

10. Introduction to React

Content

- ECMAScript (ES6) 핵심 기능
- React 개요 및 개발 환경
- React 작동 방식
- JSX
- React Components
- Props & States
- Hooks
- Examples

ECMAScript (ES6) 핵심 기능

- **let**: block scope를 갖는 변수 선언

```
var topic = "JavaScript";
if (topic) {
  var topic = "React";
  console.log(topic); // "React"
}
console.log(topic); // "React"
// 위 1행과 3행의 topic은 같은 변수
```

```
let topic = "JavaScript";
if (topic) {
  let topic = "React";
  console.log(topic); // "React"
}
console.log(topic); // "JavaScript"
//위 1행과 3행의 topic은 서로 다른 변수
```

- **const**: block scope를 갖는 상수 선언

- 선언과 동시에 값, 배열, 객체, 함수 등을 할당하고 그 후 변경 불가
 - 주의: 배열 또는 객체가 할당된 경우, 그 배열의 원소나 객체의 속성은 변경 가능

```
const PI = 3.141592653589793;
PI = 3.14; // error!
const nums = ["one", "two", "three"];
nums = ["five", "six", "eight"]; // error!
nums[0] = "four"; // OK
nums.push("five"); // OK
```

```
const sandwich = { // 객체
  bread: "wheat", meat: "tuna", cheese: "swiss"
};
sandwich = { bread: "oat",
  meat: "turkey" }; // error!
sandwich.bread = "hearty"; // OK 2
```

ECMAScript (ES6) 핵심 기능

- **`...`**: Template String (Template Literal)

- +를 이용한 문자열 접합(concatenation) 연산 대용
- 문자열 내에 변수/식의 결과를 삽입 가능 (`${...}`) 이용 → JSP의 EL과 유사)
- 문자열 내 white-space(공백/탭/개행문자)들을 유지

```
const name = lastName + ", " + firstName + " " + middleName;
→ const name = `${lastName}, ${firstName} ${middleName}`;
```

```
document.body.innerHTML = `
  <section>
    <article>
      <h2>${article.title}</h2>
      ${article.body}
    </article>
    <footer>
      <p>copyright ${new Date().getFullYear()} </p>
    </footer>
  </section>
`; // white-space들을 모두 보존
```

ECMAScript (ES6) 핵심 기능

□ Arrow Function

- 함수 표현식(anonymous function)을 간략히 표현하는 방법

```
const hello = function() { // 함수 표현식
  var greeting = "Hello World!";
  return greeting;
}
→ const hello = () => { // function 키워드 생략, => 사용
  var greeting = "Hello World!";
  return greeting;
}
```

```
const sum = function(x, y) {
  return x + y;
};
→ const sum = (x, y) => { return x + y; };
→ const sum = (x, y) => x + y; // 본문이 한 문장이면 중괄호 및 return 생략 가능
```

```
const square = function(x) {
  return x * x;
}
→ const square = x => x * x; // 인자가 하나일 경우 괄호 () 생략 가능
```

ECMAScript (ES6) 핵심 기능

□ 구조 분해(de-structuring)

- 객체 구조 분해: 객체 안의 속성 값들을 각각 새로운 변수에 할당

```
const sandwich = { // 객체
  bread: "wheat", meat: "tuna", cheese: "swiss"
};

let {bread, meat} = sandwich; // sandwich 객체를 구조 분해(두 속성을 추출, 할당)
→ let bread = sandwich.bread, meat = sandwich.meat; 와 동일

console.log(bread + ', ' + meat); // "wheat, tuna"

bread = "oat"; // 변수 값 변경 (sandwich 객체와 무관)
console.log(bread); // "oat"
console.log(sandwich.bread); // "wheat"

const printMeat = ({meat}) => { // 함수에 전달된 객체를 구조 분해(meat 속성 추출)
  console.log(meat); // meat 속성 값 출력
};

printMeat(sandwich); // 매개변수로 sandwich 객체 전달 → meat 속성 값 "tuna" 출력6
```

ECMAScript (ES6) 핵심 기능

- 주의: 함수가 객체를 직접 return하는 경우, 객체를 괄호 ()로 묶어야 함

```
const buildName = function(firstName, lastName) {
  return { // 객체 생성 및 반환
    first: firstName,
    last: lastName
  };
}
→ const buildName = (firstName, lastName) => ({
  first: firstName,
  last: lastName
}); // 괄호 ()가 없을 경우 중괄호 {}를 함수의 영역 표시로 간주하여 error 발생!

var CEOofTesla = buildName("Elon", "Musk");
console.log(CEOofTesla); // {first: "Elon", last: "Musk"}
```

5

ECMAScript (ES6) 핵심 기능

- 배열 구조 분해: 배열의 원소들을 각각 새로운 변수에 할당

```
let numbers = ["one", "two", "three"];
const [first] = numbers; // let first = numbers[0]; 와 동일
const [first, third] = numbers;
// let first = numbers[0], third = numbers[2]; 와 동일
```

□ 객체 리터럴 개선(object literal enhancement)

- 객체 구조 분해의 역: 주어진 변수들을 속성으로 이용하여 새로운 객체 생성

```
let bread = "wheat";
let meat = "tuna";
let cheese = "swiss";
const sandwich = { bread, meat, cheese }; // 객체 생성
// sandwich = { bread: bread, meat: meat, cheese: cheese } 와 동일
// == { bread: "wheat", meat: "tuna", cheese: "swiss" }; 7
```

ECMAScript (ES6) 핵심 기능

□ Spread 연산자 (...)

- 배열의 원소들이나 객체의 속성들의 나열을 나타냄
- 기존 배열의 원소들이나 객체의 속성들을 이용해서 새로운 배열 또는 객체를 생성 가능

```
const numbers = [23, 55, 21, 87, 56]; // 배열
let maxValue = Math.max(...numbers); // Math.max(23,55,21,87,56)와 동일
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const hf1 = [...q1, ...q2]; // == ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]

const sandwich1 = { // 객체
  bread: "wheat", meat: "tuna"
};
const cheese = "cheddar";
const sandwich2 = {
  ...sandwich1, cheese
};
// sandwich2 == { bread: "wheat", meat: "tuna", cheese: "cheddar" };
```

8

ECMAScript (ES6) 핵심 기능

□ Fetch API

- 비동기 요청 실행(Ajax): XMLHttpRequest, jQuery.ajax()의 대안
- 요청 전송, 응답 대기, 요청 처리 결과에 따라 callback 함수(들)을 호출
 - fetch(): 비동기적 요청 전송, Promise 객체 반환
 - Promise.then(): 요청 성공 시 호출될 callback 함수를 등록
 - Promise.catch(): 요청 실패 시 호출될 callback 함수를 등록
 - 요청 성공 시: response 객체 생성, then()으로 등록된 callback 함수 호출
 - ✓ 등록된 callback 함수가 여러 개일 경우 순차적으로 호출
 - 이전 callback 함수의 결과가 다음 callback 함수의 인자로 전달됨
 - 요청 실패 시: error 객체 생성, catch()로 등록된 callback 함수 호출

```
fetch('https://jsonplaceholder.typicode.com/posts/1') // Ajax GET 요청
.then(response => response.json()) // response body의 내용(JSON)을 parsing한 객체
.then(data => console.log('Success: ', data.userId, data.title))
.catch(error => console.error('Error: ', error));
```

10

ECMAScript (ES6) 핵심 기능

□ Spread 연산자 (...)

- Rest parameter
 - 함수의 인자 정의에 spread 연산자 사용
 - 함수 호출 시 전달된 파라미터 값들을 하나의 배열로 조합(배열 생성)

```
directions("Seoul", "Suwon", "Inchun", "Daegu"); // 아래 함수 호출
↓
function directions(...args) { // args == ["Seoul", "Suwon", "Inchun", "Daegu"]
  var [start, ...remaining] = args; // 배열 구조 분해, spread 연산자 사용
  var [finish, ...stops] = remaining.reverse(); // 배열 구조 분해, spread 연산자 사용
  console.log(`${start}, ${finish}, ${stops}`); // Seoul, Daegu, ["Inchun", "Suwon"] 출력
}
→ start == "Seoul", remaining == ["Suwon", "Inchun", "Daegu"]
   remaining.reverse() == ["Daegu", "Inchun", "Suwon"]
   finish == "Daegu", stops == ["Inchun", "Suwon"]
```

9

ECMAScript (ES6) 핵심 기능

- 참고: async 및 await 를 이용한 비동기 함수 정의 및 사용 방법

```
async function postData(url, data) { // async: 비동기 함수 정의(Promise 반환)
  try {
    const response = await fetch(url, { // await: 비동기 요청이 완료될 때까지 대기
      method: 'POST', // POST 요청의 예
      headers: {
        'Content-Type': 'application/json' // & other headers...
      },
      body: JSON.stringify(data), // data(JS 객체)를 JSON 문자열로 변환
    });
    return response.json(); // 요청 성공 시 response(JSON)를 parsing한 JS 객체 반환
  } catch(error) { console.error('Error: ', error); } // 요청 실패 시 실행
}
// 위 함수 호출
postData('https://jsonplaceholder.typicode.com/posts',
  { "userId": 100, "id": 12, "title": "TEST", "body": "TEST POST" })
  .then(data => console.log('Success: ', data)); // data: 위 함수에서 반환된 객체
```

ECMAScript 핵심 기능

배열 처리 함수

- pop(), push(), shift(), unshift(), join(), concat(), splice(), slice(), indexOf() 등

```
const nums1 = [1, 2, 3, 4];
let n = nums1.pop();           // n == 4, nums1 == [1, 2, 3]
nums1.push(5);                 // nums1 == [1, 2, 3, 5]
n = nums1.shift();             // n == 1, nums1 == [2, 3, 5]
nums1.unshift(0);              // nums1 == [0, 2, 3, 5]
const str = nums1.join(", ");  // str == "0, 2, 3, 5"
const nums2 = [6, 7, 8, 9];
const nums3 = nums1.concat(nums2); // nums3 == [0, 2, 3, 5, 6, 7, 8, 9]
// 주의: nums1, nums2의 원소들은 불변

const strs = ["A", "B", "C", "D", "E"];
let rem = strs.splice(1, 3);    // strs == ["A", "E"], rem == ["B", "C", "D"]
strs.splice(1, 0, "B", "C");    // strs == ["A", "B", "C", "D"]
rem = strs.splice(2, 2, "E", "F", "G");
// strs == ["A", "B", "E", "F", "G"], rem == ["C", "D"]
const sub1 = strs.slice(2);     // sub1 == ["E", "F", "G"] (주의: strs는 불변)
const sub2 = strs.slice(1, 3);  // sub2 == ["B", "E"] (주의: strs는 불변)
let pos = strs.indexOf("B");    // pos == 1
```

ECMAScript 핵심 기능

- Array iteration 함수 사용 예

```
const nums1 = [16, 4, 9, 45, 25];

const nums2 = nums1.map(val => val * 2); // nums2 == [32, 8, 18, 90, 50]

const nums3 = nums2.reduce(
  (total, val) => (total + val), 0); // 원소들의 합 계산: total==198
// 결과 변수: 배열의 원소, total의 다음 값, total의 초기값

const nums4 = nums2.reduce(
  (max, val) => (val > max ? val : max), 0); // 최대값 원소 찾기: max==90
// 결과 변수: 배열의 원소, max의 다음 값, max의 초기값

const nums5 = nums2.filter(val => val > 20); // nums5 == [32, 90, 50]

const nums6 = nums5.find(val => val > 40); // nums6 == 90
const nums7 = nums5.findIndex(val => val > 40); // nums7 == 1
```

ECMAScript 핵심 기능

Array iteration 함수

- 배열의 모든 원소에 대해 미리 정의된 작업(callback 함수)을 반복 실행
- forEach(fn) – 각 원소에 대해 한 번씩 callback function fn 실행
- every(fn) / some(fn) – 모든/어떤 원소에 대해 함수 fn의 실행 결과가 true 인지 검사
- find(fn) / findIndex(fn) – 함수 fn의 실행 결과가 true인 첫번째 원소 / 그 원소의 index를 찾아 반환
 - 만족하는 원소가 없으면 undefined / -1을 반환
- filter(fn) – 함수 fn의 실행 결과가 true인 원소들로 구성된 새로운 배열 생성 및 반환
- map(fn) – 각 원소에 대한 함수 fn의 결과 값들로 구성되는 새로운 배열 생성 및 반환
- reduce(fn) – 각 원소에 대해 함수 fn을 실행하면서 최종적으로 하나의 결과 값을 생성 및 반환
 - 결과 값은 primitive value, object, 또는 function 가능

React 개요

React (ReactJS, <https://react.dev/>)

- Web application에서 front-end user interface를 효율적으로 구현하기 위한 JavaScript library
- Single-Page Application(SPA) 방식의 응용 프로그램 개발에 적합
- Facebook(Meta Platforms)에서 개발 및 관리 (2013.5~)
- 유사 기술: Vue.js, AngularJS, Angular(based on TypeScript)

특징

- 선언적, 함수형 프로그래밍 (Declarative & Functional Programming)
- 재사용 가능한 UI 컴포넌트 정의: DOM 엘리먼트 생성
- Virtual DOM 트리 생성 및 이용: Real DOM 트리를 효율적으로 갱신(rendering)하기 위한 in-memory data structure
- JSX 언어 사용: JavaScript 및 HTML 코드를 간결하게 작성 가능

React 개발 환경

개발 도구

- Visual Studio Code (<https://code.visualstudio.com/>) 이용
- Node.js(<https://nodejs.org/en>) 설치 → npm, npx 포함
- create-react-app (<https://create-react-app.dev/>)
 - React 기반 프로젝트 생성 도구
 - webpack, Babel, ESLint 등의 빌드 도구를 포함
 - React, ReactDOM, react-scripts 등의 라이브러리에 대한 dependency 자동 설정
 - 설치: cmd/terminal 에서 'npm install -g create-react-app' 실행
- React 프로젝트 생성
 1. create-react-app을 설치한 후 작업 폴더로 이동, 'create-react-app project-name' 실행
 2. 또는 create-react-app을 설치하지 않고 'npx create-react-app project-name' 실행 (권장)
- 프로젝트 빌드 및 실행
 - project-name 폴더로 이동한 후 'npm install' 실행
 - ✓ 프로젝트에 필요한 라이브러리 설치(node_modules 폴더) 후 빌드 실행
 - 'npm start' 실행
 - 웹 브라우저를 실행하고 <http://localhost:3000> 요청
- 배포를 위한 production build 생성
 - 'npm run build'
 - ✓ build 폴더 내에 최적화된 빌드 결과를 생성 → 웹 서버에 배포 가능

16

React 개발 환경

프로젝트 구조

- node_modules
 - 프로젝트에 필요한 라이브러리들을 포함
- public
 - index.html, image 등 정적 파일 포함
 - 컴파일에서 제외되고, 외부에서 요청으로 접근 가능
- src
 - JavaScript(JSX) 소스 파일 포함 (컴파일 대상)
 - 예: index.js, App.js
- package.json
 - 프로젝트에 대한 기본 설정, 라이브러리들에 대한 dependency 설정 포함
- .gitignore
- README.md

```
my-app
├── node_modules
├── public
├── src
├── .gitignore
├── package.json
└── README.md
```

17

React 개발 환경

- public/index.html

```
<html lang="en">
  <head> ... </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

- src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(
  document.getElementById('root'));

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

- src/App.js

```
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        ...
      </header>
    </div>
  );
}

export default App;
```

18

React 작동 원리

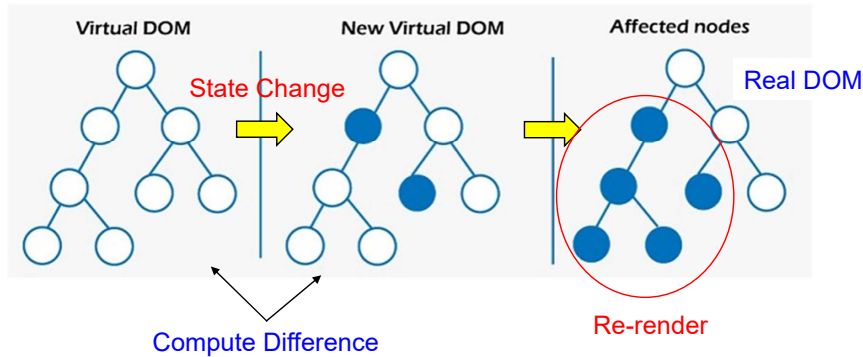
동작 방식

- 메모리 내에 Virtual DOM tree 생성 및 활용
 - 브라우저의 실제 DOM 트리를 직접 변경하지 않고, 가상 DOM 트리를 생성 및 조작(갱신)
 - React component들의 상태 변화(데이터 변경) 발생 시 새로운 가상 DOM을 생성하고 이전 버전의 가상 DOM과 차이점을 비교 분석
 - 실제 DOM 트리에서 변경이 필요한 노드들만 찾아 효율적으로 갱신 실행
 - ✓ 브라우저에서 변경이 필요한 부분만 re-rendering
- UI 화면(일부)의 변화가 자주 발생하는 애플리케이션의 경우 rendering 성능 향상 효과가 큼

19

React 작동 원리

- Virtual DOM을 통한 UI elements 갱신 과정



20

React 작동 원리

- React code in HTML

```
<html>
<head>
  <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>      <!-- target container -->

  <script type="text/babel">    // JSX code
    function Greeting() {
      return (<h1>Hello World!</h1>);
    }

    const container = document.getElementById("root");
    const root = ReactDOM.createRoot(container);
    root.render(<Greeting />);
  </script>
</body>
</html>
```

별도의 모듈(JS 파일)들로 분리 가능

21

React 작동 원리

- JSX 모듈 정의

- index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Greeting from './Greeting';
// Greeting 모듈에서 기본적으로 노출된 객체를 import (Greeting 이름으로 사용)
import './index.css';

const container = document.getElementById("root");
const root = ReactDOM.createRoot(container);
root.render(
  <Greeting />      // 아래 모듈의 Greeting() 함수를 호출한 후 결과를 rendering함
);
```

```
<div id="root">
  <h1>Hello World!</h1>
</div>
```

- Greeting.js: 함수 컴포넌트 Greeting 정의

```
import './Greeting.css';

function Greeting() {
  return (<h1>Hello World!</h1>);      // <h1>Hello World!</h1> 생성
}
export default Greeting;      // 외부에 기본적으로 노출할 객체(함수)의 이름 지정
```

22

React 작동 원리

- 컴포넌트에 속성 전달

```
import React from 'react';      // index.js
import ReactDOM from 'react-dom/client';
import Greeting from './Greeting';
import './index.css';

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(
  <div>
    <Greeting name="World"/>      /* 아래의 함수 호출 시 name 속성을 인자로 전달*/
    <Greeting name="Jain"/>
  </div>
);
```

```
<div id="root">
  <div>
    <h1>Hello World!</h1>
    <h1>Hello Jain!</h1>
  </div>
</div>
```

```
import './Greeting.css';      // Greeting.js

function Greeting(props) {
  return (<h1>Hello {props.name}</h1>);      // 전달된 name 속성 값 참조 및 출력
}
export default Greeting;
```

23

React 작동 원리

□ Rendering

- `const root = ReactDOM.createRoot(container)`
 - Virtual DOM tree의 루트 노드를 생성 및 반환
 - `container`: React 노드를 출력할 HTML element 노드 지정
 - ✓ `public/index.html` 내에 미리 정의된 엘리먼트 지정 (예: `<div id="root" />`)
- `root.render(reactNode)`
 - 루트 노드 아래에 React 노드를 생성 → rendering 실행
 - `reactNode`: JSX로 표현된 React 및 HTML element들의 sub-tree 포함
 - ✓ `React.createElement(component, props, ...children)` 함수를 이용하는 JavaScript 코드들이 생성 및 실행됨

```
root.render(  
  <Greeting name="Jain"/> → React.createElement(Greeting, {name: 'Jain'}, null)  
);  
function Greeting(props) {  
  return (<h1>Hello {props.name}!</h1>);  
}                                     → React.createElement('h1', null, `Hello ${props.name}!`)
```

JSX 개요

- 주의 사항
 - 여러 행으로 표현되는 엘리먼트는 괄호 ()로 묶음
 - 여러 개의 엘리먼트들은 반드시 하나의 최상위 엘리먼트의 자식으로 정의되어야 함
 - ✓ wrapper element(예: `<div>`) 사용을 피하기 위해 `fragment(<>)` 활용 가능
 - 표현식 내에서는 if 문 사용 불가 → `?:` 연산자 활용

```
const list = (  
  <ul>  
    <li>Dr. Jim Gray</li>  
    <li>{student1}</li>  
    <li>{student2}</li>  
  </ul>  
>);  
  
const greeting = <h1>{x > 10 ?  
  "Hello" : "Goodbye"}</h1>;
```

```
const listWithTitle = (  
  <>      /* <div> 대신 사용 */  
    <h1>Subject: CS101</h1>  
    <ul>  
      <li>Dr. Jim Gray</li>  
      <li>{student1}</li>  
      <li>{student2}</li>  
    </ul>  
  </>  
>);
```

26

JSX 개요

□ JSX (JavaScript XML)

- JavaScript(ES6) 언어 문법의 확장
- JavaScript 코드 내에 HTML 및 React element를 직접 사용 가능
 - 실행 시간에 순수한 JavaScript 코드로 변환됨 → element 객체 생성
 - ✓ Babel 프로그램이 변환 수행 (<https://babeljs.io/>)
- 표현식(JavaScript Expressions): `{expr}`
 - Element의 속성이나 content에 JavaScript 식(변수, 객체, 배열, 연산, 함수 호출 등)의 결과를 전달 가능

```
const first = "Elon"; const last = "Musk";  
const title = "CEO";  
const name = <span>{first} {last}</span>;           // string  
  
<input type="checkbox" defaultChecked={false} />      // boolean  
<h1>{"Hello " + title}</h1>                         // operation  
<h1>{title.toLowerCase()}</h1>                     // function call
```

25

JSX 개요

□ 배열 → Element list로 변환

- Iteration code

```
// data = ["a", "b", "c", ...];  
let listItems = [];           // 배열 생성  
for (let i = 0; i < data.length; i++) // 배열의 원소들 생성 및 저장  
  listItems.push(<li key={i}>Data Value: {data[i]}</li>);  
let list = <ul>{listItems}</ul>; // 배열의 원소(<li>)들이 모두 포함됨
```

- Functional programming code (권장)

- `map()` 함수 활용

```
let list = (<ul>  
  {data.map((val, i) => <li key={i}>Data Value: {val}</li>)}  
</ul>);  
// * val: 배열의 원소, i: 원소의 index
```

- `key`: 동적으로 생성되는 엘리먼트들에 대해 **식별자 속성** 정의(유일한 값 할당)
→ rendering 성능 향상을 위해 필요

27

Components

□ React Components

- 독립적이고 재사용 가능한 코드 단위
- HTML 및 React element들을 생성 및 반환

□ 종류

1. Class-based components

- `React.Component` 클래스를 확장하여 JavaScript 클래스로 정의
- 내부에 상태 변수 정의, life-cycle 함수 사용 가능

```
class Article extends React.Component {
  this.state = { title: this.props.title, desc: this.props.desc };
  render() {
    return (
      <article>
        <h2>{this.state.title}</h2>
        {this.state.desc}
      </article>
    );
  }
}
```

두 개의 상태 변수 `title, desc`를 정의하고 부모에서 전달된 속성 값을 각각 저장

상태 변수 `title, desc`의 값을 참조(출력)

28

Components

2. Function-based components (권장)

- 하나의 props 객체 인자를 갖고 하나의 HTML / React element를 반환하는 함수 정의
- 상태 정의 및 life-cycle 이벤트 처리를 위해 "Hook" 이용 (`useState` 등)

```
function Article({title, desc}) { // props 객체를 구조 분해
  const [title, setTitle] = useState(title);
  const [desc, setDesc] = useState(desc);

  return (
    <article>
      <h2>{title}</h2>
      {desc}
    </article>
  );
}
```

두 개의 상태 변수 `title, desc`를 정의하고 부모에서 전달된 속성 값 `title, desc`를 각각 저장

각 상태 변수에 대한 setter 함수를 정의

29

Props & States

□ Props (Properties)

- React component에 전달되는 인자 - 함수의 인자(argument)와 유사
- HTML 속성 형식을 통해 상위 component에서 하위 component로 데이터 전달 가능
- Component에 전달된 props의 값은 변경 불가(read-only)

```
function ResetButton(props) {
  <button style={props.btnStyle} onClick={props.clickHandler}>R</button>
}

function Counter() {
  const btnStyle = {width: 30, height: 30, ...};
  const reset = (e) => setCount(0);
  return (
    <div>
      <ResetButton btnStyle={btnStyle} clickHandler={reset}/>
    </div>
  );
}
```

30

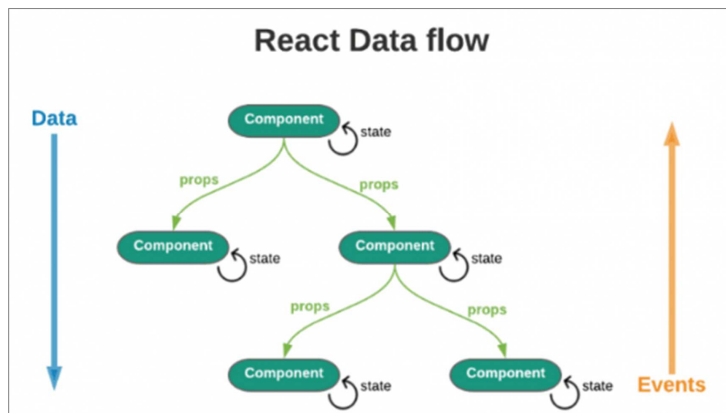
Props & States

□ States

- React component에서 내부적으로 상태 데이터를 유지하기 위한 변수들
- State 값은 하위 component들로 전파 가능 → 속성(props)을 통해 전달
- State 값이 변경되면 그 component의 render() 함수가 재실행됨 (re-rendering)
 - 그 State 값에 영향을 받는 하위 component들도 자동적으로 re-rendering 실행
- State 값 변경 시 주의 사항
 - 반드시 상태 변수에 대한 **setter 함수 호출을 통해** 변경해야 함
 - 상태 변수가 객체나 배열을 참조하고 있는 경우, 그것들의 필드나 원소를 변경하는 것이 아니라 변수의 **참조 대상을 새로운 객체나 배열로 변경해야 함**
 - ✓ 참조 대상은 불변 객체(immutable object)로 간주
- Function-based component에서 state 생성 및 관리는 `useState` hook을 이용

31

Props & States



Hooks

useState hook

- React component 내부의 state 및 state 변경 함수 정의
- `const [state, setState] = useState(initialState)`
 - `initialState` : 상태 변수의 초기 값
 - 상태 변수 `state`와 상태 값 변경을 위한 setter 함수 `setState`를 포함하는 배열을 반환

```
import {useState} from 'react';
function Counter() {
  const [count, setCount] = useState(0); // count 초기값: 0
  const increase = (e) => setCount(count + 1);
  return (
    <div>
      <button onClick={increase}>Click!</button>
      <p>You clicked {count} times</p> // count 값 출력
    </div>
  );
}
* 버튼 클릭 시 상태 변수 count의 값이 변경됨 → re-rendering 실행 → count 값 재출력
```

34

Hooks

개념

- Function-based component를 위한 state 관리와 life-cycle event 처리, rendering 성능 개선 등을 지원하기 위해 사용되는 함수들
- Built-in hooks
 - `useState` - component의 상태(state) 관리
 - `useEffect`, `useLayoutEffect` - component의 rendering 직후 실행될 관련 작업 정의 (side effect)
 - `useContext` - 부모와 자손 component들 간에 전역적으로 공유될 수 있는 변수 및 함수 정의
 - `useReducer` - component의 상태 변경 로직을 별도의 함수로 정의
 - `useRef` - component나 HTML element들에 대한 참조 생성
 - `useMemo`, `useCallback` - component의 rendering 성능 개선
 - 기타
- 기존 hook들을 합성해서 새로운 custom hook 정의 가능

33

Hooks

useEffect hook

- React component가 rendering될 때마다 기-정의된 부 작업(side effect)을 실행하는 effect function 정의
- `useEffect(function, dependency)`
 - `function` : 실행할 effect function
 - `dependency` : 상태 변수들에 대한 의존성 배열 (optional)
- 의존성 배열이 주어질 경우
 - 배열에 포함된 상태(들)의 변경 시에만 effect function이 호출됨
 - 빈 배열([])을 지정하면 component를 최초로 rendering한 직후에 한 번만 호출됨
 - Component의 초기화 작업에 유용
- effect function이 함수를 반환할 경우
 - Component가 Virtual DOM 트리에서 제거될 때 그 함수가 호출됨
 - Component의 소멸과 관련된 정리 작업 수행 가능

35

Hooks

useEffect hook

– 예

```
import { useState, useEffect } from "react";
function Timer() {
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  useEffect(() => {
    setTimeout(() => setCount(count + 1), 1000);
  }); // <Timer>가 rendering된 후 항상 실행됨 (1초마다 timer 재설정)

  useEffect(() => setCalculation(count * 2), [count]);
  // count 상태 값이 변경된 경우에만 실행됨
  return (
    <><h1>I've rendered {count} times!</h1>
    <p>Calculation: {calculation}</p></>
  ); }
}; }
```

36

Hooks

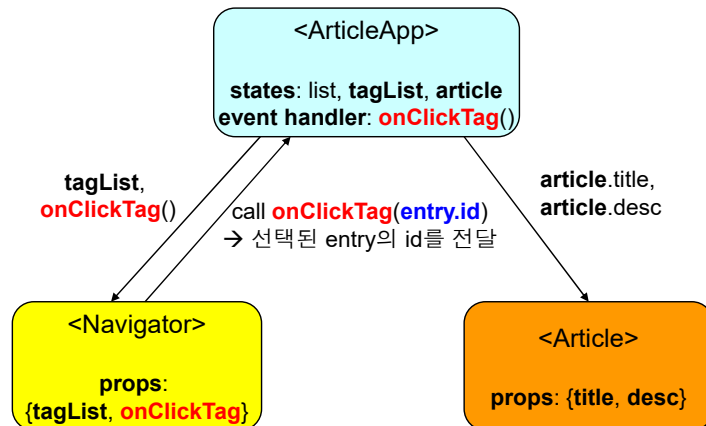
```
import { useState, useEffect } from "react";
function Timer() {
  const [count1, setCount1] = useState(0);
  const [count2, setCount2] = useState(10);
  const [calc1, setCalc1] = useState(0);
  const [calc2, setCalc2] = useState(0);
  const [calc3, setCalc3] = useState(0);
  const [calc4, setCalc4] = useState(0);

  useEffect(() => setCalc1(count1 + count2)); // <Timer>가
  (re)rendering 될 때마다 주어진 함수가 항상 실행됨
  useEffect(() => setCalc2(count1 + count2), [count1]); // count1 값이
  초기화되거나 변경되어 (re)rendering 된 경우에만 실행됨
  useEffect(() => setCalc3(count1 + count2), [count2]); // count2 값이
  초기화되거나 변경되어 (re)rendering 된 경우에만 실행됨
  useEffect(() => setCalc4(count1 + count2), []); // <Timer>가 최초로
  rendering된 직후 한 번만 실행됨
  ...
}
```

37

Example: ArticleApp

Component 계층 관계



38

Example: ArticleApp

ArticleApp.js – index.js의 root.render()에서 <ArticleApp/> 사용

```
import React, {useState} from 'react';
import Navigator from './Navigator'; import Article from './Article';

export default function ArticleApp() {
  const [list, setList] = useState( // list state: article 객체들의 list
    [ { "id":1, "title":"HTML", "desc":"..."}, // 배열 초기화
      { "id":2, "title":"JavaScript", "desc":"..."},
      { "id":3, "title":"React", "desc":"..." } ]
  );
  const [tagList, setTagList] = useState( // tagList state: tag 객체들의 list
    [ { "id":1, "title":"HTML"},
      { "id":2, "title":"JavaScript"},
      { "id":3, "title":"React" } ]
  );
  const [article, setArticle] = useState( // article state: 선택된 article 객체 참조
    { title: "Welcome!", desc: "An example of React and Ajax." } // 초기화
  );
}
```

39

Example: ArticleApp

- 하위 component에 대한 event handler(callback) 함수 정의 및 전달
 - <Navigator>에서 발생할 onClick event에 대한 처리 함수 정의
 - onClickTag props를 통해 <Navigator>에게 전달
 - <Navigator>에서 event가 발생한 대상의 ID를 입력 파라미터로 전달받음

```
return (
  <div className="App">
    <h1>React Example</h1>
    <Navigator tagList={tagList}
      onClickTag={({id}) => {
        // id: Navigator에서 선택된 tag의 id
        let article = list.find(entry => (entry.id === id));
        setArticle(article);
      }}/>
    <Article title={article.title} desc={article.desc} />
  </div>
);
```

Navigator의 특정 tag 클릭 시 실행될 event handler 함수 정의
→ onClickTag props로 전달

article state 변경 → <ArticleApp> 및 하위 component들을 re-rendering

Example: ArticleApp

- Article.js – ArticleApp()에서 <Article/> 사용

```
export default function Article({title, desc}) {
  return (
    <article>
      <h2>{title}</h2>
      {desc}
    </article>
  );
}
```

상위 component에서 전달된 props 객체를 구조 분해

/* article의 title 및 desc 값 출력 */

Example: ArticleApp

- Navigator.js – ArticleApp()에서 <Navigator/> 사용(호출)

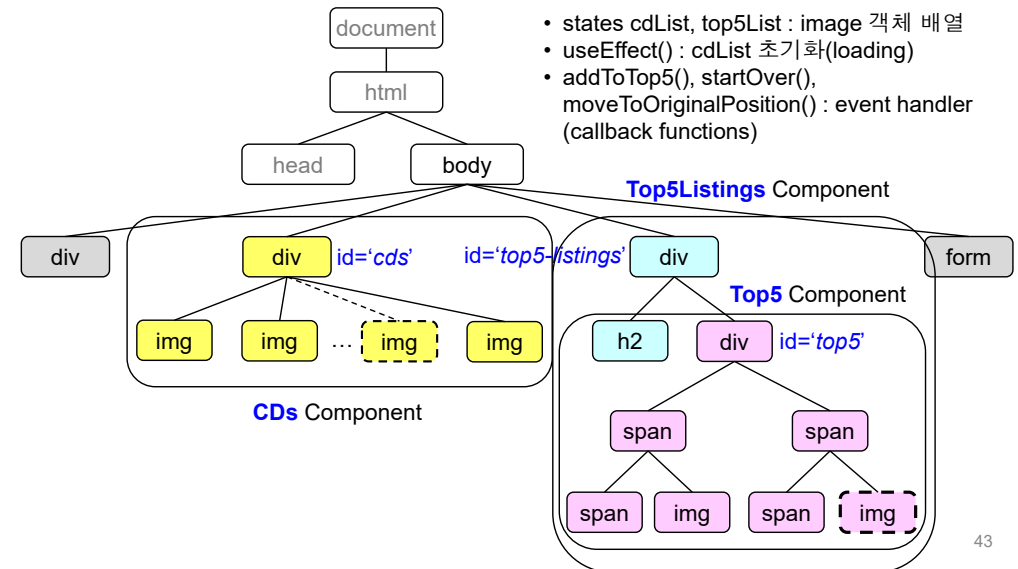
```
export default function Navigator({tagList, onClickTag}) {
  // Array.map()을 통해 tagList의 원소마다 <li> 엘리먼트를 생성함
  let list = tagList.map((entry) => (
    <li key={entry.id}>
      <a href={entry.id} onClick={
        (e) => {
          e.preventDefault();
          onClickTag(entry.id);
        }
      }>{entry.title}</a>
    </li>
  ));
  return (
    <nav>
      <ul>
        {list}
      </ul>
    </nav>
  );
}
```

<a>에 대한 onclick event handler 정의:

- <a>의 기본 동작을 막음
- 인자로 전달된 onClickTag() callback 함수를 호출 (선택된 entry의 id를 상위 component인 <ArticleApp>으로 전달)

Example: Top5CDApp

- DOM tree



Example: Top5CDApp

- Top5CDApp.js – index.js의 root.render()에서 <Top5CDApp/> 사용(호출)

```
import React, {useState, useEffect} from 'react';
import './Top5.css';

export default function Top5CDApp() {
  const [cdList, setCdList] = useState([]); // cdList, top5List state 생성
  const [top5List, setTop5List] = useState([]); // : image 객체들의 배열

  useEffect(() => { // cdList 초기화(App 시작 시 한 번만 호출됨)
    fetch('cdList.json') // Ajax 호출로 data를 가져와서 cdList state에 저장
      .then(response => response.json())
      .then(list => setCdList(list))
      .catch(error => console.error(error));
  }, []);

  const instr = `Click on a CD cover to add it to the Top 5 list ...`;
}
```

44

Example: Top5CDApp

- Top5CDApp.js (계속)

```
const addToTop5 = (selectedCd) => { // callback function
  if (top5List.length < 5) {
    const newCdList = cdList.filter(cd => (cd !== selectedCd));
    // cdList에서 selectedCd가 아닌 CD들만 추출해서 새로운 배열 생성 및 반환
    setCdList(newCdList); // cdList 상태 변경
    const newTop5List = top5List.concat(selectedCd);
    // top5List에 selectedCd를 추가한 새로운 배열 생성 및 반환
    setTop5List(newTop5List); // top5List 상태 변경
  }
  else {
    alert("You already have 5 cdList. Click \"Start Over\" to try again.");
  }
};
```

45

Example: Top5CDApp

- Top5CDApp.js (계속)

```
const startOver = () => { // callback function
  const list = Array.from(cdList); // cdList 배열 복사
  for (let curCd of top5List) { // top5List의 각 원소 curCd를 처리
    list.push(curCd); // curCd를 list의 마지막에 추가
  }
  setCdList(list); // cdList 상태 변경
  setTop5List([]); // top5List 상태 변경
};
```

Example: Top5CDApp

- Top5CDApp.js (계속)

```
const moveToOriginalPosition = (selectedCd) => { // callback function
  const newTop5List = ...
  // top5List에서 selectedCd가 아닌 cd들만 추출하여 새로운 배열 생성(Array.filter() 이용)
  setTop5List(newTop5List); // top5List 상태 변경
  const list = ... // cdList 배열 복사(또는 Array.from() 또는 [...cdList] 이용)
  let i = ...
  // list에서 id가 selectedCd.id 보다 큰 첫 번째 cd의 index를 구함(Array.filter() 이용)
  if (i >= 0) ... // selectedCd를 list의 i 위치에 삽입 (Array.splice() 이용)
  else ... // selectedCd를 list의 마지막에 추가 (Array.push() 이용)
  setCdList(list); // cdList 상태 변경
};
```

Example: Top5CDApp

□ Top5CDApp.js (계속)

```
return (  
  <>  
    <div id="instructions">{instr}</div>  
    <CDs images={cdList} onClickImage={addToTop5} /> { /* CD List 출력 */}  
    <Top5Listings name="Jain" images={top5List} // Top5 CD List 출력  
      { /* onClickImage={moveToOriginalPosition} */}  
    />  
    <form>  
      <input type="button" value="Start Over" onClick={startOver}/>  
    </form>  
  </>  
);  
}
```

48

Example: Top5CDApp

□ CDs.js – Top5CDApp()에서 <CDs/> 사용(호출)

```
export default function CDs({ images, onClickImage }) {  
  const imgTags = images.map(  
    image =>  
    <img key={image.id} src={image.src} className="cover" alt=""  
      onClick={ (e) => {  
        onClickImage(image);  
      } }  
    />  
  );  
  return (  
    <div id="cds">  
      {imgTags}  
    </div>  
  );  
}
```

cdList와 addToTop5()를 전달받음

 클릭 시 실행되는 callback 함수.
onClickImage(=addToTop5) callback 함수 호출
→ 선택된 image 객체를 인자로 전달
→ addToTop5에서는 전달된 객체를 top5List에 추가

images.map()을 통해 cdList의 각 항목(image 객체)
에 대응되는 엘리먼트들을 생성

생성된 들을 출력

49

Example: Top5CDApp

□ Top5Listings.js – Top5CDApp()에서 <Top5Listings/> 사용(호출)

```
export default function Top5Listings({ name, images, onClickImage })  
return (  
  <div id="top5-listings">  
    <h2>{name}'s Top5 CD List</h2>  
    <Top5 images={images} /*onClickImage={onClickImage}*/ />  
  </div>  
);  
}
```

top5List와 moveToOriginalPosition()을 각각 전달받음

50

Example: Top5CDApp

□ Top5Listings.js (계속)

```
function Top5({ images, onClickImage }) {  
  const imgTags = images.map(  
    (image, i) => (  
    <span key={image.id}>  
      <span className="rank">{i + 1}</span>  
      <img src={image.src} className="cover" alt=""  
        onClick={ ... } />  
    </span>  
    )  
  );  
  return (  
    <div id="top5">  
      {imgTags}  
    </div>  
  );  
}
```

top5List, moveToOriginalPosition()

 클릭 시 실행될 event handler 함수 정의:
파라미터로 넘어온 moveToOriginalPosition()
함수 호출 → 선택된 image 객체를 cdList에 추가

images.map()을 통해 top5List의 각 항목(image 객체)
에 대응되는 sub-tree들을 생성

생성된 sub-tree들을 출력

51

Example: ArticleApp

□ ArticleApp.js – index.js의 root.render()에서 <ArticleApp/> 사용

```
import React, {useState} from 'react';
import Navigator from './Navigator'; import Article from './Article';

export default function ArticleApp() {
  const [list, setList] = useState( // list state: article 객체들의 list
    [ {id:"1", "title":"HTML"},      // 배열 초기화
      {id:"2", "title":"JavaScript"},
      {id:"3", "title":"React"} ]
  );
  const [article, setArticle] = useState( // article state: 선택된 article 객체 참조
    {title: "Welcome!", desc: "An example of React and Ajax."} // 초기화
  );
  return (
    <div className="App">
      <h1>React Example</h1>
      <Navigator tagList={list} onClickTag={...} /> /* navigator 생성(list 전달) */
      <Article title={article.title} // 선택된 article 정보 출력
        desc={article.desc} />
    </div>
  ); }
```

52

Example: ArticleApp

– Fetch API를 이용한 Ajax 호출 (ArticleApp.js)

```
export default function ArticleApp() {
  ...
  const [article, setArticle] = useState(...);
  ...
  <div className="App">
    <h1>React Example</h1>
    <Navigator tagList={list}
      onClickTag={(id) => { // id: Navigator에서 선택된 항목의 id
        fetch(id + '.json') // 선택된 article에 관한 JSON data 요청
        .then(response => response.json()) // JSON parsing
        .then(obj => setArticle(obj)) // article state 변경(객체 저장)
        .catch(error => console.error(error));
      }} />
    <Article title={article.title} desc={article.desc} />
  </div>
  ...
}
```

53

Example: ArticleApp

□ Navigator.js – ArticleApp()에서 <Navigator/> 사용

```
export default function Navigator({tagList, onClickTag}) {
  // Array.map()을 통해 tagList의 원소(entry)마다 <li> 엘리먼트를 생성함
  let list = tagList.map((entry) => (
    <li key={entry.id}>
      <a href={entry.id} onClick={
        (e) => {
          e.preventDefault();
          onClickTag(entry.id); } }>
        {entry.title}</a>
    </li>
  ));
  return (
    <nav>
      <ul>
        {list}
      </ul>
    </nav>
  ) }
```

<a>에 대한 onclick event handler 정의:

- <a>의 기본 동작을 막음
- 인자로 전달된 onClickTag() callback 함수를 호출 (선택된 entry의 id를 상위 component인 <ArticleApp>으로 전달)

54

Example: Top5CDApp

□ Top5CDApp.js (계속)

```
const moveToOriginalPosition = (selectedCd) => { // callback function
  const newTop5List = top5List.filter(cd => (cd !== selectedCd));
  // top5List에서 selectedCd가 아닌 cd들만 추출해서 새로운 배열 생성 및 반환
  setTop5List(newTop5List); // top5List 상태 변경
  const list = Array.from(cdList); // cdList 배열 복사(또는 [...cdList] 이용)
  let i = list.findIndex(cd => (cd.id > selectedCd.id));
  // list에서 id가 selectedCd.id 보다 큰 첫 번째 cd의 index 반환
  if (i >= 0) list.splice(i, 0, selectedCd); // selectedCd를 list의 i 위치에 삽입
  else list.push(selectedCd); // selectedCd를 list의 마지막에 추가
  setCdList(list); // cdList 상태 변경
};
```

55

Example: Top5CDApp

□ Top5Listings.js (계속)

```
function Top5({ images, onClickImage }) { ← top5List, moveToOriginalPosition()
  const imgTags = images.map(
    (image, i) => (
      <span key={image.id}>
        <span className="rank">{i + 1}</span>
        <img src={image.src} className="cover" alt=""
          onClick={ (e) => { ← <img> 클릭 시 실행되는 event handler 함수
            console.log(event); onClickImage(image); } } />
        </span>
      )
    );
  return (
    <div id="top5">
      {imgTags} ← 생성된 <span> sub-tree들을 출력
    </div>
  );
}
```

moveToOriginalPosition()을 호출
→ 선택된 image 객체를 cdList에 추가

images.map()을 통해 top5List의 각 항목(image 객체)
에 대응되는 sub-tree들을 생성