03장 리액트 컴포넌트

■ 학습 목표
- 이 장에서는 리액트의 핵심이라고 할 수 있는 컴포넌트를 공부한다. 이 장을 공부하고 나면 원하는 화면에 맞는 컴포넌트를 구성할 수 있고, 사용자의 입력에 따라 화면을 동적으로 변경하는 방법도 알 수 있다.

3.1 컴포넌트를 표현하는 JSX

■ JSX는 JavaScript XML의 줄임말로 '자바스크립트에 XML을 추가한 확장형 문법'으로 이해하면 된다.

3.1.1 JSX 사용해 보기

■ 01장에서 create-react-app으로 do-it-example이라는 리액트 앱을 만들었다. ./src/App.js를 열어 기존 내용을 지우고 아래의 내용을 작성한다.
- 06~12: return()함수 내에 HTML을 사용했다는 점을 알 수 있다. 특히 img 엘리먼트 끝에 마침 표시 />가 있다는데 주목하자. 만약 엘리멘트의 시작 표시 <와 마침표시 />의 짝이 맞지 않으면 리액트 엔진이 JSX를 분석할 때 오류가 발생한다.

[./src/App.js]

```
01  // src폴더안에 App.js을 연 다음 기존 내용을 모두 지우고 아래의 내용으로 작성해 보세요
02  import React from 'react';
03
04  class App extends React.Component {
05    render() {
06      return (
07        // 아래의 내용이 JSX 양식 입니다.
08        <div>
09          <img src="http://www.easyspub.co.kr/images/logo_footer.png" />
10          <div>안녕하세요</div>
11        </div>
12      );
13    }
14  }
15  export default App;
```

```
# 리액트 앱 구동하기
D:\dev\workspace\react.vs\do-it-example>yarn start
```

3.1.2 JSX와 기본 개발 방법의 차이점 알아보기

■ App 컴포넌트를 사용하지 않고 화면 구성해 보기
- 09~19: 만약 App 컴포넌트를 사용하지 않는다면 index.js 파일을 아래와 같이 수정해야 한다.

[./src/index.js] ./src/03/non-jsx-sample.js 파일을 이용

```
01  /*
02  import React from 'react';
03  import ReactDOM from 'react-dom';
04  import App from './App';
05
06  ReactDOM.render(<App />, document.getElementById('root'));
07  */
08
09  // src폴더안에 index.js을 연 다음 기존 내용을 모두 지우고 아래의 내용으로 작성해 보세요
10  var img = document.createElement('img');
11  img.setAttribute('src', 'http://www.easyspub.co.kr/images/logo_footer.png');
12  var divEl = document.createElement('div');
13  divEl.innerText = '안녕하세요';
14  var welcomeEl = document.createElement('div');
15  welcomeEl.append(img);
16  welcomeEl.append(divEl);
17
18  var root = document.getElementById('root');
19  root.append(welcomeEl);
```

3.2 컴포넌트와 구성 요소

3.2.1 컴포넌트의 개념

- 웹 사이트의 화면은 각 요소가 비슷하고 반복적으로 사용한 경우가 많다. 이점을 착안하여 컴포넌트가 등장하게 되었다.

- 컴포넌트는 MVC의 뷰를 독립적으로 구성하여 재사용도 할 수 있고 컴포넌트를 통해 새로운 컴포넌트를 쉽게 만들 수도 있다.

3.2.2 간단한 컴포넌트 추가하고 화면으로 띄워보기

- src 폴더에 03이라는 폴더를 만들고, TodaysPlan.jsx 파일을 생성한다. rcc를 입력한 다음 Tab을 누르면 파일 이름에 맞게 클래스형 컴포넌트의 뼈대를 만들어 준다.

[./src/03/TodaysPlan.jsx]

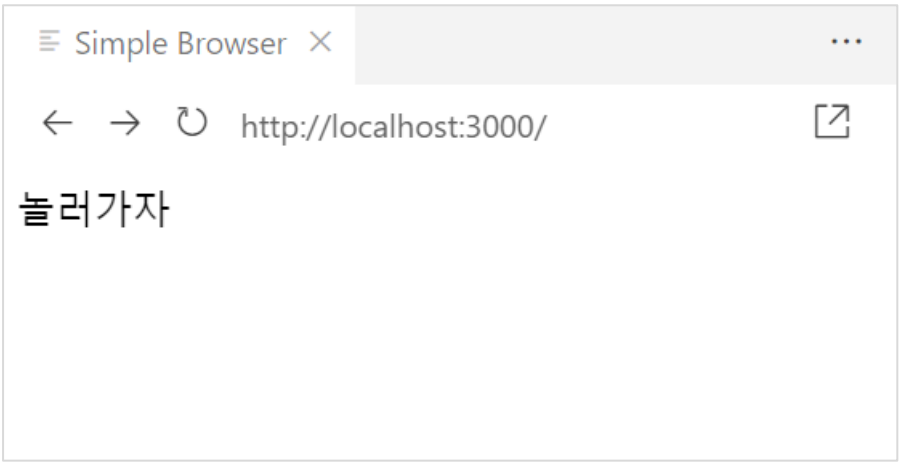
```
01  import React from 'react';
02
03  class TodaysPlan extends React.Component {
04    render() {
05      return (
06       <div className="message-container">
07         놀러가자
08       </div>
09      );
10    }
11  }
12
13  export default TodaysPlan;
```

- App 컴포넌트에 TodaysPlan 컴포넌트 추가하기
  - 02: 새로 작성한 컴포넌트를 App.js에 임포트한다.
  - 08: 컴포넌트를 JSX 안에 마크업 형식으로 추가한다.

[./App.js]

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

■ TodaysPlan 컴포넌트 출력하기



3.2.3 컴포넌트 구성 요소

■ 프로퍼티와 state는 부모와 자식 컴포넌트가 연결된 상태에서 공유하는 데이터이다. 반면 컨텍스트는 부모와 자식 컴포넌트가 연결되어 있지 않아도 데이터를 공유할 수 있게 해준다.

데이터 구성 요소	특징
프로퍼티(property)	상위 컴포넌트에서 하위 컴포넌트로 전달되는 읽기 전용 데이터이다.
상태(state)	컴포넌트의 상태를 저장하고 변경할 수 있는 데이터이다.
컨텍스트(context)	부모 컴포넌트에서 생성하여 모든 자식 컴포넌트에 전달하는 데이터이다.

3.3 컴포넌트에 데이터를 전달하는 프로퍼티

3.3.1 프로퍼티의 기초 알아보기

■ 다음은 App 컴포넌트에서 프로퍼티를 MyComponent 컴포넌트에 전달한 예제이다.
- 프로퍼티가 상위 컴포넌트에서 하위 컴포넌트로 전달된다. 이것을 '단방향으로 데이터가 흐른다'라고 표현한다.

```
// App 컴포넌트
class App extends React.Component {
  render() {
    return (
      <div className="body">
        <MyComponent name="message" />
      </div>
    );
  }
}
...

// MyComponent 컴포넌트
class MyComponent extends React.Component {
  render() {
    const name = this.props.name;
    return <span>{name}</span>;
  }
}
```

3.3.2 프로퍼티의 다양한 사용 방법 알아보기

(1) 문자열형 프로퍼티 사용하기

■ 프로퍼티의 자료형을 선언하는 방법은 리액트에서 제공하는 prop-types를 이용하면 된다.

- 02: pro-types 라이브러리를 ProTypes라는 이름으로 임포트한다.
- 08: name 프로퍼티로 받은 문자열을 출력한다.
- 15~16: 프로퍼티의 자료형을 선언한다.

[./src/03/PropsComponent.jsx]

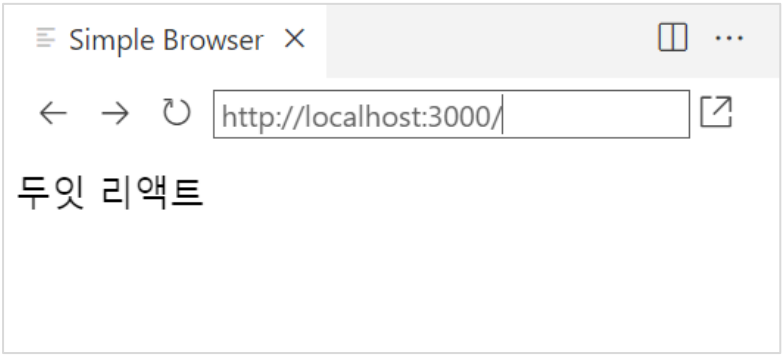
```
01  import React from 'react';
02  import PropTypes from 'prop-types';
03
04  class PropsComponent extends React.Component {
05    render() {
06      return (
07       <div className="message-container">
08         {this.props.name}
09       </div>
10      );
11    }
12  }
13
14  // 자료형을 선언하는 예제
15  PropsComponent.propTypes = {
16    name: PropTypes.string,
17  };
18
19  export default PropsComponent;
```

■ App 컴포넌트의 내용을 모두 지우고 다음을 입력한다. 여기부터는 App.js를 App.jsx로 저장하여 사용한다.

[./src/App.jsx]

```
01  import React from 'react';
02  import PropComponent from './03/PropsComponent';
03
04  class App extends React.Component {
05    render() {
06      return (
07        <PropComponent
08          name="두잇 리액트"
09        />
10      );
11    }
12  }
13
14  export default App;
```

■ 실행 결과



(2) 다양한 프로퍼티 사용하기
(3) 불리언 프로퍼티 사용하기
(4) 객체형 프로퍼티 사용하기
(5) 필수 프로퍼티 사용하기
(6) 프로퍼티 기본값 지정하기
(7) 자식 프로퍼티 사용하기

3.4 컴포넌트 상태 관리하기

3.4.1 state로 상태 관리하기

■ state는 ‘ 값을 저장하거나 변경할 수 있는 객체 ’ 로 보통 버튼을 클릭하거나 값을 입력하는 등의 이벤트와 함께 사용된다.

■ 03 폴더에 StateExample 컴포넌트를 만들어 다음 코드를 입력한다.
- 07~10: 컴포넌트에서 관리하려는 변수 state 초깃값을 this.state에 객체 형태로 정의한다.
- 12: 함수로 넘어갈 this는 반드시 생성자에서 bind()함수로 묶어주어야 한다.
- 14: setTimeout() 함수를 사용하여 4초 후에 handleData() 함수를 호출한다.
- 18: 컴포넌트 특수 변수 this.state를 사용하여 state 값에 접근한다.
- 20~23: 컴포넌트의 내장 함수 this.setState()를 사용하여 state값을 변경한다.

[./src/03/StateExample.jsx]

```
01  import React from 'react';
02
03  class StateExample extends React.Component {
04    constructor(props) {
05      super(props);
06      // 상태 정의
07      this.state = {
08        loading: true,
09        formData: 'no data',
10      };
11      // 이후 콜백 함수를 다룰때 bind를 선언하는 부분에 대해 다룹니다
12      this.handleData = this.handleData.bind(this);
13      // 생성 후 4초 후에 handleData를 호출합니다.
14      setTimeout(this.handleData, 4000);
15    }
16    handleData() {
17      const data = 'new data';
18      const { formData } = this.state;
19      // 상태 변경
20      this.setState({
21        loading: false,
22        formData: data + formData,
23      });
24      // this.state.loading 은 현재 true 입니다.
25      // 이후 호출될 출력함수에서의 this.state.loading은 false입니다.
26    }
27    // 다음과 같이 setState함수를 사용할 수 있습니다.
28    // handleData(data) {
29    //   this.setState(function(prevState) {
30    //     const newState = {
31    //       loading : false,
32    //       formData: data + prevState.formData,
33    //     };
34    //     return newState;
35    //   });
36    // }
37    render() {
38      return (
39        <div>
40          {/* 상태 데이터는 this.state로 접근 가능합니다. */}
41          <span>로딩중: {String(this.state.loading)}</span>
42          <span>결과: {this.state.formData}</span>
43        </div>
44      );
45    }
46  }
47
48  export default StateExample;
```

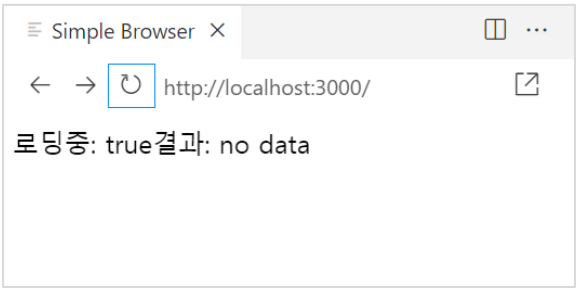
■ App 컴포넌트에 앞에서 만든 컴포넌트를 넣어 화면에 그려본다.

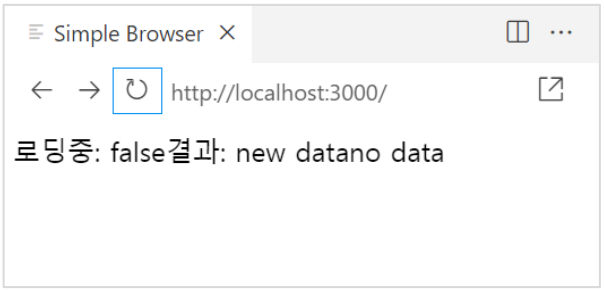
[./src/App.jsx]

```
01  import React from 'react';
02  import StateExample from './03/StateExample';
03
04  class App extends React.Component {
05    render() {
06      return (
07        <div>
08          <div>
```

```
09            <StateExample />
10          </div>
11        </div>
12      );
13    }
14  }
15
16  export default App;
```

■ 실행 결과
- 4초 뒤에 state에 저장된 값이 바뀌며 화면이 바뀐다.



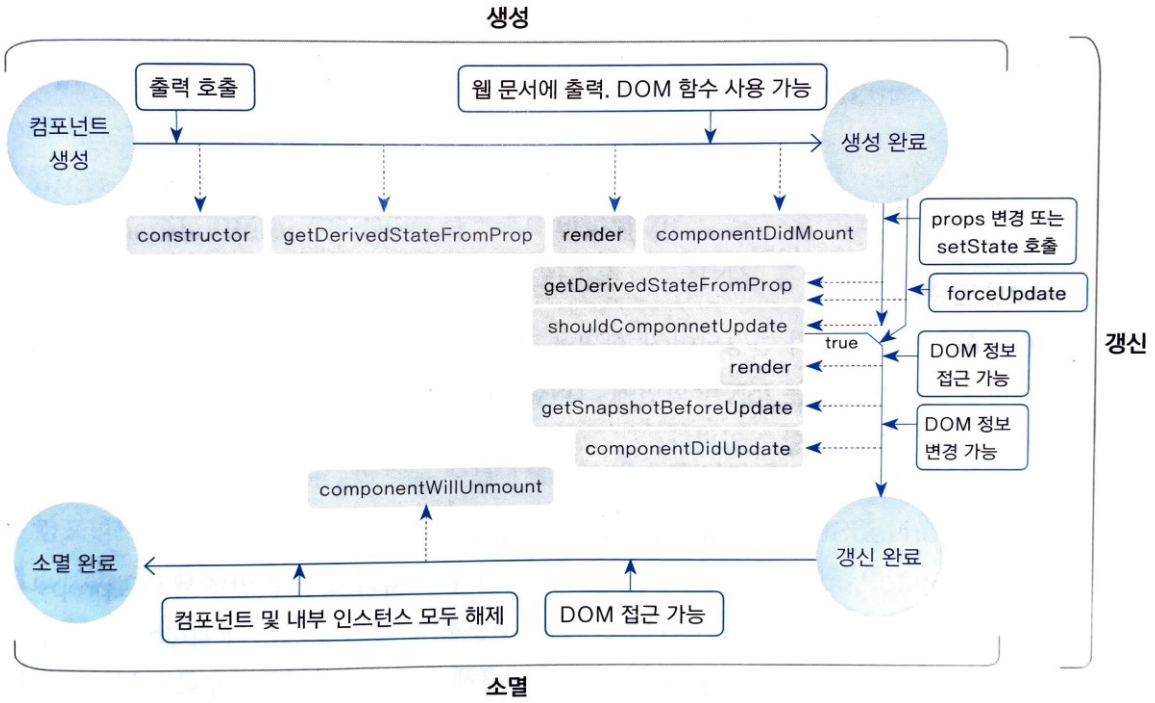


3.4.2 클래스 인스턴스 변수와 forceUpdate() 함수로 state 관리하기

3.5 컴포넌트의 생명주기

3.5.1 생명주기 함수 살펴보기

■ 생명주기 함수는 지금까지 실습에서 사용한 render() 함수를 포함하여 총 8종의 함수가 있다. 생명주기 함수는 리액트 엔진에서 자동으로 호출한다.
- 생성 과정: constructor(), getDerivedStateFromProp, render, componentDidMount
- 갱신 과정: getDerivedStateFromProp, shouldComponentUpdate, render, getSnapshotBeforeUpdate, componentDidUpdate
- 소멸 과정: componentWillUnmount



- cnstructor(props) 함수
  - 이름 그대로 '맨 처음에 생성될 때 한 번만 호출'되며, 상태(state 또는 객체 변수)를 선언할 때 사용된다. constructor() 함수를 정의할 때는 항상 super() 함수를 가장 위에 호출해야 한다.

```
...
constructor(props) {
  super(props);
  // 이후에 추가적인 state 데이터 혹은 변수를 선언한다.
}
...
```

- render() 함수
  - 데이터가 변경되어 새 화면을 그려야 할 때 자동으로 호출되는 함수이다.

3.5.2 생명주기 함수의 실행 과정 살펴보기
3.5.3 카운터 프로그램 만들며 생명주기 함수 사용해 보기

3.6 클래스형 컴포넌트

3.6.1 Component 알아보기

- Component 클래스는 프로퍼티, state와 생명주기 함수가 들어있는 구조의 컴포넌트를 만들 때 사용한다.

3.6.2 PureComponent 알아보기

- PureComponent 클래스는 Component 클래스를 상속받은 클래스이다. PureComponent 클래스로 만들어진 컴포넌트는 '얕은 비교를 통해 데이터가 변경된 경우'에만 render() 함수를 호출한다. 반면 Component 클래스로 만들어진 컴포넌트는 항상 render() 함수를 호출한다.

3.7 함수형 컴포넌트

- 함수형 컴포넌트는 조금 길게 표현하여 state가 없는 함수형 컴포넌트(SFC, Stateless Functional Component)라고 부른다.

- 다음은 함수형 컴포넌트를 사용한 예제이다.

[src/03/SFCjsx]

```
01  import React from 'react';
02  import PropTypes from 'prop-types';
03
04  function SFC(props, context) {
05    // 클래스 컴포넌트의 this.props값과 동일합니다.
06    const { somePropValue } = props;
07    // 클래스 컴포넌트의 this.context와 동일합니다.
08    // context는 차후에 자세히 다룰 예정입니다.
09    const { someContextValue } = context;
10    return <h1>Hello, {somePropValue}</h1>;
11  }
12
13  SFC.propTypes = { somePropValue: PropTypes.any };
14  SFC.defaultProps = { somePropValue: 'default value' };
15
16  export default SFC;
```

[./src/Appjsx]

```
01  import React from 'react';
02  import PropTypes from 'prop-types';
03
04  function SFC(props, context) {
05    // 클래스 컴포넌트의 this.props값과 동일합니다.
06    const { somePropValue } = props;
07    // 클래스 컴포넌트의 this.context와 동일합니다.
08    // context는 차후에 자세히 다룰 예정입니다.
09    const { someContextValue } = context;
10    return <h1>Hello, {somePropValue}</h1>;
11  }
12
13  SFC.propTypes = { somePropValue: PropTypes.any };
14  SFC.defaultProps = { somePropValue: 'default value' };
15
16  export default SFC;
```

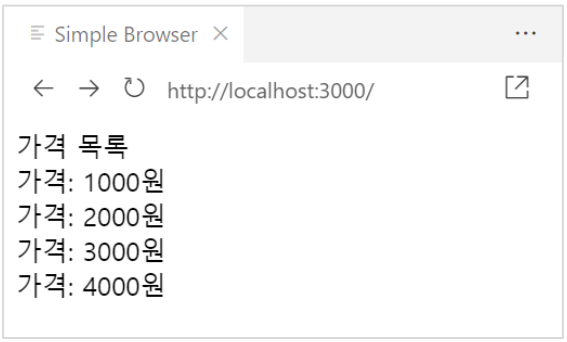
3.8 배열 컴포넌트

3.8.1 배열 컴포넌트를 위한 map() 함수 사용 방법

■ map() 함수를 활용한 가격표 목록을 출력한 예제이다.

[./src/03/ListExample.jsx]

```
01  import React from 'react';
02
03  class ListExample extends React.PureComponent {
04    render() {
05      const priceList = [1000, 2000, 3000, 4000];
06      const prices = priceList.map((price) => <div>가격: {price}원</div>);
07      return (
08        <div>
09          <label>가격 목록</label>
10          {prices}
11        </div>
12      );
13    }
14  }
15
16  export default ListExample;
```



3.8.2 map() 함수 사용하여 배열 컴포넌트 출력하기

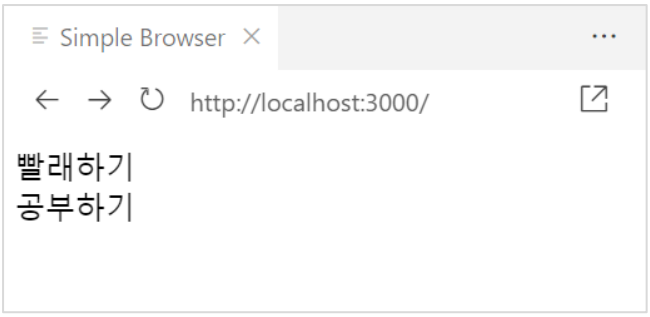
■ 배열 컴포넌트의 활용을 위한 예제를 작성해 보자.

[./src/03/Todolist.jsx]

```
01  import React from 'react';
02
03  class TodoList extends React.PureComponent {
04    render() {
05      const todoList = [
06        { taskName: '빨래하기', finished: false },
07        { taskName: '공부하기', finished: true },
08      ];
09      return (
10        <div>
11          {todoList.map(todo => (
12            <div key={todo.taskName}>{todo.taskName}</div>
13          ))}
14        </div>
```

```
      );
    }
  }

export default TodoList;
```



3.8.3 render() 함수에서 여러 개의 JSX 노드 반환하기

■ render() 함수는 트리 구조의 노드를 반환해야 한다.

```
// 잘못 사용된 예
render() {
  return (
    <input type="radio" name="option1" value="1" label="1개" />
    <input type="radio" name="option1" value="2" label="2개" />
    <input type="radio" name="option1" value="3" label="3개" />
  );
}

// 리액트 16.3 버전까지 render() 함수는 트리 구조의 노드 1개만 반환할 수 있다.
// 만약 여러 개의 노드를 반환하고 싶은 경우, 의미 없는 최상위 노드를 추가해야 한다.
render() {
  return (
    <div>
      <input type="radio" name="option1" value="1" label="1개" />
      <input type="radio" name="option1" value="2" label="2개" />
      <input type="radio" name="option1" value="3" label="3개" />
    </div>
  );
}

// 리액트 16.3 버전 이후에 React.Fragment 컴포넌트가 추가되었다.
render() {
  return (
    <React.Fragment>
      <input type="radio" name="option1" value="1" label="1개" />
      <input type="radio" name="option1" value="2" label="2개" />
      <input type="radio" name="option1" value="3" label="3개" />
    </React.Fragment>
  );
}

// React.Fragment라 표현은 다음과 같이 사용해도 된다.
render() {
  return (
    <>
      <input type="radio" name="option1" value="1" label="1개" />
```

```
      <input type="radio" name="option1" value="2" label="2개" />
      <input type="radio" name="option1" value="3" label="3개" />
    </>
  );
}

// map() 함수를 사용하면 감싸는 것조차 생략할 수도 있다.
render() {
  return [1, 2, 3].map((num) => (
    <input type="radio" name="option1" key={`${num}`} value=${num} label={`${num}개`} />
  ));
}
```

3.9 컴포넌트에서 콜백 함수와 이벤트 처리하기

- 앞에서 프로퍼티를 사용하면 상위 컴포넌트의 데이터를 하위 컴포넌트에 전달할 수 있다는 것을 배웠다. 만약 하위 컴포넌트에서 프로퍼티를 변경해야 할 때는 어떻게 해야 할까? 프로퍼티 원본을 수정할 수 있는 함수를 하위 컴포넌트에 제공하면 된다.

- 콜백 함수란 정의된 위치에서 실행되지 않고, 이후 특정 상황(이벤트, 다른 함수 호출 등)에서 실행되는 함수를 말한다. 즉, 콜백함수를 프로퍼티로 전달하면 된다.

3.9.1 콜백 함수로 프로퍼티 수정해 보기

- index.js

[src/index.js]

```
01  import React from 'react';
02  import ReactDOM from 'react-dom';
03  import App from './CounterApp';
04
05  ReactDOM.render(<App />, document.getElementById('root'));
```

- CounterApp.jsx
  - 13~15: App 컴포넌트에 count값을 증가시킬 목적으로 추가한다.
  - 18: Counter 컴포넌트를 출력한다. 그리고 App 컴포넌트에서 Counter 컴포넌트의 프로퍼티로 onAdd를 추가한다.

[src/CounterApp.jsx]

```
01  import React from 'react';
02  import Counter from './03/Counter2';
03
04  class App extends React.Component {
05    constructor(props) {
06      super(props);
07      this.state = {
08        count: 1,
09      };
10    }
11    increateCount() {
12      this.setState(({ count }) => ({ count: count + 1}));
13    }
```

```
14    render() {
15      return (
16        <Counter count={this.state.count} onAdd={this.increateCount} />
17      );
18    }
19  }
20
21  export default App;
```

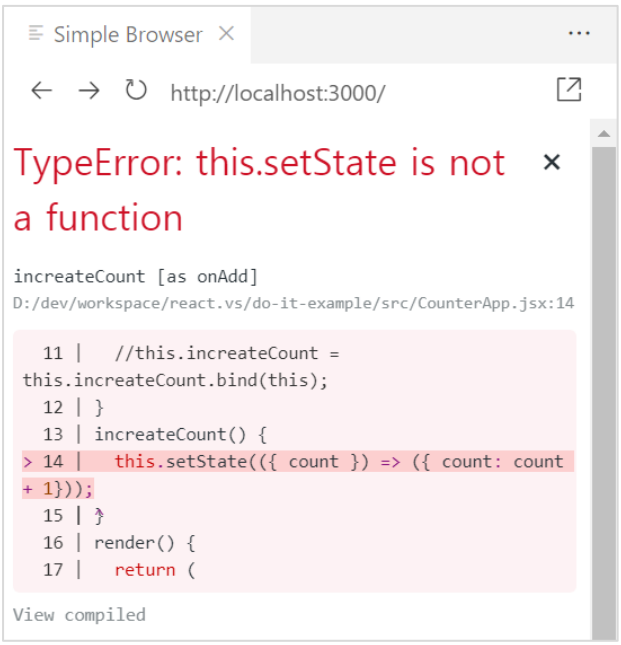
■ Counter2.jsx

[./src/03/Counter2.jsx]

```
01  import React from 'react';
02  import PropTypes from 'prop-types';
03
04  class Counter2 extends React.Component {
05    render() {
06      return (
07        <div>
08          현재 카운트: {this.props.count}
09          <button
10            onClick={() => this.props.onAdd()}
11          >
12            카운트 증가
13          </button>
14        </div>
15      );
16    }
17  }
18
19  Counter2.propTypes = {
20    count: PropTypes.number,
21    onAdd: PropTypes.func,
22  };
23
24  export default Counter2;
```

■ 실행 결과

- 코드를 작성한 다음 [카운트 증가] 버튼을 눌러도 제대로 작동하지 않는다.
- 아래 오류 메시지는 onAdd()함수에 구현되어 있는 this.setState(...)는 상위 컴포넌트에 정의되어 있는데 하위 컴포넌트에서 실행되기 때문에 발행한 것이다.



3.9.2 bind() 함수로 this 범위 오류 해결하기

- CounterApp.jsx
  - 11: this 범위 오류는 bind()함수를 사용하면 문제를 해결할 수 있다.

[src/CounterApp.jsx]

```
01  import React from 'react';
02  import Counter from './03/Counter2';
03
04  class App extends React.Component {
05    constructor(props) {
06      super(props);
07      this.state = {
08        count: 1,
09      };
10      // this 오류를 확인한 후에 아래 주석을 삭제해 주세요.
11      this.increateCount = this.increateCount.bind(this);
12    }
13    increateCount() {
14      this.setState(({ count }) => ({ count: count + 1}));
15    }
16    render() {
17      return (
18        <Counter count={this.state.count} onAdd={this.increateCount} />
19      );
20    }
21  }
22
23  export default App;
```

3.9.3 컴포넌트에서 DOM 객체 함수 사용하기

3.9.4 컴포넌트에서 DOM 이벤트 사용하기

■ CounterApp.jsx

[src/CounterApp.jsx]

```
01  import React from 'react';
02  import Counter from './03/Counter3';
03
04  class App extends React.Component {
05    constructor(props) {
06      super(props);
07      this.state = {
08        count: 1,
09      };
10      // this 오류를 확인한 후에 아래 주석을 삭제해 주세요.
11      this.increateCount = this.increateCount.bind(this);
12    }
13    increateCount() {
14      this.setState(({ count }) => ({ count: count + 1}));
15    }
16    render() {
17      return (
18        <Counter count={this.state.count} onAdd={this.increateCount} />
19      );
20    }
21  }
22
23  export default App;
```

■ Counter3.jsx
- 23: 버튼이 클릭될 때 카운드를 증가하는 함수를 호출한다.
- 24: 마우스가 버튼 밖으로 이동하면 카운트를 초기화하는 함수를 호출한다.

[src/03/Counter3.jsx]

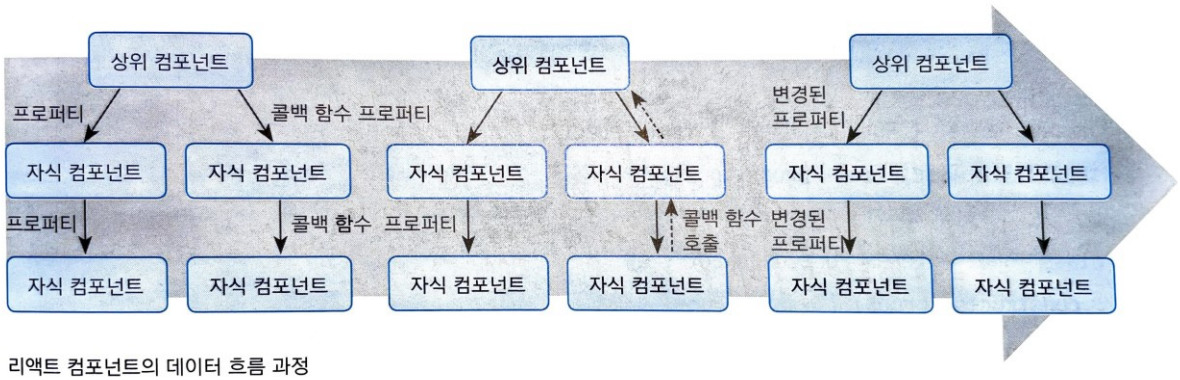
```
01  import React from 'react';
02
03  class Counter3 extends React.Component {
04    constructor(props) {
05      super(props);
06      this.state = {
07        count: 0,
08      };
09      this.increateCount = this.increateCount.bind(this);
10      this.resetCount = this.resetCount.bind(this);
11    }
12    increateCount() {
13      this.setState(({ count }) => ({ count: count + 1}));
14    }
15    resetCount() {
16      this.setState({ count: 0 });
17    }
18    render() {
19      return (
20        <div>
21          현재 카운트: {this.state.count}
22          <button
23            onClick={this.increateCount}
24            onMouseOut={this.resetCount}
25          >
```

```
26            카운트 증가
27          </button>
28          <p>버튼 밖으로 커서가 움직이면 0으로 초기화 됩니다.</p>
29        </div>
30      );
31    }
32  }
33
34  export default Counter3;
```

3.9.5 단방향 흐름 방식 개념 정리

- 리액트는 프로퍼티, state와 같은 데이터를 상위 컴포넌트에서 하위 컴포넌트 방향으로 전달한다. 만약 데이터 변경이 필요한 경우 콜백 함수를 호출하여 원본 데이터가 위치한 상위 컴포넌트에서 데이터를 변경하고 다시 자식 컴포넌트로 전달하도록 만든다.

- 이와 같은 데이터 흐름을 사용하는 기법을 ‘단방향 흐름방식’이라고 한다. 단방향 흐름 방식은 원본 데이터의 무결성을 지켜주므로 데이터 수정으로 인한 데이터 파편화를 줄여준다.



리액트 컴포넌트의 데이터 흐름 과정