

▶ Chapter 10 리액트 라이브러리 맛보기

혼자 공부하는 자바스크립트



○○대학교 ○○학과
홍길동

시작하기 전에

예제 다운로드 및 동영상 강의

https://hanbit.co.kr/store/books/look.php?p_code=B8393055290

저자 : 윤인성

출근하는 게 싫어서 책을 집필하기 시작했다. 현재 직업 특성상 집에서 나갈 이유가 별로 없다는 것에 굉장히 만족하는 성격이기도 하다. 홍차와 커피를 좋아하며 기타, 가야금, 그림 그리기, 스컬핑 등이 취미이다. 저서로는 『혼자 공부하는 파이썬』, 『IT CookBook, HTML5 웹 프로그래밍』, 『모던 웹을 위한 JavaScript+jQuery 입문』, 『모던 웹을 위한 Node.js 프로그래밍』, 『모던 웹 디자인을 위한 HTML5+CSS3 입문』 등이 있으며, 역서로는 『TopCoder 알고리즘 트레이닝』, 『자바 퍼즐러』, 『소셜 코딩으로 이끄는 GitHub 실천 기술』, 『Nature of Code』 등이 있다.

이 책의 학습 목표

▪ CHAPTER 01: 자바스크립트 개요와 개발환경 설정

- 자바스크립트 개발환경 설치와 자바스크립트 프로그래밍 기본 용어 학습

▪ CHAPTER 02: 자료와 변수

- 프로그램 개발의 첫걸음. 자료형과 변수 학습

▪ CHAPTER 03: 조건문

- 프로그램의 흐름을 변화시키는 요소. 조건문의 종류를 알아보고 사용 방법을 이해

▪ CHAPTER 04: 반복문

- 배열의 개념과 문법을 익혀 while 반복문과 for 반복문 학습

▪ CHAPTER 05: 함수

- 다양한 형태의 함수를 만들기과 매개변수를 다루는 방법 이해

▪ CHAPTER 06: 객체

- 객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습

▪ CHAPTER 07: 문서 객체 모델

- DOMContentLoaded 이벤트를 사용한 문서 객체 조작과 다양한 이벤트의 사용 방법 이해

▪ CHAPTER 08: 예외 처리

- 구문 오류와 예외를 구분하고, 예외 처리의 필요성과 예외를 강제로 발생시키는 방법을 이해

▪ CHAPTER 09: 클래스

- 객체 지향을 이해하고 클래스의 개념과 문법 학습

▪ CHAPTER 10: 리액트 라이브러리

- 리액트 라이브러리 사용 방법과 간단한 애플리케이션을 만드는 방법 학습

- CHAPTER 10: 리액트 라이브러리 맛보기

SECTION 10-1 리액트의 기본

SECTION 10-2 리액트와 데이터



CHAPTER 10 리액트 라이브러리 맛보기

리액트 라이브러리 사용 방법과 간단한 애플리케이션을 만드는 방법 학습

SECTION 10-1 리액트의 기본(1)

◦ 리액트 라이브러리(React Library)

- 규모가 큰 자바스크립트 라이브러리로, 사용자 인터페이스(UI)를 쉽게 구성하는데 도움
- 대규모 프론트엔드 웹 애플리케이션을 체계적으로 개발할 수 있으며, 리액트 네이티브를 활용해서 스마트폰에서도 빠른 속도로 작동하는 애플리케이션을 만들 수 있음
- 리액트 라이브러리의 공식 사이트 링크(한국어 버전을 제공하고 번역도 잘 되어 있는 편)
 - <https://ko.reactjs.org/>

Create React App에는 **Babel** 및 **webpack**을 사용하는 프론트 엔드 빌드 파이프라인이 포함되어 있지만 백 엔드 논리 또는 데이터베이스는 처리되지 않습니다. Node.js 백 엔드를 사용하는 React로 서버 렌더링 웹사이트를 빌드하는 경우 단일 페이지 앱에 더욱 적합한 이 **creat-react-app**을 설치하는 대신 **Next.js를 설치**하는 것이 좋습니다. 정적 콘텐츠 지향 웹사이트를 빌드하려면 **Gatsby를 설치**하는 것이 좋을 수도 있습니다.

◦ 리액트 라이브러리 사용 준비하기

- 가장 기본적인 방법은 HTML 파일에서 다음과 같은 3개의 자바스크립트를 읽어 들이는 것
 - <https://unpkg.com/react@17/umd/react.development.js>
 - <https://unpkg.com/react-dom@17/umd/react-dom.development.js>
 - <https://unpkg.com/babel-standalone@6/babel.min.js>




SECTION 10-1 리액트의 기본(2)

◦ 리액트 라이브러리 사용 준비하기

▪ 리액트 기본 사용 준비 (소스 코드 10-1.html)

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04 <title>Document</title>
05 <!-- 리액트 사용 준비-->
06 <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
07 <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
08 <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
09 </head>
10 <body>
11 <div id="root"></div> div#root
12 <!-- 리액트를 사용하는 코드 입력 -->
13 <script type="text/babel">
14
15 </script>
16 </body>
17 </html>
```

문제 출력 디버그 콘솔 터미널

필터(예: text, !exclude)   

Download the React DevTools for a better development experience: <https://reactjs.org/link/react-devtools>

You might need to use a local HTTP server (instead of file://): <https://reactjs.org/link/react-devtools-faq>

You are using the in-browser Babel transformer. Be sure to precompile your scripts for production - <https://babeljs.io/docs/setup/>

태그를 만들

type 속성에 "text/babel"을 지정

▪ 바벨(babel) 라이브러리 및 적용 부분 지정

```
<script type="text/babel"></script>
```

SECTION 10-1 리액트의 기본(3)

루트 컴포넌트 출력하기

- 컴포넌트 생성하기

```
<컴포넌트 이름></컴포넌트 f이름>
```

- 컴포넌트 출력하기

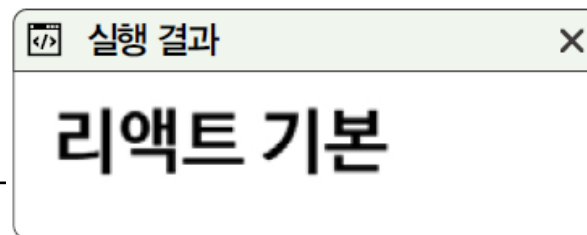
```
ReactDOM.render(컴포넌트, 컨테이너)
```

- 루트 컴포넌트 출력하기 (소스 코드 10-1-1.html)

```
01 <script type="text/babel">
02  // 컴포넌트와 컨테이너 생성하기
03  const component = <h1>리액트 기본</h1>
04  const container = document.getElementById('root')  //document.querySelector('#root')
05
06  // 출력하기
07  ReactDOM.render(component, container)
08 </script>
```

바벨 덕분에 사용할 수 있는 JavaScript 코드

문서 객체에서는 JavaScript 코드를 사용하기 위해 **innerHTML** 속성을 사용함

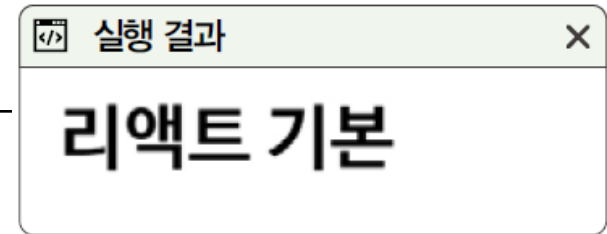


SECTION 10-1 리액트의 기본(4)

루트 컴포넌트 출력하기

JavaScript Extension 문법

```
01 <script type="text/babel">
02 // 컴포넌트와 컨테이너 생성하기
03 const component = <h1>리액트 기본</h1>    바벨 덕분에 사용할 수 있는 JavaScript 코드
04 const container = document.getElementById('root') //document.querySelector(`#root`)
05
06 // 출력하기
07 ReactDOM.render(component, container)
08 </script>
```



바벨 프로그램

- 자바스크립트 → react 요소를 변환하여 웹브라우저에서 사용 가능하도록 함

<https://babeljs.io/docs/setup/> → [try it out] 클릭 [좌측에 복사한 내용 붙여넣기] 에서 확인해보면

const component = /*#__PURE__*/React.createElement("h1", null, "\uB9AC\uC561\uD2B8 \uAE30\uBCF8");

const component = /*#__PURE__*/_react__jsxRuntime.jsx("h1", { children: "\uB9AC\uC561\uD2B8 \uAE30\uBCF8" })

→ const component = React.createElement("h1", null, 리액트 기본); 로 해석

SECTION 10-1 리액트의 기본(5)

◦ JSX 기본 문법

<태그>{표현식}</태그>

<태그 속성={표현식} />

따옴표를 사용하면 안 됩니다.

- 상수로 name과 imgUrl을 선언하고 이를 태그에 삽입해서 출력하는 코드 예시
- 표현식 출력하기 (소스 코드 10-1-2.html)

```
01 <script type="text/babel">
02  // 상수 선언하기
03  const name = '구름'
04  const imgUrl = 'http://placedog.net/400/200'
05
06  // 컴포넌트와 컨테이너 생성하기
07  const component =
08    <div>
09      <h1>{name} 님 안녕하세요!</h1>
10      <img src={imgUrl} />
11    </div>
12  const container = document.getElementById('root')
13
14  // 출력하기
15  ReactDOM.render(component, container)
16 </script>
```

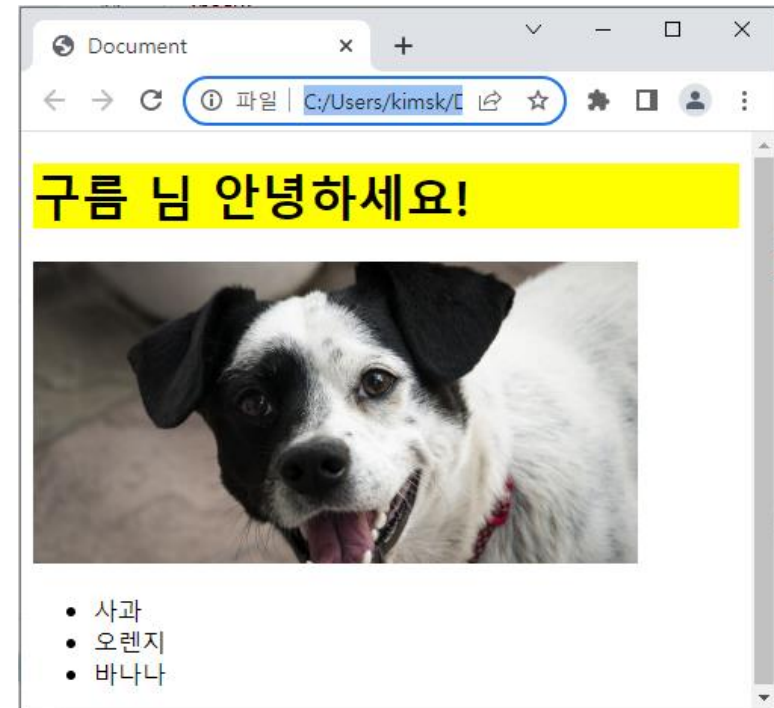


SECTION 10-1 리액트의 기본(6)

◦ JSX 기본 문법

- 상수로 name과 imgUrl, style, 배열(array), 핸들러(handler)를 선언하고 이를 태그에 삽입해서 출력하는 코드 예시

```
01 <script type="text/babel">
02  // 상수 선언하기
03  const name = '구름'
04  const imgUrl = 'http://placedog.net/400/200'
05  const bc = { backgroundColor : 'yellow' }
06  const array = [<li>사과</li>, <li>오렌지</li>, <li>바나나</li>]
07  const handler = (event) => { alert('클릭 했습니다!') }
08
09  // 컴포넌트와 컨테이너 생성하기
10  const component =
10  <div onClick = {handler}>
12    <h1 style = {bc} >{name} 님 안녕하세요!</h1>
13    <img src={imgUrl} />
14    <ul>{array}</ul>
15  </div>
16  const container = document.getElementById('root')
17
18  // 출력하기
19  ReactDOM.render(component, container)
20 </script>
```



SECTION 10-1 리액트의 기본(7)

클래스 컴포넌트

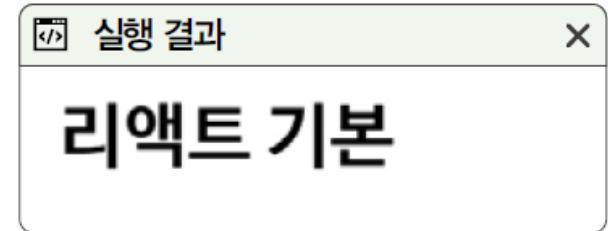
```
class 컴포넌트 이름 extends React.Component {  
  render () {  
    return <h1>출력할 것</h1>  
  }  
}
```

- 루트 컴포넌트 출력을 클래스 컴포넌트로 구현하기 (소스 코드 10-1-3.html)

```
01 <script type="text/babel">  
02 // 애플리케이션 클래스 생성하기  
03 class App extends React.Component {  
04   render () {  
05     return <h1>리액트 기본</h1>  
06   }  
07 }  
08  
09 // 출력하기  
10 const container = document.getElementById('root')  
11 ReactDOM.render(<App />, container)  
12 </script>
```

React.Component를 상속

App 컴포넌트로 변경



10-1-1.html 실행 결과와 동일

```
<script type="text/babel">  
  // 컴포넌트와 컨테이너 생성하기  
  const component = <h1>리액트 기본</h1>  
  const container = document.getElementById('root')  
  
  // 출력하기  
  ReactDOM.render(component, container)  
</script>
```

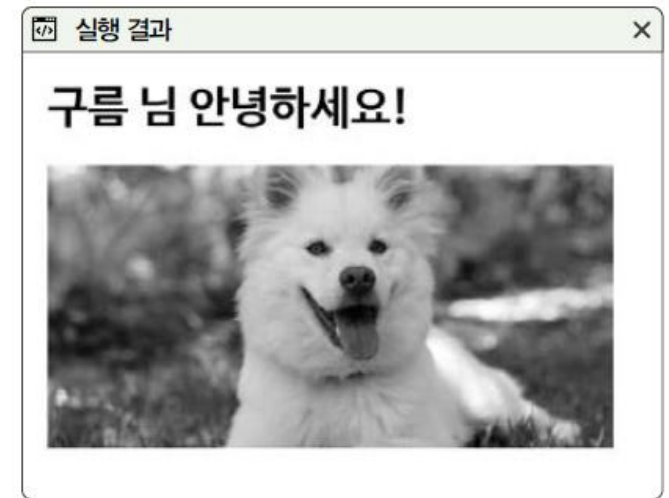
SECTION 10-1 리액트의 기본(8)

클래스 컴포넌트

- 클래스 컴포넌트를 사용하면 클래스 메소드 내부에서 `this.props` 속성을 사용할 수 있음
- 컴포넌트의 속성 사용하기 (소스 코드 10-1-4.html) └─ properties(속성) 을 나타내며 props로 사용

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    render () {
05      return <div>
06        <h1>{this.props.name} 님 안녕하세요!</h1>
07        <img src={this.props.imgUrl} />
08      </div>
09    }
10  }
11
12  // 출력하기
13  const container = document.getElementById('root')
14  ReactDOM.render(<App name="구름" imgUrl="http://placedog.net/400/200" />, container)
15 </script>
```

속성을 지정



10-1-2.html 실행 결과와 동일

SECTION 10-1 리액트의 기본(9)

- 컴포넌트의 기본적인 속성과 메소드
 - 클래스의 메소드 오버라이드하기

```
class App extends React.Component {  
  constructor (props) {  
    super(props)  
    // 생성자 코드  
  }  
  render () {  
    // 출력할 것  
  }  
  componentDidMount () {  
    // 컴포넌트가 화면에 출력될 때 호출  
  }  
  componentWillUnmount () {  
    // 컴포넌트가 화면에서 제거될 때 호출  
  }  
}
```

→ 생성자. 생성자가 여러 일을 해주므로,
super(props)를 사용해 부모 생성자를 호출

→ 컴포넌트가 내부적으로 특정 상황에
호출하는 메소드. 이런 메소드를 라이프사이클 메소드라고 칭함

SECTION 10-1 리액트의 기본(10)

◦ 컴포넌트의 기본적인 속성과 메소드

- state 속성에는 출력할 값을 저장(시간처럼 화면에 변화하는 값을 만들 때 사용)
 - state 속성 값을 변경할 때는 반드시 setState() 메소드를 사용
 - setState() 메소드로 속성의 값을 변경하면 컴포넌트는 render() 메소드를 호출해서 화면에 변경 사항을 출력

```
// 상태 선언하기(생성자 위치)
this.state = { 속성: 값 }
// 상태 변경하기(이외의 위치)
this.setState({ 변경할 속성: 값 })
```

SECTION 10-1 리액트의 기본(11)

◦ 컴포넌트의 기본적인 속성과 메소드

- 리액트를 활용한 현재 시간 출력 프로그램 (소스 코드 10-1-5.html)
 - this.state : 화면에 변화하는 값을 만들 때 사용하는 객체
 - this.setState : 화면에 변화하는 값을 지정하고 반영시킬 때 사용

```
01 <script type="text/babel">
02 class App extends React.Component {
03   constructor (props) {
04     super(props)
05     this.state = {
06       time: new Date()
07     }
08   }
09
10   render () {
11     return <h1>{this.state.time.toLocaleTimeString()}</h1>
12   }
13
```

시간을 출력할 것이므로 state 속성에 시간을 저장

state 속성에 있는 값을 출력

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(12)

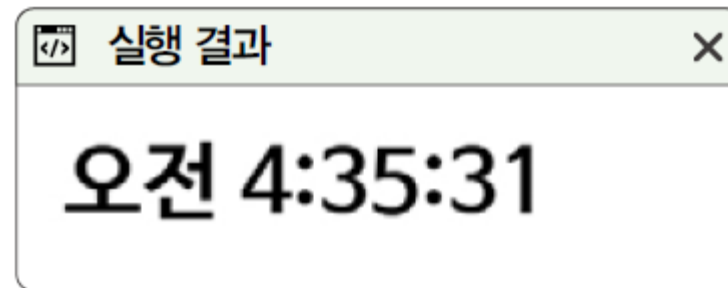
- 컴포넌트의 기본적인 속성과 메소드

- 리액트를 활용한 현재 시간 출력 프로그램 (소스 코드 10-1-5.html)

◀ 앞쪽에 이어

```
14 componentDidMount () {  
15   // 컴포넌트가 화면에 출력되었을 때  
16   this.timerId = setInterval(() =>{  
17     this.setState({  
18       time: new Date()  
19     })  
20   }, 1000)  
21 }  
22  
23 componentWillUnmount () {  
24   // 컴포넌트가 화면에서 제거될 때  
25   clearInterval(this.timerId)  
26 }  
27 }  
28  
29 // 출력하기  
30 const container = document.getElementById('root')  
31 ReactDOM.render(<App />, container)  
32 </script>
```

setState() 메소드를 사용해서 시간을 변경



SECTION 10-1 리액트의 기본(13)

이벤트 연결하기

- 컴포넌트에 이벤트를 연결할 때는

- 1) 메소드를 선언
- 2) 메소드에 this를 바인드
- 3) render() 메소드에서 출력하는 태그의 이벤트 속성에 메소드를 입력해서 이벤트를 연결

```
class App extends React.Component {  
  constructor(props) {  
    super(props)  
    this.메소드 이름 = this.메소드 이름.bind(this) → (2) 메소드에 this를 바인드  
  }  
  render () {  
    return <h1 이벤트 이름={this.메소드 이름}></h1>  
    → (3) 이벤트를 연결  
  }  
  메소드 이름 (event) {  
    // 이벤트가 호출될 때 실행할 코드  
    → (1) 메소드를 선언  
  }  
}
```

SECTION 10-1 리액트의 기본(14)

이벤트 연결하기

- 버튼을 클릭할 때 클릭한 횟수를 세는 코드 만들기
- 이벤트 연결하기 (소스 코드 10-1-6.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        count: 0
08      }
09
10    this.countUp = this.countUp.bind(this)
11  }
12
```

클릭한 횟수를 출력할 것이므로
state 속성에 일단 0을 저장

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(15)

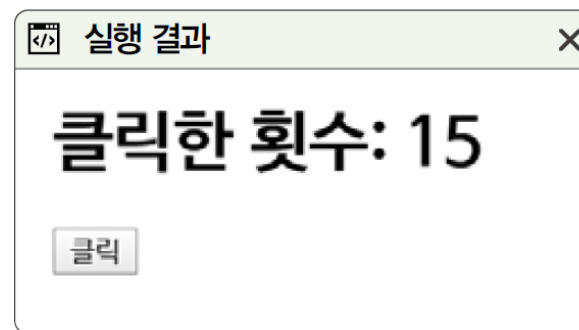
이벤트 연결하기

- 버튼을 클릭할 때 클릭한 횟수를 세는 코드 만들기
- 이벤트 연결하기 (소스 코드 10-1-6.html)

◀ 앞쪽에 이어

```
13 render () {  
14   return <div>  
15     <h1>클릭한 횟수: {this.state.count}</h1>  
16     <button onClick={this.countUp}>클릭</button>  
17   </div>  
18 }  
19  
20 countUp (event) {  
21   this.setState({  
22     count: this.state.count + 1  
23   })  
24 }  
25 }  
26  
27 // 출력하기  
28 const container = document.getElementById('root')  
29 ReactDOM.render(<App />, container)  
30 </script>
```

→ 대소문자를 지켜야 함



※ 리액트 이벤트 이름을 확인할 수 있는 주소
<https://ko.reactjs.org/docs/events.html#clipboard-events>

SECTION 10-1 리액트의 기본(16)

이벤트 연결하기

- `this.countUp = this.countUp.bind(this)`를 사용하지 않고 다음과 같은 2가지 형태를 사용하는 방법
- 이벤트 연결하기: 다른 `this` 바인드 방법(1) (소스 코드 10-1-7.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        count: 0
08      }
09    }
10
11    render () {
12      return <div>
13        <h1>클릭한 횟수: {this.state.count}</h1>
14        <button onClick={(e) => this.countUp(e)}>클릭</button>
15      </div>
16    }
```

event를 화살표 함수로 만들어 `countUp()` 메소드에 인수로 전달

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(17)

◦ 이벤트 연결하기

- `this.countUp = this.countUp.bind(this)`를 사용하지 않고 다음과 같은 2가지 형태를 사용하는 방법
- 이벤트 연결하기: 다른 `this` 바인드 방법(1) (소스 코드 10-1-7.html)

◀ 앞쪽에 이어

```
17
18   countUp (event) {
19     this.setState({
20       count: this.state.count + 1
21     })
22   }
23 }
24
25 // 출력하기
26 const container = document.getElementById('root')
27 ReactDOM.render(<App />, container)
28 </script>
```

SECTION 10-1 리액트의 기본(18)

◦ 이벤트 연결하기

- `this.countUp = this.countUp.bind(this)`를 사용하지 않고 다음과 같은 2가지 형태를 사용하는 방법
- 이벤트 연결하기: 다른 `this` 바인드 방법(2) (소스 코드 10-1-8.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        count: 0
08      }
09    }
10
11    render () {
12      return <div>
13        <h1>클릭한 횟수: {this.state.count}</h1>
14        <button onClick={this.countUp}>클릭</button>
15      </div>
16    }
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(19)

◦ 이벤트 연결하기

- `this.countUp = this.countUp.bind(this)`를 사용하지 않고 다음과 같은 2가지 형태를 사용하는 방법
- 이벤트 연결하기: 다른 `this` 바인드 방법(2) (소스 코드 10-1-8.html)

◀ 앞쪽에 이어

```
17
18 countUp = (event) => {
19   this.setState({
20     count: this.state.count + 1
21   })
22 }
23 }
24
25 // 출력하기
26 const container = document.getElementById('root')
27 ReactDOM.render(<App />, container)
28 </script>
```

countUp 함수를 화살표 함수로 지정

SECTION 10-1 리액트의 기본(20)

- 이벤트 연결하기
 - 입력 양식 사용하기 (소스 코드 10-1-9.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        text: ""
08      }
09      this.handleChange = this.handleChange.bind(this)
10    }
11
12    render () {
13      return <div>
14        <input
15          value={this.state.text}
16          onChange={this.handleChange} />
17        <h1>{this.state.text}</h1>
18      </div>
19    }
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(21)

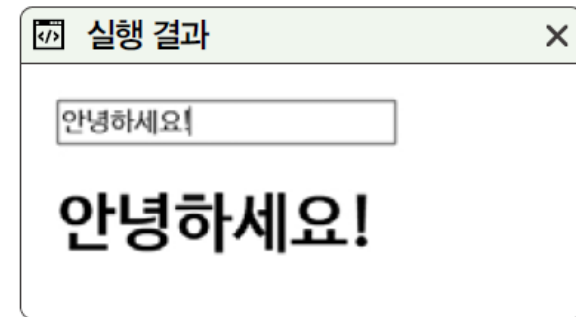
이벤트 연결하기

- 입력 양식 사용하기 (소스 코드 10-1-9.html)

◀ 앞쪽에 이어

```
20
21 handleChange (event) {
22   this.setState({
23     text: event.target.value
24   })
25 }
26 }
27
28 // 출력하기
29 const container = document.getElementById('root')
30 ReactDOM.render(<App />, container)
31 </script>
```

이벤트가 발생할 때 this.state의 text 속성에 입력 양식의 값을 넣음



SECTION 10-1 리액트의 기본(22)

스타일 지정하기

- style 속성에 객체를 지정

```
render () {  
  const style = {}  
  return <h1 style={style}>글자</h1>  
}
```

- 문서 객체 모델에서 살펴본 것과 마찬가지로 style 객체에는 캐멀 케이스로 속성을 입력
 - 문서 객체 모델 때와 차이점이 있다면 숫자를 입력할 때 단위를 입력하지 않아도 된다는 점

CSS 스타일 속성 이름	가능한 형태(1)	가능한 형태(2)
color: red	{ color: 'red' }	{ 'color': 'red' }
font-size: 2px	{ fontSize: 2 }	{ 'fontSize': 2 }

SECTION 10-1 리액트의 기본(23)

스타일 지정하기

- 이벤트와 스타일 지정을 모두 활용해서 체크되어 있을 때는 파란색, 체크되어 있지 않을 때는 붉은색으로 글자를 출력하는 코드
- 체크 상태에 따라서 스타일 지정하기 (소스 코드 10-1-10.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        checked: false
08      }
09
10    this.handleClick = this.handleClick.bind(this)
11  }
12
13  render () {
14    const textStyle = {
15      color: this.state.checked ? 'blue' : 'red'
16    }
```

체크되어 있다면 blue,
체크되어 있지 않다면 red를 출력

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(24)

- 스타일 지정하기
 - 체크 상태에 따라서 스타일 지정하기 (소스 코드 10-1-10.html)

◀ 앞쪽에 이어

```
17
18   return <div>
19     <input
20       type="checkbox"
21       onClick={this.handleClick} />
22     <h1 style={textStyle}>글자</h1>
23   </div>
24 }
25
26 handleClick (event) {
27   this.setState({
28     checked: event.target.checked
29   })
30 }
31 }
32
33 // 출력하기
34 const container = document.getElementById('root')
35 ReactDOM.render(<App />, container)
36 </script>
```

이벤트 객체를 활용해서
체크 상태를 설정

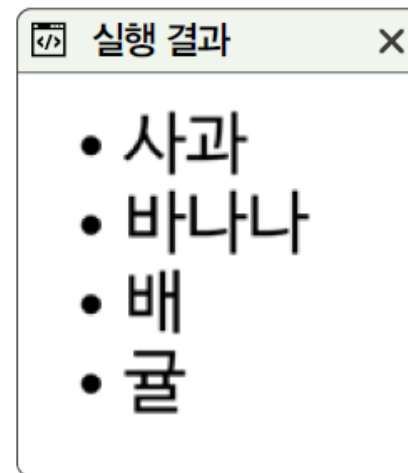


SECTION 10-1 리액트의 기본(25)

◦ 컴포넌트 배열

- 컴포넌트를 요소로 갖는 배열을 사용해서 한 번에 여러 개의 컴포넌트를 출력
- 컴포넌트 배열 사용하기(1) (소스 코드 10-1-11.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    render () {
05      const list = [
06        <li>사과</li>,
07        <li>바나나</li>,
08        <li>배</li>,
09        <li>귤</li>
10      ]
11
12      return <ul>{list}</ul>
13    }
14  }
15
16  // 출력하기
17  const container = document.getElementById('root')
18  ReactDOM.render(<App />, container)
19 </script>
```



SECTION 10-1 리액트의 기본(26)

◦ 컴포넌트 배열

- this.state에 값 배열을 만들고 render() 메소드 내부에 map() 메소드를 사용해서 이를 컴포넌트 배열로 변환해서 출력하는 코드
- 컴포넌트 배열 사용하기(2) (소스 코드 10-1-12.html)

```
01 <script type="text/babel">
02 // 애플리케이션 클래스 생성하기
03 class App extends React.Component {
04   constructor(props) {
05     super(props)
06     this.state = {
07       fruits: ['사과', '바나나', '배', '귤']
08   }
09 }
10
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(27)

◦ 컴포넌트 배열

- this.state에 값 배열을 만들고 render() 메소드 내부에 map() 메소드를 사용해서 이를 컴포넌트 배열로 변환해서 출력하는 코드
- 컴포넌트 배열 사용하기(2) (소스 코드 10-1-12.html)

◀ 앞쪽에 이어

```
11 render () {  
12   // 항목을 생성합니다.  
13   const list = this.state.fruits.map((item) => {  
14     return <li>{item}</li>  
15   })  
16   // 출력합니다.  
17   return <ul>{list}</ul>  
18 }  
19 }  
20  
21 // 출력하기  
22 const container = document.getElementById('root')  
23 ReactDOM.render(<App />, container)  
24 </script>
```

SECTION 10-1 리액트의 기본(28)

◦ 컴포넌트 배열

- 앞의 코드에서 map() 메소드를 다음과 같이 표현식으로 삽입해서 사용
- 컴포넌트 배열 사용하기(3) (소스 코드 10-1-13.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor(props) {
05      super(props)
06      this.state = {
07        fruits: ['사과', '바나나', '배', '귤']
08      }
09    }
10  }
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-1 리액트의 기본(29)

◦ 컴포넌트 배열

- 앞의 코드에서 map() 메소드를 다음과 같이 표현식으로 삽입해서 사용
- 컴포넌트 배열 사용하기(3) (소스 코드 10-1-13.html)

◀ 앞쪽에 이어

```
11  render () {
12    return <ul>{
13      this.state.fruits.map((item) => {
14        return <li>{item}</li>
15      })
16    }</ul>
17  }
18 }
19
20 // 출력하기
21 const container = document.getElementById('root')
22 ReactDOM.render(<App />, container)
23 </script>
```

this.state.fruits.map(item) => 를 과 사이에 넣어 줌

[마무리①]

- 3가지 키워드로 정리하는 핵심 포인트

- 리엑트는 사용자 인터페이스(UI)를 쉽게 구성할 수 있게 도와주는 라이브러리
- 컴포넌트는 리엑트에서 화면에 출력되는 요소를 의미
- JSX는 자바스크립트 코드 내부에서 HTML 태그 형태로 컴포넌트를 만들 수 있게 해주는 자바스크립트 확장 문법

- 확인 문제

1. 다음 중에서 틀린 것은?

- ① JSX 확장 문법을 사용하면 자바스크립트 코드 내부에 HTML 태그를 입력해서 사용할 수 있음
- ② JSX 확장 문법은 HTML 태그를 내부적으로 `React.createElement()` 메소드 호출로 변경해줌
- ③ `React.Component` 클래스를 상속받지 않아도 컴포넌트를 화면에 출력할 수 있음
- ④ 화면에 컴포넌트를 출력할 때는 `render()` 메소드를 사용

[마무리②]

◦ 확인 문제

2. 다음 중에서 틀린 것은?

- ① 속성을 지정할 때는 따옴표를 입력하면 안 됨
- ② 리액트의 이벤트 이름은 기존의 HTML 이벤트 이름과 같은 이름을 사용
- ③ style 속성으로 스타일을 지정할 때, 숫자의 경우 px 단위를 붙이지 않고 입력해도 됨
- ④ 스타일을 지정할 때는 따옴표를 입력하면 안 됨

3. 리액트의 컴포넌트는 어떤 클래스를 상속받아야 하는지 고르기

- ① React.Component
- ② ReactDOM.Component
- ③ React.DOMObject
- ④ React.Object

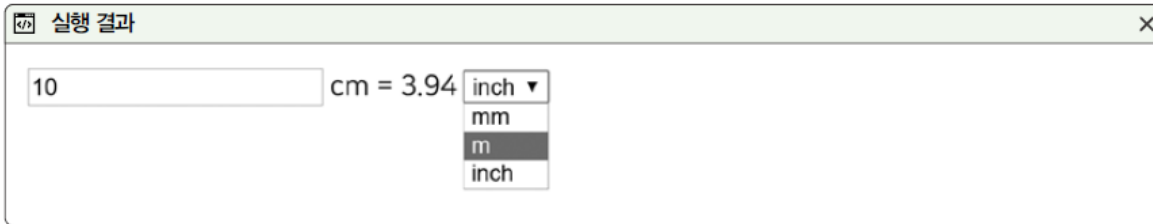
[마무리③]

◦ 확인 문제

4. 7장에서 살펴보았던 체크했을 때 작동하는 타이머 프로그램을 리액트로 구현하기



5. 7장에서 살펴보았던 단위 환산 프로그램을 리액트로 구현하기



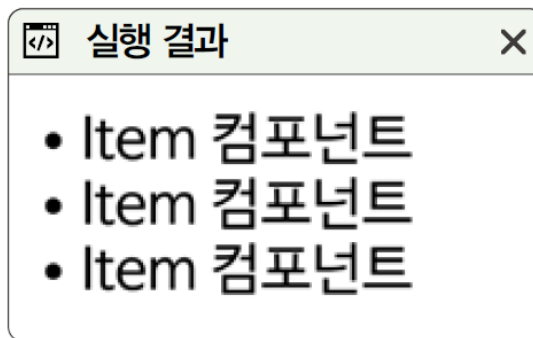
6. 7장에서 살펴보았던 글자 수를 세는 프로그램을 리액트로 구현하기. Chapter 7-2(책 355쪽)의 <좀 더 알아보기> 부분에서 살펴보았던 것처럼 타이머를 사용하는 형태로 구현



SECTION 10-2 리액트와 데이터(1)

- 여러 개의 컴포넌트 사용하기
 - Item 컴포넌트를 만들고 사용하는 코드
 - Item 컴포넌트 만들기 (소스 코드 10-2-1.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    render () {
05      return <ul>
06        <Item />
07        <Item />
08        <Item />
09      </ul>
10    }
11  }
12
13  class Item extends React.Component {
14    render () {
15      return <li>Item 컴포넌트</li>
16    }
17  }
18
19  // 출력하기
20  const container = document.getElementById('root')
21  ReactDOM.render(<App />, container)
22 </script>
```



SECTION 10-2 리액트와 데이터(2)

- 여러 개의 컴포넌트 사용하기
 - Item 컴포넌트에서 value 속성을 출력
 - Item 컴포넌트에 속성 전달하기 소스 코드 10-2-2.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    render () {
05      return <ul>
06        <Item value="Item 컴포넌트 1번" />
07        <Item value="Item 컴포넌트 2번" />
08        <Item value="Item 컴포넌트 3번" />
09      </ul>
10    }
11  }
12
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
    <!-- 리액트 사용 준비 -->
    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <!-- 리액트를 사용하는 코드 입력 -->
  </body>
</html>
```

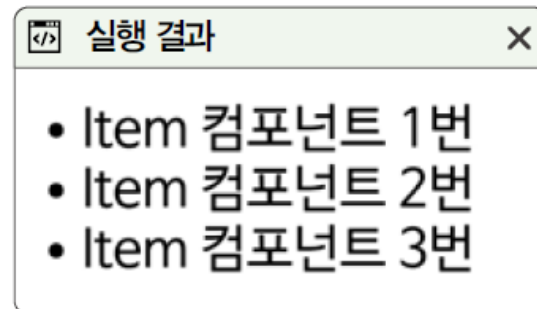
▶ 다음 쪽에 코드 이어짐

SECTION 10-2 리액트와 데이터(3)

- 여러 개의 컴포넌트 사용하기
 - Item 컴포넌트에서 value 속성을 출력
 - Item 컴포넌트에 속성 전달하기 소스 코드 10-2-2.html)

◀ 앞쪽에 이어

```
13 class Item extends React.Component {
14   constructor (props) {
15     super(props)
16   }
17
18   render () {
19     return <li>{this.props.value}</li>
20   }
21 }
22
23 // 출력하기
24 const container = document.getElementById('root')
25 ReactDOM.render(<App />, container)
26 </script>
```



SECTION 10-2 리액트와 데이터(4)

- 부모에서 자식의 state 속성 변경하기
 - 부모 컴포넌트에서 시간을 구하고, 이를 속성을 통해 자식 컴포넌트에게 전달하는 코드
 - componentDidUpdate() 메소드가 중요
 - Item 컴포넌트에 속성 전달하기 (소스 코드 10-2-3.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        time: new Date()
08      }
09    }
10
11    componentDidMount () {
12      // 컴포넌트가 화면에 출력되었을 때
13      this.timerId = setInterval(() => {
14        this.setState({
15          time: new Date()
16        })
17      }, 1000)
18    }
19  }
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-2 리액트와 데이터(5)

- 부모에서 자식의 state 속성 변경하기
 - Item 컴포넌트에 속성 전달하기 (소스 코드 10-2-3.html)

◀ 앞쪽에 이어

```
19
20 componentWillUnmount () {
21   // 컴포넌트가 화면에서 제거될 때
22   clearInterval(this.timerId)
23 }
24
25 render () {
26   return <ul>
27     <Item value={this.state.time.toLocaleString()} />
28     <Item value={this.state.time.toLocaleString()} />
29     <Item value={this.state.time.toLocaleString()} />
30   </ul>
31 }
32 }
33
34 class Item extends React.Component {
35   constructor (props) {
36     super(props)
37     this.state = {
38       value: props.value
39     }
40   }
```

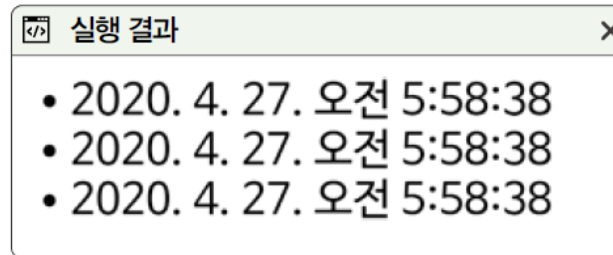
SECTION 10-2 리액트와 데이터(6)

- 부모에서 자식의 state 속성 변경하기
 - Item 컴포넌트에 속성 전달하기 (소스 코드 10-2-3.html)

◀ 앞쪽에 이어

```
41
42 componentDidUpdate (prevProps) {
43   if (prevProps.value !== this.props.value) {
44     this.setState({
45       value: this.props.value
46     })
47   }
48 }
49
50 render () {
51   return <li>{this.state.value}</li>
52 }
53 }
54
55 // 출력하기
56 const container = document.getElementById('root')
57 ReactDOM.render(<App />, container)
58 </script>
```

- 최초엔 실행되지 않음
- 자식 컴포넌트에서 변경을 적용할 때 사용하는 코드
- 고정적인 코드이므로 알아두면 좋음



SECTION 10-2 리액트와 데이터(7)

- 자식에서 부모의 state 속성 변경하기
 - 자식에서 부모의 state 속성 변경하기 (소스 코드 10-2-4.html)

```
01 <script type="text/babel">
02  // 애플리케이션 클래스 생성하기
03  class App extends React.Component {
04    constructor (props) {
05      super(props)
06      this.state = {
07        value: ""
08      }
09      this.changeParent = this.changeParent.bind(this)
10    }
11
12    render () {
13      return <div>
14        <CustomInput onChange={this.changeParent} />
15        <h1>{this.state.value}</h1>
16      </div>
17    }
```

▶ 다음 쪽에 코드 이어짐

SECTION 10-2 리액트와 데이터(8)

- 자식에서 부모의 state 속성 변경하기

- 자식에서 부모의 state 속성 변경하기 (소스 코드 10-2-4.html)

◀ 앞쪽에 이어

```
18
19  changeParent (event) {
20    this.setState({
21      value: event.target.value
22    })
23  }
24 }
25
26 class CustomInput extends React.Component {
27   render () {
28     return <div>
29       <input onChange={this.props.onChange} />
30     </div>
31   }
32 }
33
34 // 출력하기
35 const container = document.getElementById('root')
36 ReactDOM.render(<App />, container)
37 </script>
```

자신의 속성을 변경하는 메소드
내부에서 this 키워드를 사용하므로, this 바인드를 적용

input 태그에 변경 사항이 있을 때,
부모로부터 전달받은 메소드를 호출



SECTION 10-2 리액트와 데이터(9)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]
 - 할 일 목록 만들기 (소스 코드 10-2-5.html)

class App		
	save ()	this.state 를 localStorage에 state 로 저장
	load ()	localStorage의 state 를 읽어 파싱 후 output으로 출력
	componentDidUpdate ()	save () 호출
	render ()	제목과 TodoItem을 호출 todos.map() 메소드를 활용해 컴포넌트 (dataKey, isDone, text, removeItem, changeCheckData) 배열을 만들어 화면 출력
	textChange (event)	텍스트 입력 감지
	textKeyDown (event)	텍스트 입력이 끝나고 엔터를 입력하면 버튼(추가하기)클릭 호출
	buttonClick (event)	전개 연산자를 활용해서 기존의 배열을 복사하고, 뒤에 요소를 추가
	removeItem (key)	filter() 메소드를 활용해서 기존의 배열에서 키값이 같은 요소를 제거
	changeCheckData (key, changed)	변경된 요소를 찾고, isDone 속성을 변경
class TodoItem		
	render ()	checkbox, text, button(클릭 시 removeItem (key) 호출) 화면 출력
	checkboxClick ()	isDone이 변경된 경우 dataKey와 isDone으로 changeCheckData (key, changed) 호출
	componentDidUpdate (prevProps)	isDone 속성이 부모로부터 변경되면 출력에 반영



SECTION 10-2 리액트와 데이터(9)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]
 - 할 일 목록 만들기 (소스 코드 10-2-5.html)

```
01 <script type="text/babel">
02 // 애플리케이션 클래스 생성하기
03 class App extends React.Component {
04   constructor (props) {
05     super(props)
06
07     // 지난 설정 불러오기
08     this.state = this.load()
09
10     // 메소드 바인드
11     this.textChange = this.textChange.bind(this)
12     this.textKeyDown = this.textKeyDown.bind(this)
13     this.buttonClick = this.buttonClick.bind(this)
14     this.removeItem = this.removeItem.bind(this)
15     this.changeCheckData = this.changeCheckData.bind(this)
16   }
17 }
```



하늘색 부분은 localStorage 객체를 사용해서
할 일 목록을 저장하고 읽어들이는 부분을 표시

state 속성 전체를 읽어들이м([실행]-[검사]-[Application]-[LocalStorage])에서
state 에 저장된 내용 확인 가능

▶ 다음 쪽에 코드 이어짐

SECTION 10-2 리액트와 데이터(10)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
18 save () {  
19   localStorage.state = JSON.stringify(this.state)  
20 }  
21 load () {  
22   let output  
23   try { output = JSON.parse(localStorage.state)  
24   } catch (e) {}  
25   if (output !== undefined  
26     && output.keyCount !== undefined  
27     && output.currentValue !== undefined)  
28   {  
29     output = JSON.parse(localStorage.state)  
30   } else {  
31     output = {  
32       keyCount: 0,  
33       currentValue:"",  
34       todos: []  
35     }  
36   }  
37   return output  
38 }
```

state 속성 전체를 읽어들이

속성이 제대로 존재하는지 확인

속성이 없거나 제대로 되어
있지 않으면 초기값 할당

SECTION 10-2 리액트와 데이터(11)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
39
40 componentDidUpdate () {
41   this.save()
42 }
43
44 render () {
45   return <div>
46     <h1>할 일 목록</h1>
47     <input
48       value={this.state.currentValue}
49       onChange={this.textChange}
50       onKeyDown={this.textKeyDown} />
51     <button onClick={this.buttonClick}>추가하기</button>
52     <div>
53       {this.state.todos.map((todo) => {
54         return <TodoItem
55           dataKey={todo.key}
56           isDone={todo.isDone}
57           text={todo.text}
58           removeItem={this.removeItem}
59           changeCheckData={this.changeCheckData} />
60       }}}
61     </div>
```



todos.map() 메소드를 활용해
컴포넌트 배열을 만들

SECTION 10-2 리액트와 데이터(12)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
62   </div>
63   }
64
65   handleChange (event) {
66     this.setState({
67       currentValue: event.target.value
68     })
69   }
70
71   textKeyDown (event) {
72     const ENTER = 13
73     if (event.keyCode === ENTER) {
74       this.buttonClick()
75     }
76   }
77
```



입력 양식에서 Enter 키를 입력했을 때도
버튼을 클릭한 것과 같은 효과

SECTION 10-2 리액트와 데이터(13)

◦ 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

• 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
78  buttonClick (event) {
79    if (this.state.currentValue.trim() !== "") {
80      this.setState({
81        todos: [...this.state.todos, {
82          key: this.state.keyCount.toString(),
83          isDone: false,
84          text: this.state.currentValue
85        }]
86      })
87      this.state.keyCount += 1
88      this.state.currentValue = ""
89    }
90  }
91
92  removeItem (key) {
93    this.setState({
94      todos: this.state.todos.filter((todo) => {
95        return todo.key !== key
96      })
97    })
98  }
```

전개 연산자를 활용해서 기존의 배열을 복사하고,
뒤에 요소를 추가
setState() 메소드를 호출하지 않으면 배열의 변경이
화면에 반영되지 않으므로, 이런 코드를 사용



filter() 메소드를 활용해서 기존의 배열에서 요소를 제거

SECTION 10-2 리액트와 데이터(14)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
99
100 changeCheckData (key, changed) {
101   let target = [...this.state.todos]
102   target.filter((todo) => todo.key === key)[0].isDone = changed
103   this.setState({
104     todos: target
105   })
106 }
107 }
108
109 class TodoItem extends React.Component {
110   constructor (props) {
111     super(props)
112     this.state = {
113       isDone: props.isDone
114     }
115     this.checkboxClick = this.checkboxClick.bind(this)
116   }
117 }
```

배열을 전개 연산자로 복사

변경된 요소를 찾고, isDone 속성을 변경

SECTION 10-2 리액트와 데이터(15)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
118 render () {
119   const textStyle = {}
120   textStyle.textDecoration
121     = this.state.isDone ? 'line-through' : "      → isDone 의 상태에 따라 할 일에 라인을 긋거나 그대로 유지
122   return (
123     <div style={textStyle}>
124       <input
125         type="checkbox"
126         checked={this.state.isDone}
127         onChange={this.checkboxClick}/>
128
129       <span>{this.props.text}</span>
130       <button onClick={()=>this.props.removeItem(this.props.dataKey)}>제거
131     </button>      → 부모에게 어떤 항목의 제거 버튼을 클릭했는지 알려줌
132   </div>
133 )
134 }
135
```

SECTION 10-2 리액트와 데이터(16)

- 리액트로 만드는 할 일 목록 애플리케이션[누적 예제]

- 할 일 목록 만들기 (소스 코드 10-2-5.html)

◀ 앞쪽에 이어

```
136 checkboxClick () {  
137   const changed = !this.state.isDone  
138   this.props.changeCheckData(this.props.dataKey, changed)  
139 }  
140  
141 componentDidUpdate (prevProps) {  
142   if (prevProps.isDone !== this.props.isDone) {  
143     this.setState({  
144       isDone: this.props.isDone  
145     })  
146   }  
147 }  
148 }  
149  
150 // 출력하기  
151 const container = document.getElementById('root')  
152 ReactDOM.render(<App />, container)  
153 </script>
```

→ isDone 속성이 부모로부터 변경되면 출력에 반영





부록 A

구 버전의 인터넷 익스플로러 지원하기

[부록 A] 구 버전의 인터넷 익스플로러 지원하기

◦ 인터넷 익스플로러 개요

- 2015년에 11버전을 마지막으로 더 이상 메이저 버전이 바뀌지 않고 있으며, 2020년 8월을 기준으로 완전히 업데이트가 종료
- 국내에서는 ActiveX 등의 요인으로 인해서 아직도 높은 점유율
 - 2020년 10월을 기준으로 모든 플랫폼(스마트폰, 데스크톱 포함)에서 5%, 데스크톱 웹 브라우저 9%의 점유율
- 인터넷 익스플로러에서 웹 페이지를 제대로 운용하려면 최신 자바스크립트 기능을 사용하면 안 됨



[부록 A] 구 버전의 인터넷 익스플로러 지원하기

- 최신 자바스크립트 코드를 인터넷 익스플로러에서 실행하기
 - 바벨(Babel): 트랜스 컴파일러, 최신 버전의 자바스크립트로 작성된 코드를 구 버전에서 작동하게 만들어주는 변환기
 - Babel REPL
 - Babel REPL(<https://babeljs.io/repl>)
 - 코드를 입력할 수 있는 왼쪽 영역에 코드를 입력하면 오른쪽 영역에 구 버전에서 실행할 수 있는 코드로 변환되어 출력

// 간단한 코드

```
const a = [1, 2, 3, 4]
```

```
a.filter((v) => v % 2 !== 0).forEach(console.log)
```

// 이 책에서도 다루지 않은 더 최신 버전의 코드

```
const b = 10
```

```
const c = b ?? 20
```



```
"use strict";
```

// 간단한 코드

```
var a = [1, 2, 3, 4];
```

```
a.filter(function (v) {  
  return v % 2 !== 0;  
}).forEach(console.log);
```

// 이 책에서도 다루지 않은 더 최신 버전의 코드

```
var b = 10;
```

```
var c = b !== null && b !== void 0 ? b : 20;
```

[부록 A] 구 버전의 인터넷 익스플로러 지원하기

- 최신 자바스크립트 코드를 인터넷 익스플로러에서 실행하기
 - 폴리필(Polyfill)
 - 문법적으로 변환을 해주어도 기능적인 변환을 해주지는 않음. 예를 들어 배열의 filter(), forEach() 메소드 등은 구 버전의 인터넷 익스플로러에서 동작하지 않음.
 - 이러한 메소드 등의 기능을 추가해줌
 - 대표적인 폴리필: Modernizr, es-shims, Polyfill.io 등
 - es-shims
 - es5-shim: <https://github.com/es-shims/es5-shim>
 - es6-shim: <https://github.com/es-shims/es6-shim>

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/es5-shim/4.5.14/es5-shim.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.35.5/es6-shim.min.js"></script>
```
