#4 RN 들어가기전에 React사전지식(어려움)

안녕하세요 열어분

지난 자료에서 바뀐 JS에 대해서 보고 오셨습니당

뭘 자꾸 RN이나 알려주지 사전지식을 풀고 있나 하시겠지만...

나중에 편하기 위해 지금 좀 고생한다고 생각해주세여 ㅎㅎ

그럼 바로 시작합시다

참 아래의 내용이 너무 어려워서 한번에 이해가 안되더라도
한번 두번 읽으시면서 아~ 이런게 있구나 해주셔도 나중에 많은 도움이 될거에요

React

페북에서 개발한 인기있는 **프론트엔드 Library**

▼ 프론트엔드 Library란?

web application에서 **UI를 나타내는 부분을 관리**해주기 위해 사용해주며,
이는 플젝의 규모가 커지고 다양한 유저와의 상호작용이 있을때 유용하다

=⇒ 고로 기능개발에 집중할수 있다.

▼ react의 가장 기본적인 IDEA

model의 변경사항(**데이터의 변경사항**)에 대해

View(HTML파일) 를 **수정하는것이 아닌 새로 만들어 버리는것**

그리고 이때 사용하는 것이 Virtual DOM(DOM HTML tag들이라 생각해줍시다) 이다

즉 Virtual DOM에 새로 렌더링 해주고

기존의 것과 비교해 필요한 곳만 빠르게 Update를 해준다

> React and the Virtual DOM
> https://www.youtube.com/watch?v=muc2ZF0QIO4](https://www.youtube.com/watch?v=muc2ZF0QIO4)

React의 특징

- 컴포넌트라는 개념을 바탕으로 조립해서 사용할수 있게 합니다.
- 생태계가 넓고 사용하는 곳도 많습니다.

💡 뜨든 우리가 앞으로 react-native를 사용함에 있어 새로운 표현식(?)이 나타났습니다

JSX (JavaScriptXml)

```
import React from 'react';   //JSX를 사용하기 위해서는 꼭 React를 import해줘야 합니다

const element = <h1>Hello, world!</h1>;
```

위의 코드를 보면 알수 있듯 HTML처럼 보이는 JS라고 생각해주시면 편합니다.

JSX는 react의 **element**를 생성하는데요

이로써 우리는 react의 개발이 쉬워집니다.

JSX 작성규칙

1. **태그는 꼭 닫혀야 합니다(`<div>,</div>`)**
2. **두개이상의 element는 하나의 element로 감싸져야합니다.**
   (로 감싸기 애매하면 <Fragment>로 감싸자)

   ▼ e.g

   ```
   <Fragment> <div> element1 </div> <div> element2 </div> </Fragment>
   ```

3. **JSX 안에서 JS값 사용하기**

```
class App extends React.Component {
  render() {
    const name = 'likelion';
    return (
      <h1>
        hello {name}!
      </h1>
    );
  }
}
```

4. **JSX내에서 style과 CSS 클래스 설정**
   - **style 설정방법**

```
class App extends React.Component {
  render() {

    const style = {
      color: 'black',
    };

    return (
      <div style={style}>
           Here
      </div>
    );
  }
}
```

   - **react 컴포넌트에서는 class 대신 className을 사용합니다.**

```
<div className="App">
       Likelion
</div>
```

▼ 왜 className을 쓰나요

ReactDOM(JSX에서 적는 HTML태그라고 생각합시다)**은 HTML속성 대신 camelCase를 씁니다**

즉 JSX에서 class는 className과 같은 이름으로 바꿔 적습니다.

즉 위와같이 **JSX로 표현되는것**(return되는 것)이 원래는 React 객체인데 **이를 React Element라고 합니다**.

React Element

Element는 **React 앱의 가장 작은 단위**입니다. 그리고 **component의 구성요소입니다**

```
class App extends React.Component {
  render() {
    const name = 'likelion';
    return (
      <h1>                    //  <h1>태그로 감싸진 부분이 element입니다.
        hello {name}!         //
      </h1>                   //
    );
  }
}
```

React DOM은 React element와 일치하도록 우리의 DOM을 업데이트합니다.

(앞에서 말했듯 DOM은 쉽게 HTML 태그라고 생각합시다.)

React DOM은 **필요한 경우(React element와 우리의 DOM이 불일치)에만 DOM을 업데이트**합니다
~~따라서 비동기적으로 처리가 된다 할 수 있습니다.~~

▼ 비동기적인 처리란?

위의 facebook nav에서 보면 알림이 오면 아이콘의 상태만 변화하게 됩니다

전체적인 page를 reload 하는 것이 아닌 특정 부분만 reload해주는 방식을

비동기적인 처리방식이라고 생각해주시면 됩니다.

Component

- 컴포넌트의 이름은 **항상 대문자로 시작**해야합니다.
- UI를 재사용가능한 Component로 분리 작성함으로 **유지보수에 용이**하도록 합시다.

Component 생성 방법

- **function**

```
const App = ({ name }) => {
  return (
    <div>
      hello {name}!
    </div>
  );
};
```

  - 간단하게 component를

    작성해 줄 수 있습니다.

    (props 받아와서 보여주는 것에 특화)

- **Class**

```
class App extends React.Component {
  render() {
    const name = 'likelion';
    return (
      <h1>
        hello {name}!
      </h1>
    );
  }
}
```

위의 식에서는 App이라는 Component를 class로 만들어 줬습니다.

  - **Component에는 render함수가**

    있어야합니다.

  - 내부에 **JSX를 return** 해줘야 함다
  - **State와 Life Cycle**을 가질수 있슴다

Props (변경 불가 값)

Props 사용 예시

```
//function component
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

//class component
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

//rendering예시
const element = <Welcome name="Sara" />;   //Sara라는 props를 Welcome component에 전달
ReactDOM.render(
  element,
  document.getElementById('root')          //root id 를 가진 HTML tag 에 element를 render
);
```

React 가 사용자 정의 **component(Welcome)** 로 작성한 **element**를 발견하면

JSX 속성(name = "Sara")을 해당 **component**(Welcome)에 전달합니다.

그럼 그 전달되는 값(name = "Sara")을 **Props**라 합니다.

🔥 Props 는 변경할 수 없습니다!!!

🔥 따라서 **수정하면 안되는 props대신**
state를 통해서 동적이고 변경가능한 action을 가능하게 합니다.

▼ function component와 Class component의 차이 (TMI)

지속적으로 update되는 UI를 구현하기 위해서는 function 차원에서는

ReactDOM.render함수를 지속적으로 실행시켰어야 했습니다(그래야 계속 변하니까요!)

Class로 구현해주면 method는 update되지만(실행되지만) 같은

React.DOM을 렌더링 하는경우 하나의 instance만 생성해서

그 안에서 lifecycle을 사용할수 있게 됩니다. 자세한 내용은 밑에서 확인하시죠!

State (변경 가능값) 그리고 lifecycle

컴포넌트 내부에서 선언(비공개적으로)하며 내부에서 값변경 가능

아래 예시코드의 1 ~ 6 까지의 과정을 보면 state가 어떻게 변하고 사용될 수 있는지 나와있습니다.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }    // 1.Clock을 호출하면 constructor가 실행되고 state가 지정된다

  componentDidMount() {   // 3. Mount이후에 실행되는 method
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );  // 4. 매초(1000ms)마다 tick함수를 실행한다
  }

  componentWillUnmount() {
    clearInterval(this.timerID); //만약 DOM으로 부터 Clock이 삭제된적 있으면 시계를 멈춘다.
  }

  tick() {
    this.setState({
      date: new Date()
    });   // 5. tick함수는 setState로 state의 값을 변경하고
  }

  render() {
```

```
    return (
      <div>
        <h1>Hello, world!</h1>   // 6. react는 setState덕분에 state가 변경된것을 감지해
                                 // 다시 render를 호출하고 this.state가 변하면 ReactDOM 업뎃
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(   // 7. Clock이 다시 mount되면 3번이 다시 실행됨
  <Clock />,       // 2. Clock이 DOM에 render되는것을 mount라 한다
  document.getElementById('root')
);
```

1. state 선언
2. ReactDOM에 mount
3. `componentDidMount()` 실행
4. 특정 method내 `setState`를 통해 state변화
5. `setState`로 변화를 감지한 react는 `render`를 호출해 `this.state`상태변화
6. element의 상태가 변했기 때문에 ReactDOM 다시 `render`
7. ReactDOM이 render(mount) 되었으니 `componentDidMount()` 실행 ....

위의 과정을 **component내부적으로 반복**할수 있게 됩니다.

따라서 **변화하는 state의 값을 component는 사용할 수 있게 되는 것**입니다.

▼ state사용시 주의 사항

```
this.setState({comment: 'Hello'});
```

이렇게 `setState`를 사용하는 경우에만 react가 인지하여 다시 rendering되는 것입니다.

```
this.state.comment = 'Hello';
```

위의 경우는 `constructor`에만 사용할수 있어요!

▼ state는 자식 component에게 state를 props로 전달할수 있습니다.

```
<FormattedDate date={this.state.date} />
```

😂 다소 어려운내용인데 끝까지 읽어주셔서 감사합니다 ㅠㅠ