R033_ReturnMap'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <ReturnMap/>
11      </div>
12    );
13  }
14
15  export default App;
```

■ R033_ReturnMap.js 파일

- 11~13: map() 함수로 element_Array 배열에 있는 element들을 순서대로 꺼내 <ul> 태그 안쪽에 출력한다.

[react200/src/033/R033_ReturnMap.js]

```
01  import React, { Component } from 'react';
02
03  class R033_ReturnMap extends Component {
04    render() {
05      const element_Array = [
```

```
06        <li>react</li>
07        , <li>200</li>
08        , <li>Array map</li>
09      ]
10      return (
11        <ul>
12          {element_Array.map((array_val) => array_val)}
13        </ul>
14      )
15    }
16  }
17
18  export default R033_ReturnMap;
```

■ 실행 결과

034 reactstrap Alerts 사용하기

■ bootstrap은 프론트엔드 디자인을 쉽게 구현할 수 있도록 도와주는 html, css, js 프레임워크다. bootstrap을 reactt에서 사용할 수 있도록 패키지로 만든 것이 reactstrap이다.

```
// reactstrap, bootstrap 패키지를 설치한다.
D:\dev\workspace\react\react200-lab>yarn add reactstrap
D:\dev\workspace\react\react200-lab>yarn add bootstrap
```

■ App.js 파일
- 04: bootstrap.css를 임포트해 <reactstrap> 태그를 사용할 때 boostrap.css가 적용될 수 있도록 한다.

[react200/src/034/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ReactstrapAlerts from './R034_ReactstrapAlerts'
04  import 'bootstrap/dist/css/bootstrap.css'
05
06  function App() {
07    return (
08      <div>
09        <h1>Start React 200!</h1>
10        <p>CSS 적용하기</p>
11        <ReactstrapAlerts/>
12      </div>
13    );
14  }
15
```

```
16  export default App;
```

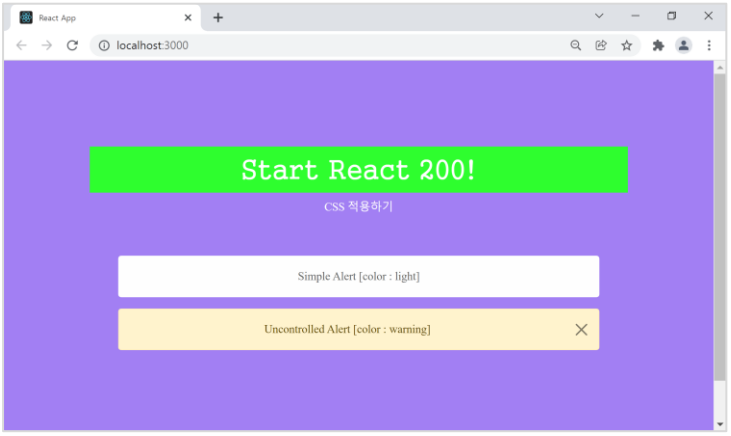
■ R034_ReactstrapAlerts.js 파일

- 02: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. 기본 알림 Alert와 삭제 기능이 추가된 알림 UncontrolledAlert를 사용하기 위해 선언한다.
- 11~13: 삭제 가능한 알림 영역인 UncontrolledAlert를 구현한다. x버튼을 누르면 알림 영역이 삭제된다.

[react200/src/034/R034_ReactstrapAlerts.js]

```
01  import React, { Component } from 'react';
02  import { Alert, UncontrolledAlert } from 'reactstrap';
03
04  class R034_ReactstrapAlerts extends Component {
05    render() {
06      return (
07        <div>
08        <Alert color="light">
09          Simple Alert [color : light]
10        </Alert>
11        <UncontrolledAlert color="warning">
12          Uncontrolled Alert [color : warning]
13        </UncontrolledAlert>
14        </div>
15      )
16    }
17  }
18
19  export default R034_ReactstrapAlerts;
```

■ 실행 결과



035 reactstrap Badge 사용하기

■ Badge 패키지는 부모 요소에 추가로 특정 문자열이나 숫자를 표시할 때 사용된다.

■ App.js 파일

[react200/src/035/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ReactstrapBadges from './R035_ReactstrapBadges'
04  import 'bootstrap/dist/css/bootstrap.css'
05
06  function App() {
07    return (
08      <div>
09        <h1>Start React 200!</h1>
10        <p>CSS 적용하기</p>
11        <ReactstrapBadges/>
12      </div>
13    );
14  }
15
16  export default App;
```

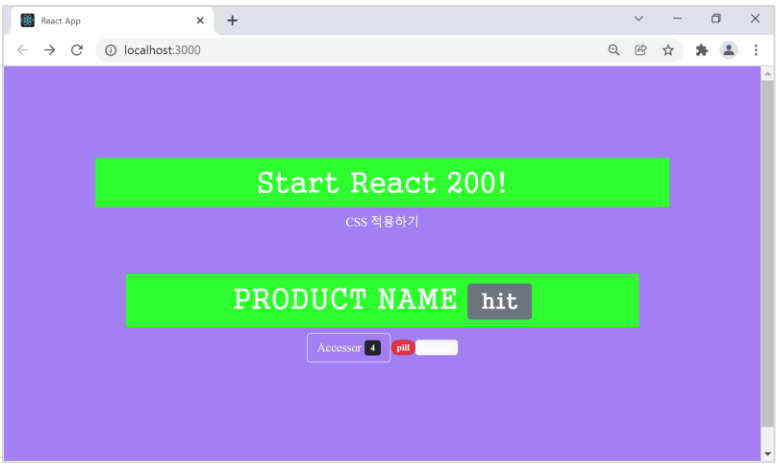
■ R035_ReactstrapBadge.js 파일

- 09~11: 버튼 안에 배지를 추가했다. 접속자, 메시지의 수와 같은 어떤 속성의 수치를 나타낼 때 사용할 수 있다.
- 12: pill 속성을 추가하면 테두리가 둥글게 적용할 수 있다.
- 13: href 속성을 추가해 배지에 링크를 연결할 수 있다.

[react200/src/035/R035_ReactstrapBadge.js]

```
01  import React, { Component } from 'react';
02  import { Badge, Button } from 'reactstrap';
03
04  class R035_ReactstrapBadges extends Component {
05    render() {
06      return (
07        <div>
08          <h1>PRODUCT NAME <Badge color="secondary">hit</Badge></h1>
09          <Button color="light" outline>
10            Accessor <Badge color="dark">4</Badge>
11          </Button>
12          <Badge color="danger" pill>pill</Badge>
13          <Badge href="http://example.com" color="light">GoLink</Badge>
14        </div>
15      )
16    }
17  }
18
19  export default R035_ReactstrapBadges;
```

■ 실행 결과



036 reactstrap Breadcrumbs 사용하기

- Breadcrumbs 패키지는 페이지 위치 경로를 지정한 웹 내비게이션에 사용된다. 보통 웹 사이트 상단에 표시되는 메뉴 리스트에 사용하며 특정 메뉴를 선택하면 해당하는 페이지 위치로 이동시킨다.

- App.js 파일

[react200/src/036/App.js]

```
01  import React from 'react';
02  import './App.css';
03  import ReactstrapBreadcrumbs from './R036_ReactstrapBreadcrumbs'
04  import 'bootstrap/dist/css/bootstrap.css'
05
06  function App() {
07    return (
08      <div>
09        <h1>Start React 200!</h1>
10        <p>CSS 적용하기</p>
11        <ReactstrapBreadcrumbs/>
12      </div>
13    );
14  }
15
16  export default App;
```

- R036_ReactstrapBreadcrumbs.js 파일
  - 02: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Breadcrumb와 BreadcrumbItem을 사용하기 위해 선언한다.

[react200/src/036/R036_ReactstrapBreadcrumbs.js]

```
01  import React, { Component } from 'react';
02  import { Breadcrumb, BreadcrumbItem } from 'reactstrap';
03
04  class R036_ReactstrapBreadcrumbs extends Component {
05    render() {
06      return (
07        <div id="top">
08          <Breadcrumb tag="nav" listTag="div">
```

```
09            <BreadcrumbItem tag="a" href="#top">Go_top</BreadcrumbItem>
10            <BreadcrumbItem tag="a" href="#bottom">Go_bottom</BreadcrumbItem>
11          </Breadcrumb>
12          <div id="bottom" style={{marginTop:"1000px"}}>
13            <span>bottom</span>
14          </div>
15        </div>
16        )
17    }
18  }
19
20  export default R036_ReactstrapBreadcrumbs;
```

■ 실행 결과