React-1,2장

생성일	@2022년 5월 4일 오후 5:41
분류	React-리액트를 다루는 기술 책
주제	React
날짜	@2022년 5월 4일

React

- 오직 뷰(View)만 신경쓰는 라이브러리
- 컴포넌트(component): 특정부분이 어떻게 생길지 정하는 선언체
  - 재사용이 가능한 API
- 렌더링: 화면에 뷰(html DOM)를 보여주는것

초기 렌더링

- 맨 처음 어떻게 보일지 정하는 렌더링
- render()
  - 함수형 컴포넌트- JSX객체 반환
  - 컴포넌트가 어떻게 생겼는지 정의하는 역할
  - 뷰가 어떻게 생겼고 어떻게 작동하는지에 대한 정보를 지닌 객체를 반환
  - 실행시 그 내부에 있는 컴포넌트들도 재귀적으로 렌더링

조화 과정

- 업데이트= 화면 갱신
- 컴포넌트에서 데이터에 변화가 있을때 우리가 보기엔 변화에따라 뷰가 변형되는 것처럼 보이지만, 실제로는 새로운 요소로 갈아끼우기 때문
  ⇒ render()함수의 역할

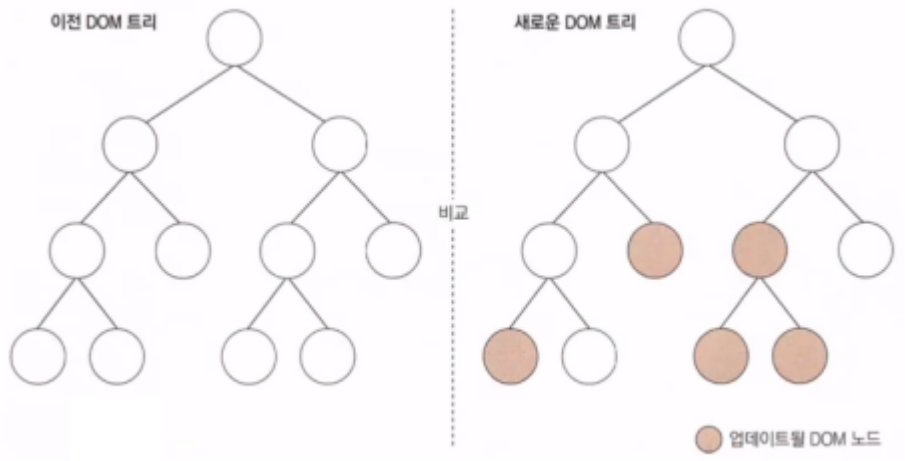
특징

DOM

- Document Object Model
- 객체로 문서 구조를 표현하는 방법
- XML이나 HTML로 작성→ Tag에 의한 구조 표현
- 동적 UI에 최적회 되어있지 않음→ JS를 사용해 동적으로 만들수 있음
- HTML 마크업을 시각적인 형태로 변환하는것- 웹 브라우저가 하는 주 역할임
- DOM을 조작 할때마다 엔진이 웹 페이지를 새로 그리기 때문에 업데이트가 너무 잦으면 성능이 저하 될수있음- useEffect로 회피
  - 웹 브라우저 단에서 DOM에 변화가 일어나면 웹 브라우저가 CSS를 다시 연산하고 레이아웃을 구성하고, 페이지를 리페인트함→ 이 과정에서 시간이 허비됨
    - 화면 HTML제어를 이벤트 핸들러안에 직접 제어x
  - 규모가 큰 웹 애플리케이션에서 DOM에 직접 접근하여 변화를 주면 성능 이슈가 발생할수 있음- 느려짐
  - DOM을 최소한으로 조작해 작업을 처리하는 방식으로 개선- useEffect의 역할
- Virtual DOM방식을 사용해 DOM업데이트를 추상화함으로써 DOM처리 횟수를 최소화하고 효율적으로 진행함

Virtual DOM

- 실제 DOM에 접근하여 조작하는 대신, 이를 추상화한 JS객체를 구성하여 사용
  - 실제 DOM: 브라우저에 노출되는 Element(크롬 개발자도구에 노출되는 요소)
  - 추상화한 JS 객체: DOM을 메모리에 복사한 것



▲ DOM 비교

- 이전 DOM 트리
  - 브라우저에 표시되고 있는 내용
  - 수정한 후 웹페이지를 다시 그림
  - 처리가 느려졌을때 대표적인 현상- 화면 깜빡임
    - 사람이 체감을 할정도로 컴퓨터의 속도가 느려짐
- 새로운 DOM 트리
  - 메모리에 복사

    → 메모리 안에서 수정

    → 전부 수정 후 전체를 덮어씌움
- 지속적으로 데이터가 변화하는 대규모 애플리케이션 구축

작업 환경 설정

Node.js

- React프로젝트를 만들때 Node.js를 반드시 먼저 설치해야함
- 프로젝트를 개발하는데 필요한 주요 도구들이 Node.js를 사용하기 때문에 설치 ⇒자동 실행됨
  - 바벨(babal): ES6를 호환시켜줌
  - 웹펙(webpack): 모듈화된 코드를 한 파일로 합치고(번들링) 코드를 수정할때 마다 웹 브라우저를 리로딩하는 등 여러 기능을 지님

- npm
  - Node.js패키지 매니저 도구
  - 패키지 버전 관리
  - 보다 진보된 도구: yarn
- yarn
  - npm을 대체할수 있는 도구
  - npm보다 더 빠르며 효율적인 캐시 시스템과 기타 부가 기능을 제공
  - npm을 계속 사용해도 무방
  - npm install -g(—global) yarn :명령어로 설치

VS Code확장 프로그램

- ESLint: JS문법 및 코드 스타일을 검사해줌
  - 설치후 터미널에서 npm install -g eslint 명령 실행 필요
- Reactjs Code Snippets- 제작자: charalampos Karypidis
  - 리액트 컴포넌트 및 라이프사이클 함수를 작성할때 단축 단어를 사용해 간편하게 코드를 자동으로 생성해 낼 수 있는 코드 스니펫 모음

React 프로젝트 생성

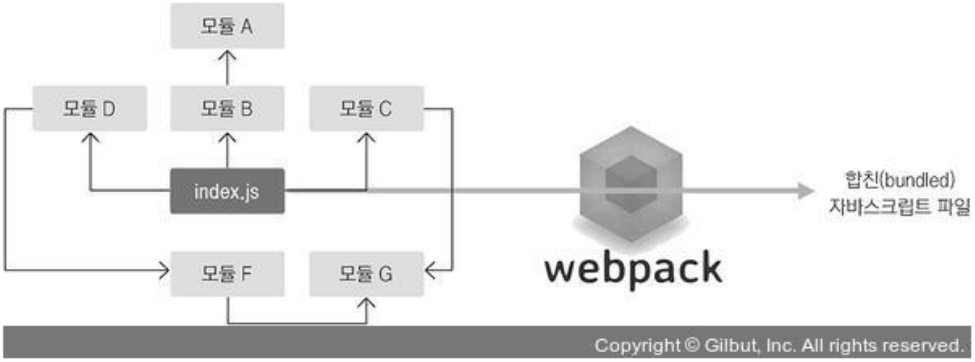
- yarn create react-app 프로젝트이름 명령 실행
- 소문자 x
- cd 프로젝트이름 :터미널에서 프로젝트로 이동
- yarn start : react 실행

JSX

import

- import를 사용해 다른 파일들을(JS) 불러와 사용

- 원래 브라우저에는 없던 기능- 일반 JS에서는 import 불가
- Node.js에서 지원하는 기능
  - require라는 구문으로 패키지를 불러올수 있다
    ⇒ X, 최신 Node.js에서는 import도 사용 가능
- 번들러(bundler) 사용: 분할된 js를 import해서 조립- 파일을 묶듯이 연결



⇒ yarn start명령이 해주는 것

- webpack
  - 번들러 사용
  - 웹 브라우저에서 사용할수 있게 병합해주는 중간 도구
- 합친 JS파일: 웹 브라우저에서 사용

ES6→ ES5 변환 과정

작성 <ES6>-Node용

↓ webpack

병합

↓ babal

변환 <ES5>- 웹 브라우저용

⇒ 이러한 과정을 거쳐 React 코드 실행

JSX

- 함수 컴포넌트
  - 프로젝트에서 컴포넌트를 렌더링하면 함수에서 반환하고 있는 내용을 나타냄
    - 렌더링: ‘보여준다’는것을 의미
  - return하는 코드들: JSX
- JSX
  - JS의 확장 문법
  - 코드가 번들링되는 과정에서 바벨을 사용해 일반 JS형태의 코드로 변환
  - 공식적인 JS문법이 아님- 개발자들이 임의로 만든 문법

```
function App(){
  return React.createElement('div',null,'Hello',React.createElement('b',null,'react'));
}

//위의 코드 축약
function App(){
  return(
    <div>
      Hello <b>react</b>
    </div>
  );
};
```

src/index.js

```
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>,
  document.getElementById('root')
);
```

- getElementbyId(’root’)
  ⇒id가 root인 요소안에 렌더링을 하도록 설정
  이 요소는 public/index.html파일을 열어보면 있음

index.html U

public > index.html > html > head > title

```
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
</html>
```

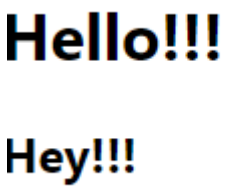
문법

- 컴포넌트에 여러 요소가 있다면 반드시 부모 요소 하나로 감싸야함
  - VirtualDOM에서 컴포넌트 변화를 감지해 낼 때 효율적으로 비교할수 있도록 컴포넌트 내부는 하나의 DOM트리 구조로 이루어져야 한다는 규칙이 있기 때문

```
function App() {
  return (
    <div>
      <h1>Hello!!!</h1>
      <h2>Hey!!!</h2>
    </div>
  );
}

export default App;
```

출력 결과



- Fragment 사용

출력 결과

```jsx
import {Fragment} from 'react';

function App() {
  return (
    <Fragment>
      <h1>Hello!!!</h1>
      <h2>Hey?</h2>
    </Fragment>
  );
}

export default App;
```

출력 결과

Hello!!!

Hey?

- <> </>

```jsx
function App() {
  return (
    <>
      <h1>Hello!!!</h1>
      <h2>Hey? yo</h2>
    </>
  );
}

export default App;
```

출력 결과

Hello!!!

Hey? yo

JS 표현

- 변수값 출력- var는 사용x⇒ var의 결점을 해결해주는 let, const 사용

출력 결과

```jsx
function App() {
  const name='React!'
  return (
    <>
      <h1>Hello!!!{name}</h1>
      <h2>Hey? yo</h2>
    </>
  );
}

export default App;
```

Hello!!!React!

Hey? yo

- 조건부 연산자
  - {조건 ? 결과1 : 결과2}
  - 결과1: 참인 경우 출력
  - 결과2: 거짓인 경우 출력

```jsx
function App() {
  const name='React!'
  return (
    <>
    {name === 'React!'? (
      <h1>Hello!!!{name}</h1>
    ): (
      <h2>Hey? yo</h2>
    )}
    </>
  );
}

export default App;
```

출력 결과

Hello!!!React!

- AND연산자 &&

- 특정 조건을 만족할때 내용을 보여주고, 만족하지 않을때 아예 아무것도 렌더링 하지 않아야 하는 상황
- null을 렌더링⇒ 아무것도 나오지 않음
  - AND연산자로 같은 작업 가능

출력 결과- 빈화면 출력

```
function App() {
  const name='React!'
  return (
    <>
    {name === 'REACT'? <h1>Hello!!!{name}</h1>: null}
    </>
  );
}

export default App;
```

같은 결과- AND연산자 사용

- 조건이 참인 경우만 &&기호 뒤의 구문 실행

```
function App() {
  const name='React!'
  return (
    <>
    {name === 'REACT'&& <h1>Hello!!!{name}</h1>}
    </>
  );
}

export default App;
```

- 0은 예외적으로 화면에 나타남

출력 결과- 빈 화면이 아님

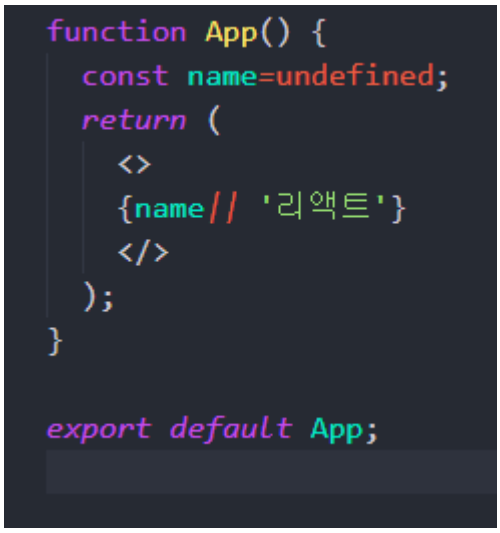
```
function App() {
  const name='0'
  return name && <h1>Hello!!!</h1>
}

export default App;
```

Hello!!!

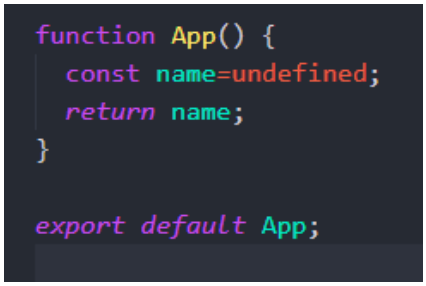
- OR연산자 ||
  - 앞의 조건값이 undefined, false, null일때 뒤의 구문 출력

출력 결과



리액트

- 함수가 undefiend를 리턴하면 에러⇒ 버전 업이되면서 에러 발생하지 않음



- 출력 결과는 빈 화면

- 태그안에 존재하지 않는 값(undefined)를 출력⇒ 빈 출력⇒ 빈화면 출력

스타일 적용

- DOM요소에 스타일을 적용할 때- 객체 형태
- 카멜 표기법 사용- ex)borderBottom
- 객체를 미리 선언

```jsx
function App() {
  const name= 'Hi! React!!';
  const style={
    fontSize: '36px',
    color: 'green',
    fontWeight: 'bold',
    padding: '10px'
  }
  return <div style={style}>{name}</div>
}

export default App;
```

출력 결과

Hi! React!!

- 바로 style값 지정

```jsx
function App() {
  const name = "Hi! React!!";

  return (
    <div
      style={{
        fontSize: "36px",
        color: "red",
        fontWeight: "bold",
        padding: "10px",
      }}
    >
      {name}
    </div>
  );
}

export default App;
```

출력 결과

Hi! React!!

- className
  - class는 JS의 예약어이므로 className을 사용
  - class로 사용해도 스타일이 적용되기는 하지만 크롬 개발자 도구의 콘솔 탭에 경고가 나타남

```
▶Warning: Invalid DOM property    react-dom.development.js:99
`class`. Did you mean `className`?
    at div
    at App
```

```jsx
import "./App.css";
function App() {
  const name = "Hi! React!!";

  return <div className="react">{name}</div>;
}

export default App;
```

출력 결과

Hi! React!!

- 태그는 꼭 닫아야함
  - 닫지않으면 오류 발생

```jsx
import "./App.css";
function App() {
  const name = "Hi! React!!";

  return <div class="react">
    {name}
    <br>
      </div>;
}

export default App;
```

```
Module build failed (from ./node_modules/babel-loader/lib/index.js):
SyntaxError: C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\src\App.js: Unterminated JSX contents. (7:18)

   5 |   return <div class="react">
   6 |     {name}
> 7 |     <br>    </div>;
     |                 ^
   8 | }
   9 |
  10 | export default App;
    at instantiate (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:72:32)
    at constructor (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:358:12)
    at Object.raise (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:3335:19)
    at Object.jsxReadToken (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:7627:20)
    at Object.getTokenFromCode (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:8047:19)
    at Object.getTokenFromCode (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:6181:20)
    at Object.nextToken (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:2019:10)
    at Object.next (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:1923:10)
    at Object.eat (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:1928:12)
    at Object.expect (C:\Users\-\Desktop\Se-young\Studynote\05-ReactJS\react-book\node_modules\@babel\parser\lib\index.js:4002:10)


ERROR


src\App.js
  Line 7:18:  Parsing error: Unterminated JSX contents. (7:18)
```

- self-closing태그: 태그사이에 별도의 내용이 들어가지 않은 경우
  - 태그를 선언하면서 동시에 닫을수 있는 태그

```
import "./App.css";
function App() {
  const name = "Hi! React!!";

  return <div class="react">
    {name}
    <input></input>
        </div>;
}

export default App;
```

출력 결과

주석

- 기본 주석- {/*내용*/}
- 시작 태그 안 주석- //내용

출력 결과

```jsx
import "./App.css";
function App() {
  const name = "Hi! React!!";

  return (
    <div class="react">
      {/*태그 밖 주석 */}
      {name}
      <div //시작태그 안에서의 주석
      >
      </div>
      //과 /** */은 그대로 노출
    </div>
  );
}

export default App;
```

```
Hi! React!!
//과 /** */은 그대로 노출
```