2-(1) 가상 DOM과 JSX 구문

가상 DOM

index.tsx 살펴보기

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

const root = ReactDOM.createRoot(

  document.getElementById('root') as HTMLElement
);
root.render(
  //오류 메시지를 보여주는 컴포넌트
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals(); //성능 측정 기능으로, 리액트 개발과는 직접 관련이 없음!
```

react와 react-dom 패키지

- 리액트 프로젝트는 항상 **react**와 **react-dom** 패키지가 필요함
- react는 리액트 앱이 동작하는 환경과 무관하게 **공통**으로 사용하는 기능을 제공
- **렌더러(renderer)** : 앱이 동작하는 환경에 종속적인 기능을 제공하는 데 특화된 패키지
    - react-dom/client, react-dom/server, react-native …
- **CRS**(Clien-side rendering) 방식으로 동작하는 앱 : react, react-dom/client 조합
- **SSR**(Server-side rendering) 방식으로 동작하는 앱 : react, react-dom/server 조합
- **모바일 앱** : react, react-native 조합

- **react와 렌더러 패키지의 경계에 가상 DOM 이라는 매커니즘이 존재**

XML 마크업 언어

- 문서(document) : 마크업 언어로 작성한 텍스트가 담긴 파일이나 인터넷 망을 통해 전송되는 스트림(stream)
- 마크업 언어(markup language) : 쉽게 작성할 수 있고, 컴퓨터도 쉽게 의미를 파악할 수 있는 특수한 형태의 언어
- HTML 문서
  - 요소(element) : body, head, html, div, meta…
  - 구조 : 트리구조(계단구조) , 부모-자식 요소 존재

문서 객체 모델

- **문서 객체 모델(Document Objcet Model; DOM)** : 웹 브라우저가 HTML 형식의 문자열을 화면에 출력할 때, 문자열을 분석하여 **특별한 형식의 자바 스크립트 객체 조합**으로 바꾸는데, 이 자바 스크립트 객체는 모두 자신의 **특징에 맞는 인터페이스**를 구현한다. 이 **인터페이스의 총칭**이 DOM이다.
- **브라우저 객체 모델(Browser Objcet Model; BOM)** : 웹 브라우저 js 엔진은 window라는 전역변수를 제공함. 웹 브라우저의 특정 웹 페이지를 의미하는 객체
- HTML 문서의 html 요소는 head와 body태그를 1개씩만 가질 수 있음
- 웹 브라우저는 dom의 다양한 인터페이스를 각각의 목적에 맞게 구현한 객체로 생성할 수 있도록 **document.createElement** 메소드를 제공

```
let newDiv = document.createElement("div) //div 요소를 자바스크립트로 생성하는 예
```

HTMLElement

- HTMLElement : 모든 종류의 HTML 요소가 구현하는 인터페이스
- 인터페이스 형태 : HTML요소명Element
- 위의 코드에서 newDiv 객체의 type ⇒ HTMLDivElement

- HTMLElement는 EventTarget, Node, Element 인터페이스를 상속받음
- Node
  - appendChilde 메소드 제공
  - HTMLElement는 모든 HTML 태그의 부모 인터페이스
  - 모든 HTML 태그는 appendChild 메소드를 가짐
- js로 HTML 요소를 웹 페이지에 나타나게 하려면 **createElement(HTML DOM 요소 객체 생성)**와 **appendChild(DOM 객체 웹 브라우저 화면에 출력-렌더링)**를 거쳐야 함

```
let p = document.createElement("p) //p요소 생성
//<p>요소를 <body>의 마지막 자식 요소로 추가
document.body.appendChild(p)
```

리액트를 사용하는 프론트엔드 개발(가상 DOM) //재작성

- document.createElement와 유사하게 react 패키지는 **createElement** 함수를 제공

```
const p = React.createElement('p') //p 요소 생성
const pVirtualDOM = React.createElement('p',null,'Hello World!') //'hello world!' 문자열은
//유효한 html 요소이므로, 문자열은 <p> 요소의 자식 요소가 될 수 있음
const root = ReactDOM.createRoot(document.getElementByID('root') as HTMLElement)
root.render(pVirtualDOM) // 렌더링!!!!!!!
```

- 2행 pVirtualDOM ⇒ 생성만 되고 아직 가상 DOM 트리에 추가되지 않음
- **root.render** 메소드 ⇒ **가상 DOM을 물리 DOM으로 전환**해 주는 기능을 함
- **document.getElementByID** 메소드 ⇒ 이미 생성된 특정 **물리 DOM 객체를 찾아주는 역할**, 위의 코드에서는 id 속성값이 ‘root’인 DOM 객체를 찾아냄

```
const root = ReactDOM.createRoot(document.getElementByID('root') as HTMLElement)
const pVirtualDOM = React.createElement('p', null, 'Hello virtual DOM world!')

root.render(pVirtualDOM)
```

- 1,2행의 순서가 바뀜 ⇒ 앞선 코드는 root DOM 객체를 찾은 후에 가상 DOM 트리를 물리 DOM 트리로 변환 하였지만, 위의 코드는 HTML 요소를 가장 DOM 트리 구조로 구현한 후, render 메소드가 호출되는 순간 가상 DOM 트리를 물리 DOM 트리로 변환해 줌.

- 이전 코드는 이미 존재하는 물리 DOM 트리에 특정 HTML 요소의 속성값을 바꿔야 할 때 문제가 생김.

JSX 구문 이해하기

React.createElement 호출의 복잡성 문제

- React.createElement는 가상 DOM 객체를 만들어 주는 함수이지만, HTML 요소가 부모/자식 관계로 구성되면 코드가 지나치게 복잡해지는 문제가 있음.
- 이러한 복잡성을 해결하기 위해 js 언어에 없는 JSX 기능을 언어 확장 형태로 추가하였음.

JSX = JavaScript + XML

- XML 구문에 자바스크립트 코드를 결합하는 용도
- XML 용어
  - 와 같은 시작 태그, 와 같은 끝 태그
  - 시작 태그에는 id, style과 같은 속성을 기술, 속성값은 ‘ ‘ 혹은 “ “로 감싸줌
  - 시작태그와 끝태그 사이에 <h1>hello<h1>과 같은 자식 요소 삽입 가능
  - 자식 요소가 없으면 <요소명/>과 같이 표현 ⇒ **스스로 닫는 태그**
- XML 규약을 엄격하게 준수해야 함
- { } 의미
  - 자바스크립트 코드를 감싸는 형태의 문법을 제공
  - 자바스크립트 변수에 들어있는 값을 XML 구문 안에 표현 가능
  - {hello}
  - 중괄호 안에는 return 키워드 없이 값만을 반환해야 함! ⇒ **표현식**

배열과 JSX 구문

- JSX 구문의 목적 : React.createElement 함수 호출을 간결하게 하기 위함

- 반환값 : 가상 DOM 객체 ⇒ 변수나 배열에 담을 수 있음!
- 주의사항 : 변수가 부모 컴포넌트 없이 존재할 수 없음. <부모>{변수}</부모> 형태 준수!

컴포넌트 이해하기

컴포넌트

- 객체지향 언어의 원조인 스몰토그에서 유래
- 화면 UI를 처리하는 클래스
- 모델-뷰-컨트롤러(Model-View-Controller; MVC)설계 지침에 따라 구현된 클래스여야 함
    - 모델 : 앱의 데이터 부분
    - 뷰 : 화면에 렌더링 하는 부분
    - 컨트롤러 : 사용자의 키보드와 마우스 입력을 수신받아 모델과 뷰에 적절한 형태로 반영하는 역할을 하는 부분

리액트 컴포넌트와 사용자 컴포넌트

- **리액트 제공 컴포넌트(리액트 컴포넌트)**
    - div, h1과 같이 첫 글자를 소문자로 시작
    - HTML5의 각 태그에 대응하는 리액트 컴포넌트를 제공
    - 리액트가 제공하는 컴포넌트를 React.createElement로 생성할 때는 컴포넌트 타입에 ‘h1’과 같은 문자열을 입력해야 함 ⇒ **컴포넌트 이름을 일일이 임포트하지 않기 위해**
- **사용자 정의 컴포넌트(사용자 컴포넌트)**
    - MyComponent와 같이 첫 글자가 대문자로 시작
    - 함수 방식 컴포넌트 타입은 FunctionComponent<P>
    - 클래스 방식 컴포넌트의 타입은 ComponentClass<P>

- React.createElement 호출이나 JSX 문으로 생성하는 가상 DOM 생성 코드를 사용자 컴포넌트 쪽으로 이동하여 **코드를 간결하게 하기 위해 사용**

클래스 컴포넌트 만들기

- 클래스 컴포넌트는 반드시 react 패키지가 제공하는 Component 클래스를 상속해야함
- Component를 상속한 클래스 컴포넌트는 render라는 이름의 메소드를 포함해야 함
- render 메소드는 null이나 React.createElement 호출로 얻은 반환 값, JSX 문으로 가상 DOM 객체를 반환해야함
- null ⇒ 반환할 가상 DOM 객체가 없다!

```
import {Component} from 'react'

export default class ClassComponent extends Component{
  render() { return null }
}
```

```
import {Component} from 'react'

export default class App extends Component{
  rener(){
    const isLoading = true
    if(isLoading) return <p>loading...</p>

    return(
      <ul>
        <li>
          <a href="http://www.google.com">
            <p>go to google</p>
          </a>
        </li>
      </ul>
      )
    }
  }
```

- 타입스크립트와 JSX 구문을 함께 쓸 수 있게 하는 것 ⇒ 사용자 컴포넌트를 제작하는 이유
- const~~</p>

- **단축평가 형태로 구현 가능** ⇒ JSX 구문 분석기는 undefined나 null인 문장은 그냥 무시함

```
return(
  <div>
    {isLoading && <p>loading...</p>
    {!isLoading && <ul>{children}</ul>
  </div>
)
```

- **두 방법을 합쳐** isLoading 값에 따라 분기하는 JSX문 2개를 children과 같은 **변수에 담아 해결** 가능

```
import {Component} from 'react'

export default class App extends Component{
  rener(){
    const isLoading = true
    const children = isLoading ? (
      <p>loading...</p?
    ) : (
      <ul>
        <li>
          <a href="http://www.google.com">
            <p>go to google</p>
          </a>
        </li>
      </ul>
      )
      return <div>{children}</div>
    }
  }
```

속성

- 속성(property) : 부모 컴포넌트에서 자식 컴포넌트 쪽으로 전달되는 객체 타입의 데이터
- 리액트 컴포넌트 관점에서의 속성 : 값이 변하면 해당 컴포넌트를 다시 렌더링하여 수정 값을 화면에 반영
- JSX는 XML 이므로 모든 속성은 작은 따옴표로 감싸야 함. 모두 string
- 문자열이 아닌 것은 중괄호로 감싸야 함

```
<Person anme='Jack'/>
<Person name='Jack' age={22} />
<Person person = {{name: 'Jack', age : 32}} /> //안쪽 {}는 객체 생성, 바깥은 JSX 구문
```