2021년 11월 19일 금요일

03. 리액트 컴포넌트

---

03. study-component

- 개발 하고자 하는 최상위 폴더에서 react 폴더를 만든다.
  - react 폴더로 이동 후
  - create-react-app study-component
  - 오류가 난다면
  - npx create-react-app study-component --use-npm
    - 이제 리액트 컴포넌트 강의는 이곳에서 실습 합니다.

2021년 11월 19일 금요일

03. 리액트 컴포넌트

03-1. 컴포넌트를 표현하는 JSX

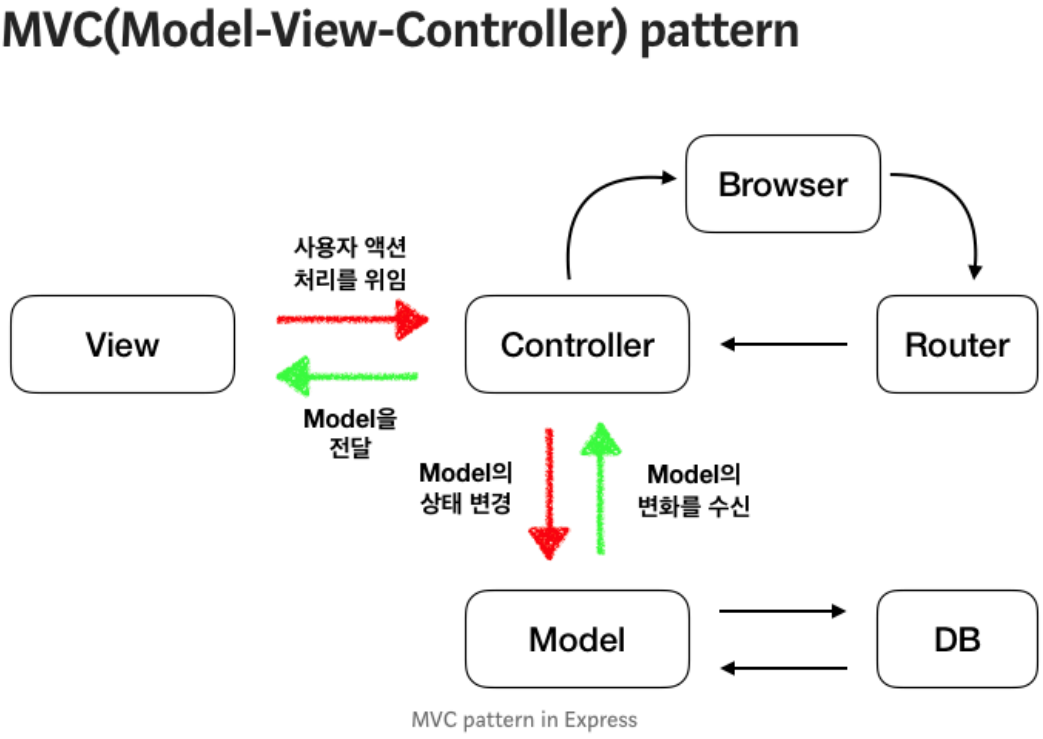
- react/study_component
  - cd src
  - mkdir component
- 기존 HTML 작성하는 방법에 대해 먼저 알아보자.
  - react/study_component/src/index.js
  - 해당 파일이 제일 먼저 렌더링이 되므로 해당 파일에 html 을 작성하는 기본 문법을 작성한다.
  - 그리고 JSX 문법을 이용하면 더 효율적인 코딩이 되는 것을 이해한다.
- JSX
  - Javascript XML 의 줄임말
  - 자바스크립트에 XML 을 추가한 확장형 문법
  - XML 또한 HTML의 표현법을 확장한 문법
  - 기존 자바스크립트는 HTML과 자바스크립트 분리해서 작성
  - 이제는 그럴필요 없다.
  - App.js 에 붙여 넣어라
  - component/C01_JSXSample.js
- JSX와 기존 개발 방법의 차이
  - index.js와 C01_JSXSample.js 비교해보면 JSX 이용시 더 편리하게 개발이 가능하다.
- Jsx 작동원리
  - 리액트 엔진은 JSX의 XML 구조를 분석하여 자바스크립트 함수 코드를 반환한다.

2021년 11월 19일 금요일

. 리액트 컴포넌트

03-2.컴포넌트와 구성 요소

- 컴포넌트 : 리액트의 꽃. 시작과 끝
  - 기존의 웹 프레임워크 방식 MVC
    - Model : 정보 담당 데이터, setter, getter
    - View : 화면 담당
    - Controller : 구동 담당 서비스
  - 장점 : 소스 관리가 용이 개발하기 편함
  - 단점 : 각 요소마다 의존성이 높아 화면 하나를 바꾸기가 쉽지 않음
- 하지만 컴포넌트는 MVC의 view 부분을 독립적으로 구성하여 재사용 할 수 있게함.
  - 컴포는트를 통해 새로운 컴포넌트를 쉽게 만들 수 있다.



03. 리액트 컴포넌트

Today Plan 컴포넌트 만들기

- react/study_component
  - /src/component/C02_TodayPlan.js
  - .js 만든 후 roc 입력 후 Tab 하면 파일 이름에 맞게 클래스형 컴포넌트가 자동생성
  - 컴포너는트는 첫글자는 무조건 대문자로 해야함.
- App.js 컴포넌트에 TodayPlan.js 추가하기

```
// import logo from './logo.svg';
import './App.css';
import TodayPlan from './03/TodayPlan';

function App() {
```

- Import 문에 .js 생략해도 되나요?
  - Create-react-app 은 js 또는 jsx 파일의 확장자를 생략해도 해당 파일을 자동으로 찾을 수 있게 설정 되어 있음.
  - 웹팩 코드 검색 확장자 (webpack module resolution) 기능
    - 확장자가 파일 이름에 있는 파일을 먼저 import
    - 확장자가 파일 이름에 없는 경우 웹팩의 확장자 옵션(extentions)에 정의된 확장자 목록을 보고 해당 확장자 이름을 포함한 파일이 있는지 확인
    - Ex) import MyFile 이라면 MyFile.js > MyFile.jsx 순으로 탐색

03. 리액트 컴포넌트

03-3 컴포넌트에 데이터를 전달하는 프로퍼티

- react/study_component
  - /src/component/C03_MyComponent.js
- 프로퍼티(property)는 상위 컴포넌트가 하위 컴포넌트에 값을 전달할 때 사용
  - 프로퍼티 또는 props 로 불른다. 문서상에선 프로퍼티로 명명 하지만 실제 코딩할땐 props 로 씀
  - 프로퍼티의 값은 수정 할 수 없다.
- App 컴포넌트 -> MyComponent 에 데이터 전달
  - 다같이 작성
    - App.js 컴포넌트에서 name 이라는 이름의 프로퍼티를 보냄
    - MyComponent 컴포넌트의 render( ) 함수에서 받을 수 있다.
    - render( ) 함수는 props.name 으로 프로퍼티 값을 참조 할수 있다.
    - 여기서 가장 중요한건 단방향으로 데이터가 흐른다.
    - 상위 -> 하위 App -> MyComponent
- prop-types
  - /src/component/C04_PropsTypeComponent.js
  - 컴포넌트의 프로퍼티에 대해 자료형(String, Number …)을 선언하는 방법
  - PropsComponent.js
- 다양한 프로퍼티 사용하기
  - /src/component/C05_PropsType_1.js
  - 문자열(String) 에 대해서는 “ 큰따옴표 또는 ‘ 작은 따옴표를 사용
  - Boolean형 또는 number 인 경우 { } 중괄호를 사용
- Boolean 형 던지기
  - /src/component/C06_PropsType_2.js
  - 프로퍼티 명만 명시해도 true 넘어간다

  - 프로퍼티 명을 명시하지 않으면 false가 아닌 undefined로 넘어간다

- 객체형 프로퍼티
  - /src/component/C07_PropsType_3.js
  - 다양한 값을 객체에 담아 보낼수 있다.
  - 객체에 PropTypes.shape 를 이용하여 객체 프로퍼티를 정의하면 객체 목록을 한눈에 확인 할 수 있다.

- 필수 프로포터 사용하기
  - /src/component/C08_PropsType_4.js
  - 특정 컴포넌트에 꼭 전달되어야 하는 프로퍼티가 있다면 해당 프로퍼티를 필수 프로퍼티로 지정하면 된다.

- 프로퍼티에 기본값 지정하기
  - /src/component/C09_PropsType_5.js
  - defaultPorps 값을 지정한다.
  - 기본값을 지정하면 값을 던지지않아도 해당 기본값으로 처리된다.

- 자식 프로퍼티 사용하기
  - /src/component/C10_PropsType_6.js
  - <button>버튼</button>
  - 이런식으로 중간에 노드를 배치하는 표현식이 있다면?

03. 리액트 컴포넌트

03-4 컴포넌트 상태 관리하기

- react/study_component/src/component/C11_StateExample.js
- state로 상태 관리하기
  - 프로퍼티는 상태를 바꿀수 없음.
  - 프로퍼티를 바꾸기 위해선 state를 이용한다.
  - state는 '값을 저장하거나 변경할 수 있는 객체'
  - 버튼을 클릭하거나 값을 입력하는 등의 이벤트와 함께 사용.

- state를 사용할 때 주의할 점
  - 생성자(constructor)에서 반드시 초기화 해야 함
  - State 값을 변경할 때는 setState() 함수 (상태관리함수) 를 반드시 사용
  - setState() 함수는 비동기로 처리
  - setState() 코드 이후로 연결된 함수들의 실행이 완료된 시점에 화면 동기화 과정 진행
- **State 값은 반드시 setState()로 변경**
  - **State 값을 찍접 변경한 경우**
  - **Render() 함수로 화면을 그려주는 시점을 리액트에서 인지하지 못함**
  - **setState() 메소드로 값을 변경해야 리액트에서 인지하고 render() 함수로 화면을 그림**
- **setState()로 함수의 인자로 함수를 전달하면 이전 state 값을 쉽게 알수있다.**
-

03. 리액트 컴포넌트

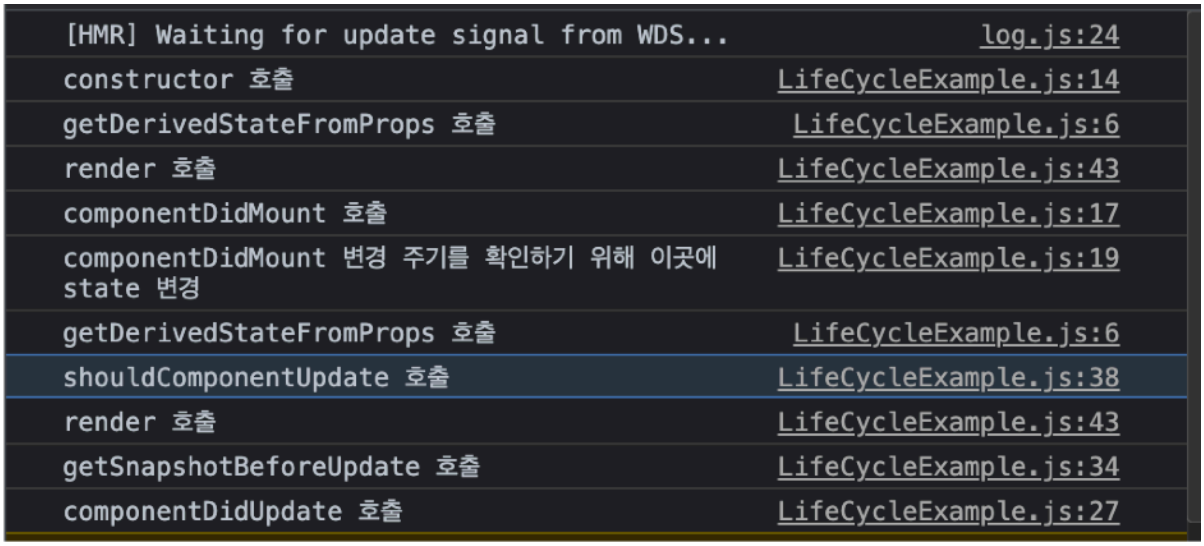
03-5 컴포넌트 생명주기

- react/study_component/src/component/C12_LifeCycleEx.js
- 컴포넌트의 생성부터 소멸까지를 생명주기(lifeCycle) 이라고 한다.
- render() 함수를 포함하여 총 8개 함수가 있다.
- 개발자가 마음대로 호출하는게 아니다.
- 생성부터 생성 완료까지
  - constructor(props)
    - 생성자 함수
    - 맨처음 한번 호출
    - 항상 super() 함수를 가장 맨위에 호출
    - super(). 함수에는 프로퍼티와 생명 주기 상태 등을 초기화 하는 과정이 포함
  - getDerivedStateFromProp
    - Static getDerivedStateFromProps(props, state)
    - 정적함수
    - 함수에서 this.props 이나 this.state 와 같은 방법으로 프로퍼티나 state 값에 접근 못함
    - Props 상위 컴포넌트에서 넘어온 데이터
    - State 현재 컴포넌트에서 사용하는 데이터
    - 이 함수는 상위 컴포넌트에서 넘어온 props 를 state 값을 연동할 때 주로 사용
  - render
    - 데이터가 변경되어 새화면을 그려야 할 때 자동으로 호출
      - JSX 를 화면에 뿌려준다.
  - componentDidMount
    - render() 함수가 JSX 를 화면에 그린 이후에 호출
    - 만약 모든 화면에 모든 그림이 그려진후 뭔가 하려면 여기에 하면 된다.

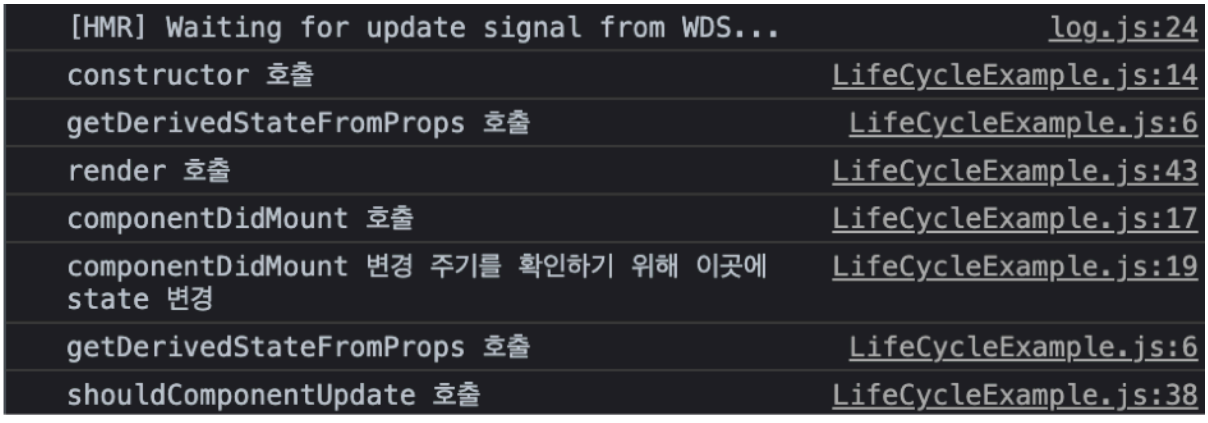
- 생성 완료 -> 갱신 완료
  - shouldComponentUpdate(nextProps, nextState)
    - 프로퍼티를 변경 하거나 setState 함수를 호출해서 state 값을 변경했을 때
    - 화면을 새로 그려줘야 하는지 판단하는 함수
    - 이 함수는 함수를 그릴지 말지 판단, state 값을 비교하는 과정이 포함
    - 이걸 쓰면 성능에 많은 영향을 준다.
  - getSnapshotBeforeUpdate(prevProps, prevState)
    - 컴포넌트의 변경된 내용이 가상 화면에 완성된 이후 호출되는 함수
    - 컴포넌트가 실제로 출력되기 전에 호출되므로 화면에 출력될 엘리먼트의 크기 또는 스크롤 위치 등의 DOM 정보에 접근할때 사용
  - componentDidUpdate(prevProps, prevState, snapshot)
    - 컴포넌트가 실제 화면에 출력된 이후 호출되는 함수
    - 부모 컴포넌트로 전달된 이전 프로퍼티(prevProps)
    - 이전 값(prevState)
    - getSnapshotBeforeUpdate 애서 반환된 snapshot 을 인자로 전달 받는다.
    - 스크롤 위치를 옮기거나 커서를 이동시키니는 등의 DOM 정보를 변경할때 사용
- 갱신완료 -> 소멸 완료
  - componentWillUnmount()
    - 컴포넌트가 소멸되기 직전에 호출되는 함수
    - 컴포넌트에서 감시하고 있는 작업들을 해제 할때 필요한 함수
    - 메모리 누수 현상을 줄이는 작업을 해야 한다.
    - setInterval 사용 한다면 해제 해야 한다.
- (LifeCycleExample.js) 파일에서
  - componentDidMount() 함수에 setState 선언하면 갱신을 확인할 수 있다.
- 생명주기 함수의 실행 과정 살펴보기
  - LifeCycleExample.js

  - 컴포넌트 생성 > constructor > getDerivedStateFromProp > render > componentDidMount > 생성완료

- 변경 과정의 생명주기 함수 실행 과정
  - 변경 과정의 생명주기 함수들은 상위 컴포넌트의 프로퍼티의 변화때 실행
  - State 의 변화가 생기면 실행
  - 보통 특별한 이벤트를 주지 않는 이상 componentDidMount()에 state 값을 변경하면 된다.
  - shouldComponentUpdate( ) 여기서 return true 인 경우 render 이후 실행



  - shouldComponentUpdate( ) 여기서 return false 인 경우



  - shouldComponentUpdate( ) return false 더라도 관계 없이 화면 동기화 과정을 진행해야 한다면
    - forceUpdate() 함수를 사용

- componentDidMount( ) 함수에 setState( ) 함수 대신 forceUpdate( ) 함수로 변경

```
[HMR] Waiting for update signal from WDS...                    log.js:2
constructor 호출                                    LifeCycleExample.js:1
getDerivedStateFromProps 호출                       LifeCycleExample.js:
render 호출                                         LifeCycleExample.js:4
componentDidMount 호출                              LifeCycleExample.js:1
componentDidMount 변경 주기를 확인하기 위해 이곳에 state 변경   LifeCycleExample.js:1
getDerivedStateFromProps 호출                       LifeCycleExample.js:
render 호출                                         LifeCycleExample.js:4
getSnapshotBeforeUpdate 호출                        LifeCycleExample.js:3
componentDidUpdate 호출                             LifeCycleExample.js:2
```

  - 강제로 실행되는걸 확인 할 수있다.

- 생성완료 > getDerivedStateFromProps > shouldComponentUpdate >
- Render >
- getSnapshotBeforeUdpate > componentDidUpdate > 갱신 완료

- 소멸 과정의 생명주기 함수들 실행해보기
  - 소멸과정은 컴포넌트가 화면에서 생략되면 실행됨
  - App.js 를 다음과 같이 바꾸보자
    - react/study_component/src/App_destroy.js
    - 삼항 연산자를 통해 사라지게 하면 소멸 연산자가 호출된다.

```
[HMR] Waiting for update signal from WDS...                    log.js:24
constructor 호출                                    LifeCycleExample.js:14
getDerivedStateFromProps 호출                        LifeCycleExample.js:6
render 호출                                         LifeCycleExample.js:47
componentDidMount 호출                              LifeCycleExample.js:17
componentDidMount 변경 주기를 확인하기 위해 이곳에 state 변경   LifeCycleExample.js:19
componentWillUnmount 호출                           LifeCycleExample.js:43
```

- 생성완료 or 갱신완료 > componentWillUnmount > 소멸완료

- 카운터 프로그램 만들며 생명주기 함수 사용해보기
  - react/study_component/src/component/C13_Counter.js
    - 가장 일반적인 counter 예제
  - NewCounter.js
    - 상위 컴포넌트로 부터 count 정보를 받기 위해
    - getDerivedStateFromProps() 선언 및 동기화
  - App_count.js
    - 상위 컴포넌트에서 +10 하면 NewCounter.js 만 동기화 처리된다.
  -

03. 리액트 컴포넌트

03-6 클래스형 컴포넌트

- Component Class
  - 프로퍼티
  - State
  - 생명주기 함수가 들어있는 구조에 대해서 만든다.
- PureComponent Class
  - 예로 1000개의 컴포넌트가 존재함.
  - 이때 999개는 다 사용 완료.
  - 1개만 남은 상황에서 1개를 사용할때 상위 Component에서 프로퍼티 변경시
  - 999개에 대해 다시 그리는 현상 발생. (1개만 변경해도 되는데!!)
  - 이를 위해 PureComponent의 얕은 비교를 통해 이런 현상을 없앨 수 있다.
- App_purecomponent.js
  - React.PureComponet 예제
    -

03. 리액트 컴포넌트

03-7 함수형 컴포넌트

- Stateless Functional Component
  - state가 없는 함수형 컴포넌트
  - state를 포함하지 않는 컴포넌트
  - 데이터를 있는 그대로 받아 출력할때 사용
  - SFC.js

03. 리액트 컴포넌트

03-8 배열 컴포넌트

- react/study_component/src/component/C17_ListExample.js
- react/study_component/src/component/C18_TodoList.js
- 게시판, 유튜브 영상 목록의 구현은 자바스크립트 배열로 처리한다.
- 자바스크립트 배열은 다양한 자료형을 저장할 수 있다.
- Strings, Boolean, JSX, 등 다양함
  - ListExample.js
  - Todolist.js

03. 리액트 컴포넌트

03-9 컴포넌트에서 콜백 함수와 이벤트 처리하기

- 상위에서 하위 컴포넌트로 프로퍼티를 이용해서 값을 던져 준다.
- 하위에서 프로퍼티를 변경해야 한다면 어떻게 해야하나
- 하위에 프로퍼티 원본을 수정할 수 있는 함수를 하위 컴포넌트에 제공하면 된다.
- 콜백함수는 정의된 위치에서 수행되지 않고,
- 이후 특정 상황에서만 (이벤트, 다른 함수 호출) 실행되는것을 말함
- 즉, 콜백 함수를 프로퍼티로 전달하면 된다.
  - App_countcallback.js
  -