React Study 3강

목 차

2018. 11. 01

1. 리액트 클래스

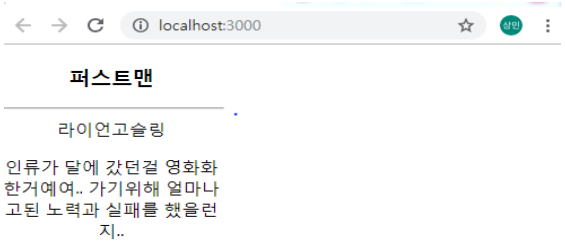
- 기존 무비카드를 클래스로 변경해서 연습 해봅니다.

- 클래스로 만들면 펑셔널컴포넌트로 만들때보다
- state, lifecycle, ref참조등을 추가적으로 사용 가능

- class 개념 역시 ES6 에 새로 도입된 요소중 하나 입니다. 모든 Component 는 React.Component 를 상속합니다.
- ES5 환경에서는 React.createClass() 라는 메소드를 사용합니다. 또한, ES5 에서 클래스를 만들때는 메소드들을 nest 할 수 없고 prototype을 사용했어야 했는데, 많이 편해졌죠.

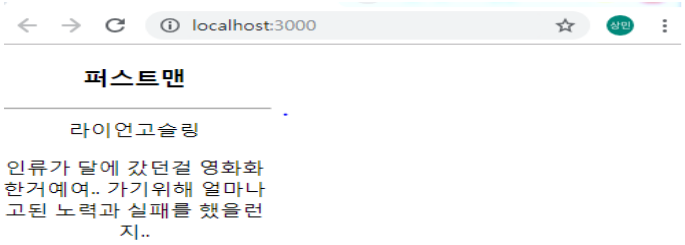
- 클래스를 한번 만들어봅니다. Npx create-react-app three_1
- app.js render부분에서 <MovieCardClass/> 호출하게 함
- src폴더에 NewFile 해서 MovieCardClass.js 생성후 렌더링 할 태그를 기록
- class 내가지정한클래스명칭 extends React.Component {

실습1 : 기존 무비카드를 클래스로 만들어 봅니다.
결과는 옆에처럼 나옵니다



1. 리액트 클래스

- 실습1. 클래스만들기
- 클래스이름은 MovieCardCls
- npx로 three_study_one 만듭니다.
- app.js 파일 아래 참조 왼쪽
- MovieCardCls 파일 아래 참조 오른쪽
- 밑에것 대로 치면 썰렁하게 나오는데
- 결과화면처럼 나오도록 수정해보시죠




```jsx
import React, { Component } from 'react';
import './App.css';
import { MovieCardCls } from './MovieCardCls';

class App extends Component {
  render() {
    return (
      <div className="App">
        <MovieCardCls />
      </div>
    );
  }

}

export default App;
```

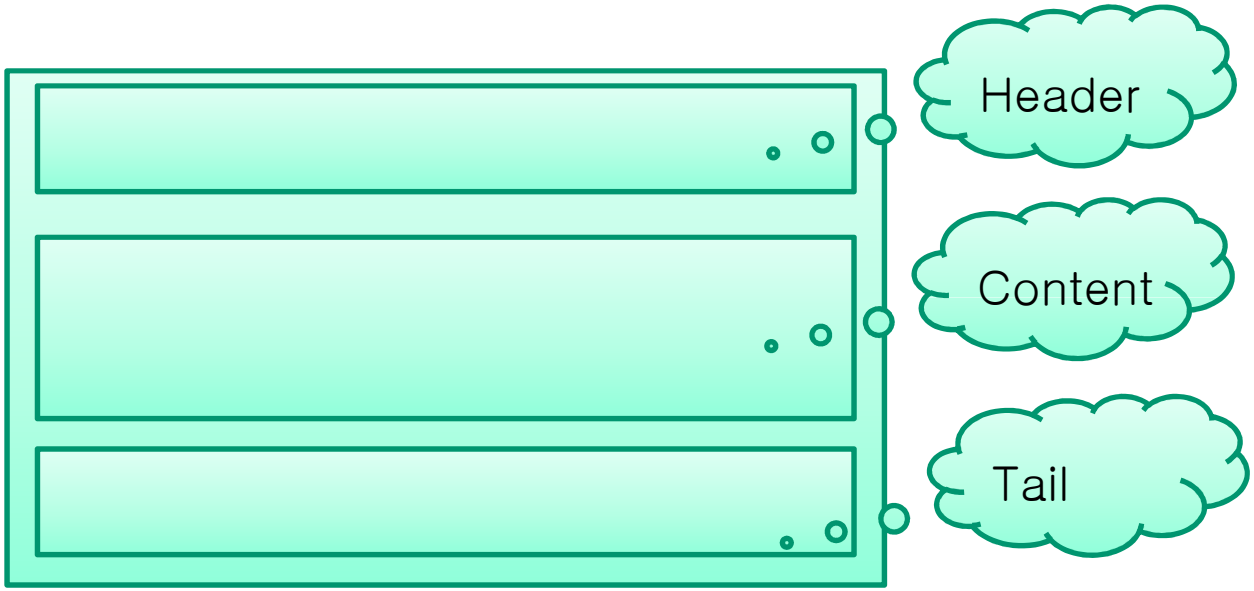
```jsx
import React from 'react'
import './MovieCardCls.css'

export class MovieCardCls extends React.Component {
    render() {
        return <div>
            <div className="myStyle">
                <h3>영화제목</h3>
                <hr/>
                <span>주연배우</span>
                <p>{this.props.children}</p>
            </div>
            </div>
    }
}
```

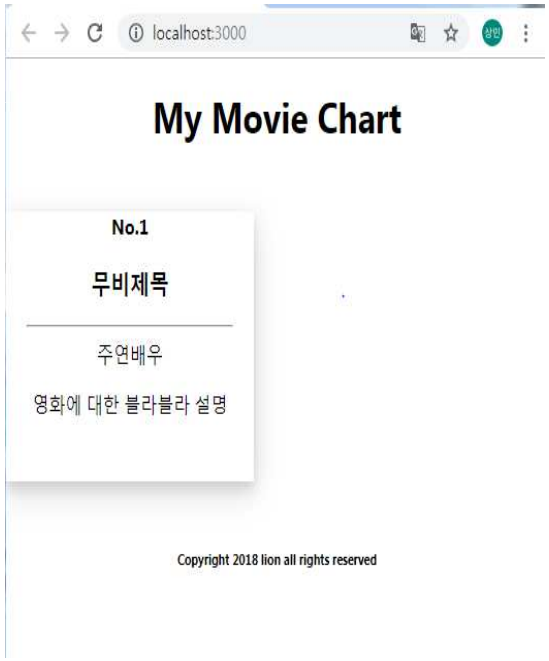
1. 리액트 클래스 – 여러 개 컴포넌트로 설계를 해볼까요

- 클래스를 중첩해서 쓸수도 있지만, 유지보수를 위해 클래스하나당 js파일 하나로관리
- 클래스별로 css 파일을 만들어 컴포넌트별 스타일링파일로 관리

- 여러 개 클래스로 컴포넌트를 만드는 연습해봅니다.




```
class MovieCard extends Component {
    render() {
        return (
            <div>
                <MovieCard100Header />
                <MovieCard200Contents />
                <MovieCard300Tail />
            </div>
        );
    }
}
```



1. 리액트 클래스 – 여러 개 컴포넌트로 설계를 해볼까요 실습2

- 오른쪽 결과처럼 나오도록 여러 개 컴포넌트로 실습합니다.
- app.js 파일 아래 참조
- 파일은 4개 만들어야죠..
- MovieCard.js, MoveCard100Header.js
- MovieCard200Header.js, MovieCard300Tail.js
- 만드신후 rcc 코드스니펫으로 간단구현 연습합니다.
- 관련자료는 슬랙에 올립니다.

```
import React, { Component } from 'react'
import './App.css'
import MovieCard from './MovieCard'


class App extends Component {
  render() {
    return (
      <div className="App">
        <MovieCard/>
        {/*  <MovieCard/>  */}
        {/*  <MovieCardCls movieTitle="퍼스트맨" />  */}
      </div>
    )
  }
}
export default App
```

2. ES6 Object – 무비카드를 배열내에 Object로 표시

- 객체란, 현실의 사물을 프로그래밍에 반영

- MovieInfo 객체를 만들어 보겠습니다.
- const MovieInfo = { 만들어질덩어리 }
- const MovieInfo = {
  movieTitle : “퍼스트맨”,
  mainActor : “라이언고슬링”
- }
- 속성(쉼표로구분됨), 키, 밸류, 메소드

- 배열안에는 들어가는 것을 item이라고 하며, item은 객체가 들어갈수도 있습니다.
- item은 쉼표로 구분됩니다,
- 배열은 const arrMovieList = []

2. ES6 Object – 무비카드를 배열내에 Object로 표시

- 실습3 참고자료
- 배열 오브젝트를 만들어 봅니다.
- 배열을 마치 디비라고 생각하고 배열서 데이터를 가져다 카드에 넣으려 합니다.
- 배열 선언은 무비카드 클래스 위에 합니다.
- 참고소스는 슬랙 참조합니다.

- 실습4는 배열 데이터를 실제 뿌려서 아래처럼 만듭니다.
- JSX내에서 map()함수를 이용합니다.

3. ES6 화살표 함수

- 3가지 화살표 함수 연습
Arrows(화살표) 함수는 => 문법을 사용하는 축약형 함수입니다.
Arrows는 표현식의 결과 값을 반환하는 표현식 본문(expression bodies)뿐만 아니라 상태 블럭 본문(statement block bodies)도 지원합니다. 하지만 일반 함수의 자신을 호출하는 객체를 가리키는 dynamic this와 달리 arrows 함수는 코드의 상위 스코프(lexical scope)를 가리키는 lexical this를 가집니다.

```
var evens = [2, 4, 6, 8,]; // Expression bodies (표현식의 결과가 반환됨)
var odds = evens.map(v => v + 1); // [3, 5, 7, 9]
var nums = evens.map((v, i) => v + i); // [2, 5, 8, 11]
var pairs = evens.map(v => ({even: v, odd: v + 1})); // [{even: 2, odd: 3}, ...]

// Statement bodies (블럭 내부를 실행만 함, 반환을 위해선 return을 명시)
nums.forEach(v => { if (v % 5 === 0) fives.push(v); });

// Lexical this // 출력결과 : Bob knows John, Brian
var bob = { _name: "Bob", _friends: ["John, Brian"], printFriends()
{ this._friends.forEach(f => console.log(this._name + " knows " + f)); } }
```

https://jsdev.kr/t/es6/2944

3. ES6 화살표 함수 – 익숙해져봅시다.

```
 const a = function () {
   return new Date()
 }
```

펑션을 생략하고 화살표로 직관적으로 표시함

```
const arrow_a = () => {
  return new Date()
}
```

리턴문만 있는 경우에는 중괄호와 리턴도 생략 가능

```
const arrow_aa = () => new Date()
```

This를 따로 바인드로 전달없이 바로 위의 this를 사용할 수 있는 장점

4. 리액트 State - concept

- State 변경되는 데이터
- 컴포넌트 내부에서 읽고 수정할 수 있는 값
- 기본값을 설정해야 함
- 업데이트를 this.setState()로 해야함 : 값을 수정한후 적당한 시점에 렌더링까지됨
- 기본값설정은 생성자를 통해 함
- 생성자는 컴포넌트가 새로 만들어질때 실행됨
- 컴포넌트에 생성자로 state를 만들어봅니다.
-

```
class MovieCard100Header extends Component {
    constructor(props) {
        super(props)
        this.state = {
            userName : '기본값',
            number : 0
        }
    }
```

- constructor()
- component를 상속했으니 부모 생성자 호출해주는게 문법
- 실제 리액트상태는 객체형태로 JSON 양식대로 표시해주어 기본값 설정 합니다.
-
-

4. 리액트 State – 상태rendering

- JSX내에서 렌더링 해봅니다.
- {this.state.username}

```
render() {
    return (
        <div>
            <h1>My Movie Chart </h1>
            <h2>{this.state.userName} 반갑습니다.</h2>
        </div>
    );
}
```

-헉 프랍하고 머가 다른거지..고민중.
- 변경되는 값이니 내가 세팅할 수 있겠군 그게 다른점입니다.
- state 업데이트는 setState()

4. 리액트 State - 상태업데이트

- State 업데이트로 해보시죠… setState()
- header에 3초후에 홍길동님 반갑습니다. 표시해보자

My Movie Chart

기본값 반갑습니다.

```
//3초후에 사용자명을 로그인한 유저명으로 변경해봅니다. setTimeout(콜백함수, 수행시간)
setTimeout(perform_3second.bind(this), 3000)

function perform_3second() {
    this.setState({userName: '홍길동'})
}
```

- 화살표함수로 해보죠..

5. 리액트 라우팅 - concept

- 다른 주소에 따라 다른 뷰를 보여주는것을 라우팅
- react-router 는, 써드파티 라이브러리
- 리액트 라우터를 설치
- npx create-react-app router-study
- cd router-test
- yarn add react-router-dom : 브라우저에서 사용되는 리액트 라우터
- index.js에 브라우저라우터 사용 준비
- import { BrowserRouter } from 'react-router-dom

- // * App 을 BrowserRouter 로 감싸기
- ReactDOM.render(
-<BrowserRouter> <App /> </BrowserRouter>, document.getElementById('root'));

-

5. 리액트 라우팅 - 기본실습

- Home.js 파일추가

src/Home.js

```
import React from 'react';

const Home = () => {
  return (
    <div>
      <h1>홈</h1>
      <p>이곳은 홈이에요. 가장 먼저 보여지는 페이지죠.</p>
    </div>
  );
};

export default Home;
```

- 특정 주소에 컴포넌트 연결하기
- <Route path="주소규칙" component={보여주고싶은 컴포넌트}>
- app.js에 <Route path="/" component={Home} />

5. 리액트 라우팅 - 두번째실습

- About.js 파일추가후 /about으로 라우팅 해봅시다.

- exact = {true}

5. 리액트 라우팅 - Link

- 다른 주소로 이동시키는 컴포넌트 <Link
- a tag의 경우 전체페이지를 이동시키므로 전체를 렌더링 하게 됨
- <a 태그 대신 <Link를 사용해서 다른주소로 이동시킴
- 이 컴포넌트는 HTML5 History API 를 사용하여
- 브라우저의 주소만 바꿀뿐 페이지를 새로 불러오지는 않습니다.
- <Link to="/about">소개</Link>

-

6. 리액트 스타일링

- 기존CSS 활용
- SCSS
- 스타일 IN JS