1. CRA 도구(1)

CRA 도구

- 리액트 프로젝트를 생성할 때 필요한 webpack 등의 설정을 자동화하여 개발 환경을 구축해주는 도구
- 생성 방법
  - npx create-react-app [프로젝트명]
  - 프로젝트 명으로 디렉토리가 만들어지고 기본적인 boilerplate 코드가 제공됨
- 기본 구조
  - react-scripts 패키지에 대부분의 구성 포함
  - Entry : src/index.js
  - 소스 코드 : src 디렉토리에 작성
  - 빌드된 아웃풋 : build 디렉토리에 저장
- 실행
  - 개발 : yarn start
  - 프로덕션 빌드 : yarn build

1

■ npm 과 yarn 비교

▪ **npm init**	**yarn init**
▪ **npm install**	**yarn**
▪ **npm install --save react**	**yarn add react**
▪ **npm uninstall --save react**	**yarn remove react**
▪ **npm install --save-dev cross-env**	**yarn add --dev cross-env**
▪ **npm update --save**	**yarn upgrade**

2. Hello React 앱 작성(1)

- **create-react-app을 이용해 boilerplate 코드 생성**
  - npx create-react-app helloapp
- **npx create-react-app hello 명령을 실행했을 때 다음 오류가 발생하는 경우가 있다.**
  - "You are running 'create-react-app' x.x.x, which is behind the latest release (x.x.x). We no longer support global installation of Create React App."
  - 이 경우에는 다음 명령어를 실행한 후 프로젝트 생성을 재시도한다.
    - npm uninstall -g create-react-app
    - npx clear-npx-cache

2. Hello React 앱 작성(2)

App 컴포넌트 작성

- src 디렉토리 아래의 파일을 index.js, index.css 파일을 제외한 파일 삭제
- src/App.js 작성
  - ES6 의 Class 기반의 컴포넌트

```
import React, { Component } from 'react';

class App extends Component {
    render() {
        return (
            <div>
                <h1>Hello World</h1>
            </div>
        );
    }
}
export default App;
```

- yarn start로 실행 후 확인
  - 개발용 서버가 구동될 때까지 약간의 시간이 걸림.

3

■ Visual Studio Code 설치 후에 reactjs code snippets 과 같은 플러그인을 설치했다면 App.js 파일을 생성한 후에 rcc라고 입력하면 class component skeleton 코드를 자동 생성할 수 있다.

■ React Component 클래스는 Component 클래스를 상속받아 작성한다.

■ render() 메서드는 JSX 구문을 사용할 수 있으며, 컴포넌트가 화면에 렌더링할 UI를 리턴한다.

■ 만들어진 컴포넌트는 다른 모듈에서 import 하여 사용할 수 있도록 export 한다.

■ 윈도우 powershell에서 실행하는 경우 보안 오류가 발생할 수 있는데 이 때에는 다음과 같이 조치한다.

- powershell을 관리자 권한으로 실행한다.
- Set-ExecutionPolicy Unrestricted 명령을 실행한다.
- y를 눌러 설정을 마무리한다.
- 다시 yarn start를 실행한다.

2. Hello React 앱 작성(3)

동적인 값의 표현

- JSX 내부에서 { } 보간법 사용하여 표현
- 중괄호 내부의 자바스크립트 식은 계산되어 출력될 수 있음
- src/App.js 변경

```
render() {
        let msg = "World";
        return (
            <div>
                <h1>Hello {msg}!!</h1>
            </div>
        );
}
```

- 중괄호 내부에 메서드의 리턴값을 출력할 수 있음.

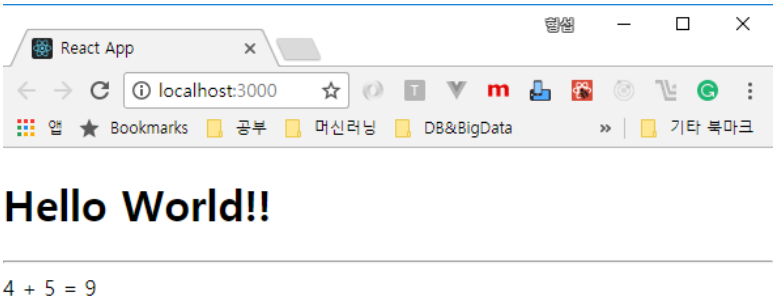
2. Hello React 앱 작성(4)

- src/App.js 다시 변경 : 메서드의 리턴값을 보간하도록...

```
import React, { Component } from 'react';

class Hello extends Component {
    createString(x,y) {
        return (
            <div>{x} + {y} = {x+y}</div>
        )
    }
    render() {
        let msg = "World";
        return (
            <div>
                <h1>Hello {msg}!!</h1>
                <hr />
                { this.createString(4,5) }
            </div>
        );
    }
}

export default Hello;
```

2. Hello React 앱 작성(5)

- 실행 결과



2. Hello React 앱 작성(6)

함수 컴포넌트의 작성

- ES6 클래스 기반의 컴포넌트
  - 다양한 생명주기 이벤트 훅을 사용할 수 있음 --> 자세한 내용은 다음 장에서
  - 함수형 컴포넌트에 비해 렌더링 속도가 느림
- 함수 컴포넌트
  - ES6 클래스 기반의 컴포넌트에 비해 최대 45% 렌더링 속도가 빠름
    - https://medium.com/missive-app/45-faster-react-functional-components-now-3509a668e69f
  - 직접 호출하여 보간법으로 렌더링할 수 있음.
  - 다양한 생명주기 이벤트 훅을 사용할 수 없음
  - 이미 1장에서 작성한 바 있음.
  - 속성을 전달받아 단순하게 렌더링하는 컴포넌트의 작성에 적당함.

```
let Hello = () => {
        return(
            <div className="container">
              <h1>Hello World</h1>
            </div>
        )
}
```

■ 함수 컴포넌트

2. Hello React 앱 작성(7)

함수 컴포넌트 (이어서)

- 간단한 함수형 컴포넌트

```
let Title = (props) => {
    return(
        <div><h2>{props.title}</h2></div>
    )
}
export default Title
```

- 함수형 컴포넌트 사용
  - JSX 마크업, 함수호출형 보간법 두가지 방법 모두 사용 가능

```
import Title from './Title';

let App = () => {
    let data = { title : '해야할 일 목록' };
    return (
        <div>{ Title(data) }</div>
    );
}
```

8

■ 함수 컴포넌트는 React Hook(6장)을 다룬 후에 본격적으로 사용함.

3. 간단한 스타일 적용(1)

css 파일을 import 할 수 있음.

- import './index.css';
- 전역 수준에서 css 참조
- src/index.css 변경

```
hr.dash-style {
   background-color: #fff;
   border-top: 2px dashed gray;
}
```

- Bootstrap 설치
  - 실행중인 개발 서버를 중지시킨 후 bootstrap 4 버전을 설치한다.
    - yarn add bootstrap@4.x.x
  - bootstrap은 반응형이며 모바일 우선인 웹프로젝트 개발을 위한 HTML, CSS, JS 프레임워크

9

■ bootstrap은 4.x.x 버전을 사용한다. 명시적으로 bootstrap의 특정 버전을 설치하려면 yarn add bootstrap@4.x.x와 같이 설치한다.

3. 간단한 스타일 적용(2)

- bootstrap css를 사용하도록 src/index.js 변경

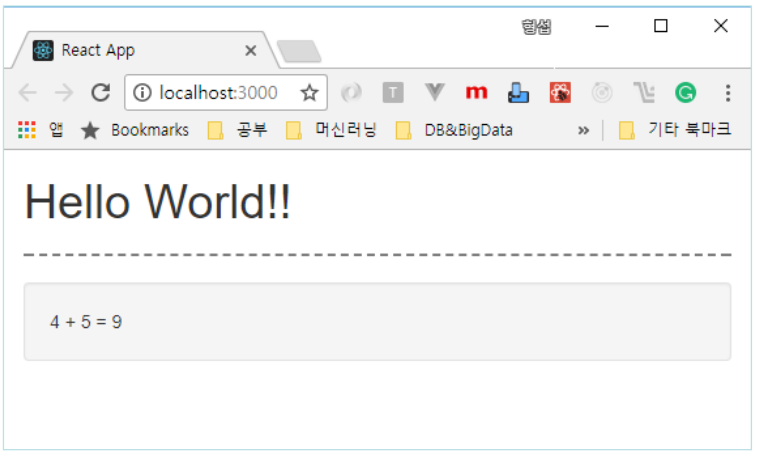
```
.....
import 'bootstrap/dist/css/bootstrap.css'
import './index.css'
.....
```

- src/App.js 변경

```
.....
<div class="card card-body bg-light mb-3">{x} + {y} =
{x+y}</div>
.....
<div class="container">
    <h1>Hello {msg}</h1>
    <hr class="dash-style" />
    { this.createString(4,5) }
</div>
.....
```

3. 간단한 스타일 적용(3)

- 전역 수준에서 참조
  - index.js에 import한 css 스타일을 App.js 컴포넌트에서 사용하였음.
- 실행 결과

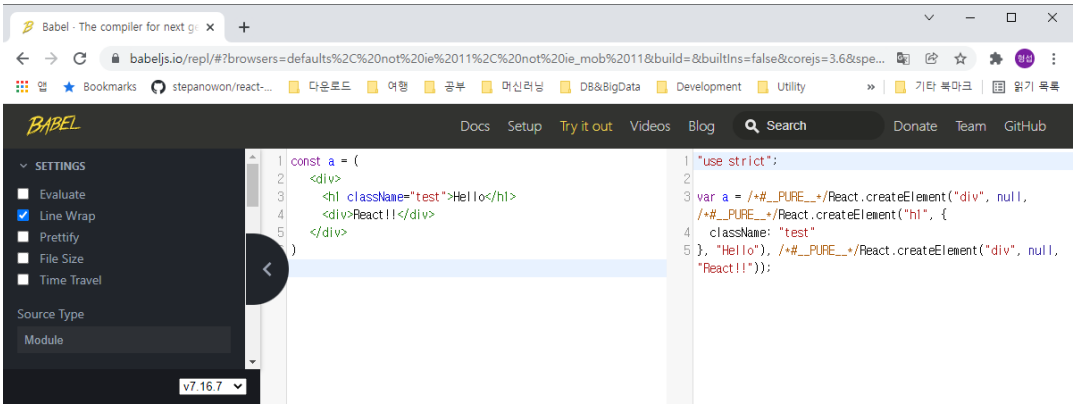


■ 이 코드에서는 class 명이 잘못되었다고 오류가 발생하지만 다음 절에서 오류를 수정할 것이므로 무시한다.

4. JSX(1)

JSX란?

- NOT HTML!!
- 선언적 XML 스타일의 자바스크립트 확장 문법
  - Javascript 코드로 변환되어 실행됨.
- babel repl 도구를 이용한 변환 (https://babeljs.io/repl/)



■ JSX를 사용한 코드가 React.createElement() 메서드를 사용한 코드보다 가독성, 유지보수성이 훨씬 좋다.

4. JSX(2)

- JSX는 선택적 요소임
  - 반드시 사용해야 하는 것은 아님. 하지만 장점이 많음
  - UI를 표현하기에 더 적합함. 특히 HTML, XML 의 트리 구조
  - 애플리케이션의 구조를 시각화하기에 더 좋음
  - 자바스크립트 코드이므로 언어의 의미가 변형되지 않음

4. JSX(3)

주의사항

- 태그의 Attribute는 카멜 표기법(camel casing)을 준수함
  - 예) onclick --> onClick
    - HTML : <button **onclick**="start()" />
    - JSX : <button **onClick**={start} />
- Attribute 이름이 DOM API 스펙에 기반을 두고 있음.
  -
  - document.getElementById("a").className="test";
  - HTML : Hello
  - JSX : <div **className**="test">Hello

14

4. JSX(4)

JSX 표현식에 동적으로 값을 넣고자 한다면?

- { } 보간법(interpolation)을 사용함.
- { } 내부에 값, 메서드의 리턴값, 속성 등이 위치할 수 있음.
- { } 에 복잡한 자바스크립트 구문을 배치할 수 없음
  - 예1) if문을 { } 내부에 작성할 수 없음.
    - 3항 연산식은 허용함({ a? b:c })
  - 예2) for 문 반복문을 { } 내부에 작성할 수 없음
  - 외부 에서 연산처리하여 변수에 값을 저장한 후 보간해야 함.
- 값은 모두 HTML Encoding 되어 출력됨
  - XSS 공격 때문에 자동으로 HTML Encoding을 수행함.
  - 그럼에도 불구하고 HTML 그대로 출력하려면 dangerouslySetInnerHTML 특성을 사용함
    - 예제는 아래 내용 참조

15

■ dangerouslySetInnerHTML 특성을 사용하여 HTML 을 그대로 렌더링하기.

```
let msg = "<i>World</i>";
let Hello = function(props) {
    return (
        <div>
            Hello
            <span dangerouslySetInnerHTML={{ __html:msg }} />
        </div>
    );
}
```

4. JSX(5)

- 보간 기능을 테스트하기 위한 컴포넌트
- src/CountryList.js

```
import React, { Component } from 'react';

class CountryList extends Component {
   render() {
      let list = [ { no:1, country:'이집트', visited:false }, { no:2, country:'일본',  visited:true },
         { no:3, country:'피지', visited:false }, { no:4, country:'콜롬비아', visited:false }  ];
      let countries = list.map((item, index) => {
        return (
            <li key={item.no}
               className={item.visited ? 'list-group-item active' : 'list-group-item' }>
               {item.country}
            </li>
        )
      })
      return (
         <ul className="list-group">{countries}</ul>
      );
   }
}
export default CountryList;
```

4. JSX(6)

- src/CountryList.js 의 3항 연산자를 사용한 부분을 다음과 같이 변경할 수 있음

```
return (
    <li key={item.no}
        className={item.visited ? 'list-group-item active' : 'list-group-item' }>
        {item.country}
    </li>
)
```

```
let countryClass = "";
if (item.visited)   {
    countryClass = 'list-group-item active';
} else {
    countryClass = 'list-group-item';
}
return (
    <li key={item.no} className={countryClass}>
        {item.country}
    </li>
)
```


17

4. JSX(7)

- CountryList 컴포넌트 테스트
  - src/App.js 코드 변경

```
……
import CountryList from './ CountryList ';

class App extends Component {

    render() {
        let msg = "World!!";
        return (
            <div className="container">
                ……
                <CountryList />
            </div>
        );
    }
}
……
```

4. JSX(8)

단일 루트 노드

- 단일 루드 요소만 렌더링할 수 있음.
- 여러개의 요소를 렌더링하려면 와 같은 요소로 감싸주어야 함.

```
render() {
  return (
    <div>Hello</div>
    <div>World</div>
  );
}
```

```
render() {
  return (
    <div>
      <div>Hello</div>
      <div>World</div>
    </div>
  );
}
```

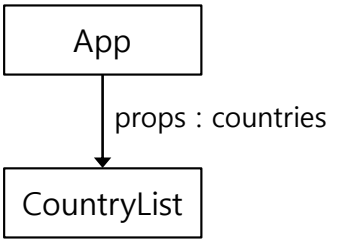

■ React V16부터는 등으로 감싸지 않아도 되는 방법을 지원한다. <></>로 감싸는 방법인데, 이 요소는 렌더링시에는 반영되지 않고 요소들의 그룹을 지정하는 역할만을 수행한다.

```
render() {
  return (
    <>
      <div>Hello</div>
      <div>World</div>
    </>
  );
}
```

5. 속성(1)

속성 : props

- 컴포넌트가 외부로부터 데이터를 전달받기 위해 사용
  - 부모 컴포넌트 --> 자식컴포넌트로 정보 전달
- 전달받은 속성의 값은 그 컴포넌트에서 변경하지 않음
- src/CountryList.js를 App 컴포넌트로부터 속성을 전달받도록 변경



■ 속성으로 전달한 데이터는 부모의 소유이다.

5. 속성(2)

- src/App.js 변경

```
.......
class App extends Component {
    ......
    render() {
        let list = [
            { no:1, country:'이집트', visited:false },
            { no:2, country:'일본',  visited:true },
            { no:3, country:'피지', visited:false },
            { no:4, country:'콜롬비아', visited:false }
        ];

        let msg = "World!!";
        return (
            <div className="container">
                <h1>Hello {msg}</h1>
                <hr className="dash-style" />
                { this.createString(4,5) }
                <CountryList countries={list} />
            </div>
        );
    }
}
......
```

■ list 변수의 배열 값을 countries 속성을 통해 자식 컴포넌트로 전달함.

5. 속성(3)

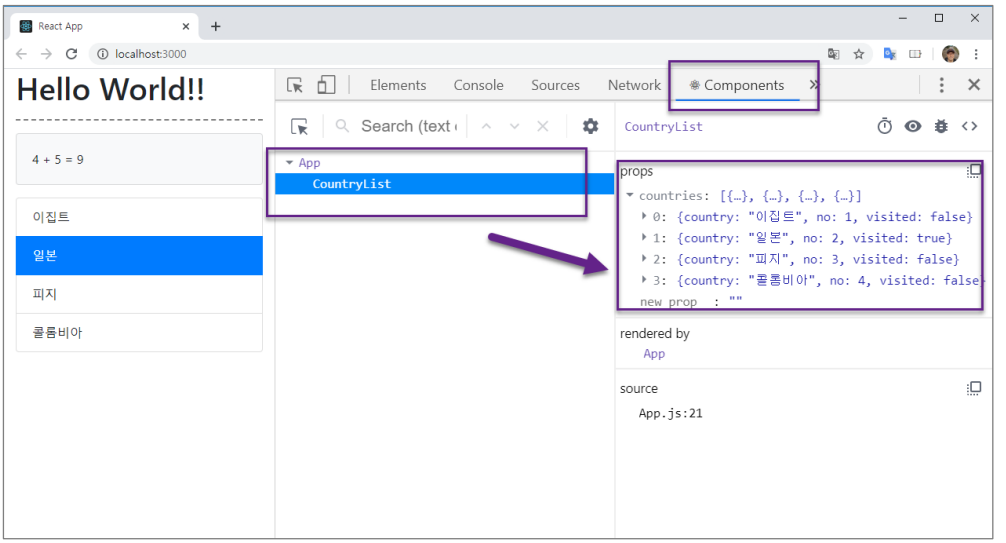
- src/CountryList.js 변경

```
……
class CountryList extends Component {
    render() {
        const list = this.props.countries;
        let countries = list.map((item, index) => {
          return (
              <li key={item.no}
                  className={item.visited ? 'list-group-item active' : 'list-group-item' }>
                  {item.country}
              </li>
          )
        })
        return (
            <ul className="list-group">{countries}</ul>
        );
    }
}
……
```


■ 자식 컴포넌트에서 this.props의 값은 변경할 수 없음.

5. 속성(4)

- 실행 결과 확인
  - React Developer Tools 활용하여 구조 확인

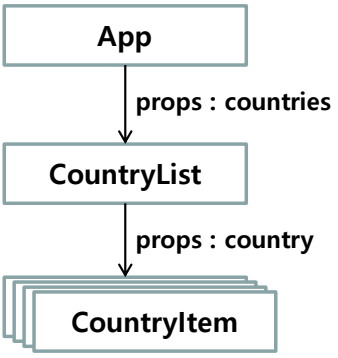


23

5. 속성(5)

컴포넌트 세분화!

- CountryItem.js
  - 나라 정보 하나를 다루는 컴포넌트
  - CountryList 컴포넌트 내부에 Country Item컴포넌트 여러개를 포함함.
  - Country Item컴포넌트의 렌더링에 필요한 정보를 속성(props)을 통해 전달함.



5. 속성(6)

- src/CountryList.js 컴포넌트 변경

```
import React, { Component } from 'react';
import CountryItem from './CountryItem'

class CountryList extends Component {
    render() {
        let countries = this.props.countries.map((item, index) => {
            return (
                <CountryItem key={item.no} country={item}/>
            )
        })

        return (
            <ul className="list-group">
                {countries}
            </ul>
        );
    }
}

export default CountryList;
```

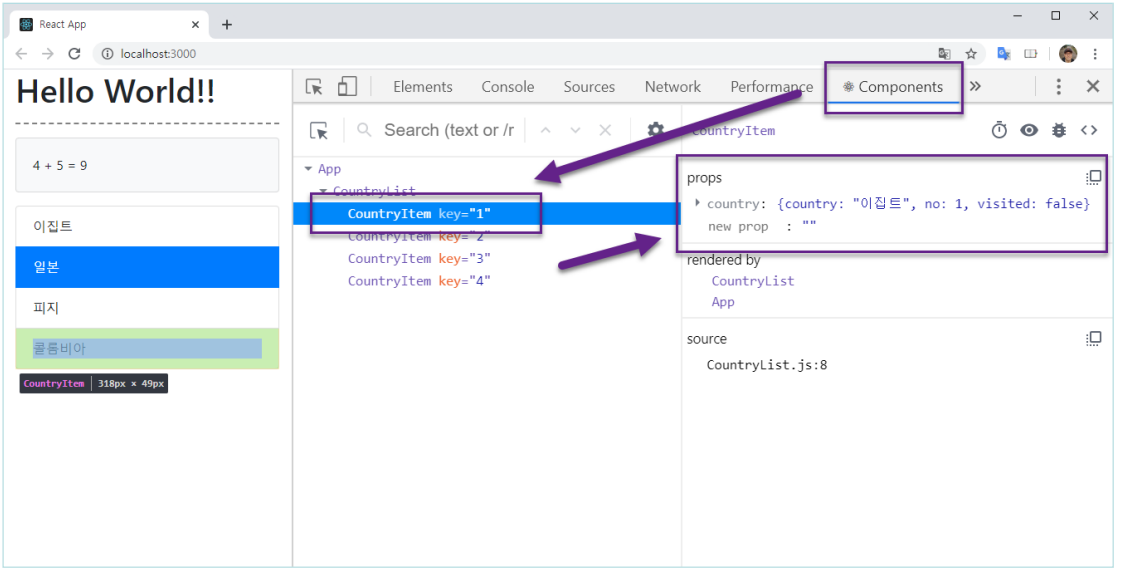
5. 속성(7)

- src/CountryItem.js 컴포넌트 작성

```
import React, { Component } from 'react';

class CountryItem extends Component {
    render() {
        let item = this.props.country;
        return (
            <li className={item.visited ? 'list-group-item active' : 'list-group-item' }>
                    {item.country}
            </li>
        );
    }
}

export default CountryItem;
```

5. 속성(8)

- 실행 결과


React App
localhost:3000
Hello World!!
4 + 5 = 9
이집트
일본
피지
콜롬비아
CountryItem 318px × 49px
Elements
Console
Sources
Network
Performance
Components
Search (text or /r
App
CountryList
CountryItem key="1"
CountryItem key="2"
CountryItem key="3"
CountryItem key="4"
CountryItem
props
country: {country: "이집트", no: 1, visited: false}
new prop : ""
rendered by
CountryList
App
source
CountryList.js:8

27

5. 속성(9)

ES6의 Spread Operator를 사용한 속성 전달

- 전달해야할 속성이 여러 개인 경우 유용한 방법
- 자식 컴포넌트의 속성(props)의 이름과 객체의 속성명이 일치하다면...

```
render() {
    return (
        <li className={this.props.visited ? 'list-group-item active' : 'list-group-item' }>
            {this.props.country}
        </li>
    );
}
```

```
let countries = this.props.countries.map((item, index) => {
    return (
        <CountryItem key={item.no} { ...item }/>
    )
})
```

28

■ CountryList.js 컴포넌트 코드의 item 객체는 다음과 같은 형태이다.

- { no:'1', country:'이집트', visited: false }

■ { ...item } 과 같은 Rest Operator는 item 객체의 속성의 동일한 이름의 자식 컴포넌트 속성으로 전달한다. 따라서 자식 컴포넌트에서는 this.props.visited, this.props.country 와 같이 전달받은 속성 값을 이용할 수 있다.

■ 어느 방식을 권장할까?

- 속성으로 객체를 전달하는 방법을 권장함
  - 렌더링 성능은 속성값으로 객체를 전달하는 것이 더 빠르다.
  - 속성으로 전달된 값이 참조형이므로 자식 컴포넌트에서 변경할 수 있다는 문제점이 있지만 개발자들에게 코딩 규칙으로 가이드하면 해결될 것

5. 속성(9)

정리

- 하위컴포넌트로 속성을 전달할 때
  - <CountryItem key={item.no} country={item}/>
- 하위 컴포넌트에서 속성을 이용할 때
  - let item = this.props.country;
- 속성의 전달방향 : 부모 컴포넌트 --> 자식 컴포넌트
- 자식 컴포넌트에서 부모로부터 전달받은 속성의 값을 변경하지 않음
  - 속성을 전달하기 시작한 부모 컴포넌트에서만 변경
  - 부모 컴포넌트에서 데이터를 변경하면 자식 컴포넌트의 UI가 다시 렌더링됨
- 속성을 이용해 메서드를 자식컴포넌트로 전달할 수 있음.

29

6. 상태(1)

상태 : state

- 컴포넌트가 보유하는 데이터
- 속성을 통해서 자식 컴포넌트로 전달할 수 있음.
- 상태의 변경은 보유하고 있는 컴포넌트에서만 수행함.
  - 속성을 통해 자식컴포넌트로 전달된 경우 자식컴포넌트에서 상태 변경 불가.

상태 컴포넌트와 무상태 컴포넌트

- 상태 컴포넌트 : stateful component
  - 자신의 상태를 가지는 컴포넌트
- 무상태 컴포넌트 : stateless component
  - 자신의 상태가 없으며 속성을 통해서 부모 컴포넌트로부터 데이터를 전달받아야만 하는 컴포넌트
- 컴포넌트의 재사용성은 무상태 컴포넌트가 좋다!!

30

6. 상태(2)

상태의 초기화와 변경

- 상태 데이터의 초기화는 생성자(constructor)에서 처리함.
  - 반드시 super() 구문을 먼저 호출한 다음 초기화되어야 함.
  - this.state = { }
- 상태의 변경은 setState() 메서드를 이용해야 함.
  - 생성자에서 초기화할 때만 this.state = { } 을 허용함.
  - 초기화한 이후에는 변경을 위해 반드시 this.setState 메서드를 이용해야 함.
- 상태의 변경은 상태를 보유한 컴포넌트에서만 가능함.

■ 상태의 변경과 관련된 부분은 다음장 React 컴포넌트를 학습할 때 좀더 자세하게 다룬다.

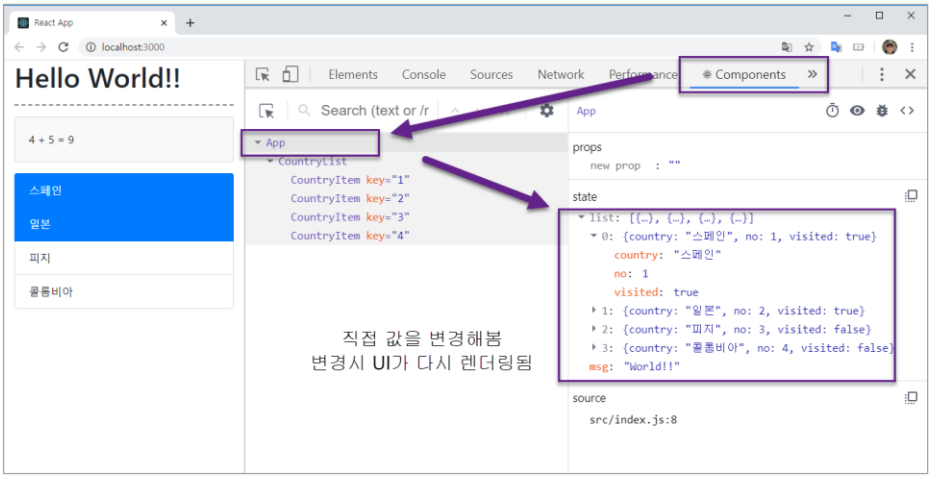
6. 상태(3)

- src/App.js를 상태 컴포넌트로 변경

```
import React, { Component } from 'react';
import CountryList from './CountryList';

class App extends Component {
    constructor(props) {
        super(props);
        this.state = {
            msg : "World!!",
            list : [
                { no:1, country:'이집트', visited:false },
                { no:2, country:'일본',  visited:true },
                { no:3, country:'피지', visited:false },
                { no:4, country:'콜롬비아', visited:false }
            ]
        }
    }

    createString(x,y) {
        return (
            <div className="card card-body bg-light mb-3">
                {x} + {y} = {x+y}</div>
        )
    }

    render() {
        return (
            <div className="container">
                <h1>Hello {this.state.msg}</h1>
                <hr className="dash-style" />
                { this.createString(4,5) }
                <CountryList countries={this.state.list} />
            </div>
        );
    }
}

export default App;
```

6. 상태(4)

- yarn start로 실행한 후 결과 확인



■ 개발자 도구 화면(위 화면) 상에서 직접 값을 수정하면 화면이 변경됨

- State ------> UI

6. 상태(5)

상태(State) 정리

- 컴포넌트의 변경 가능한 데이터
- 상태 데이터를 보유한 컴포넌트 내부에서만 변경할 수 있음.
  - 상태의 변경에 대해서는 다음 장에서 다룸

가능하다면 상태가 없는 컴포넌트를 만들자!

- props를 전달받아 실행하는 컴포넌트
- 상태가 없을수록 컴포넌트의 재사용성이 좋아짐.
  - 부모 컴포넌트(Stateful Component)의 상태를 자식 컴포넌트들이 전달받아 사용하도록 작성한다.