

6. 리액트 상태 관리 (2)

Prof. Seunghyun Park (sp@hansung.ac.kr)

Division of Computer Engineering

학습 목표: 6장. 리액트 상태 관리

- 리액트 상태
 - 예제 (별점 프로젝트: StarRating, Star 컴포넌트 구성)
- 컴포넌트 상태와 상태 변경을 위한 `useState()`
- 컴포넌트 트리
- 폼만들기
- 리액트 컨텍스트

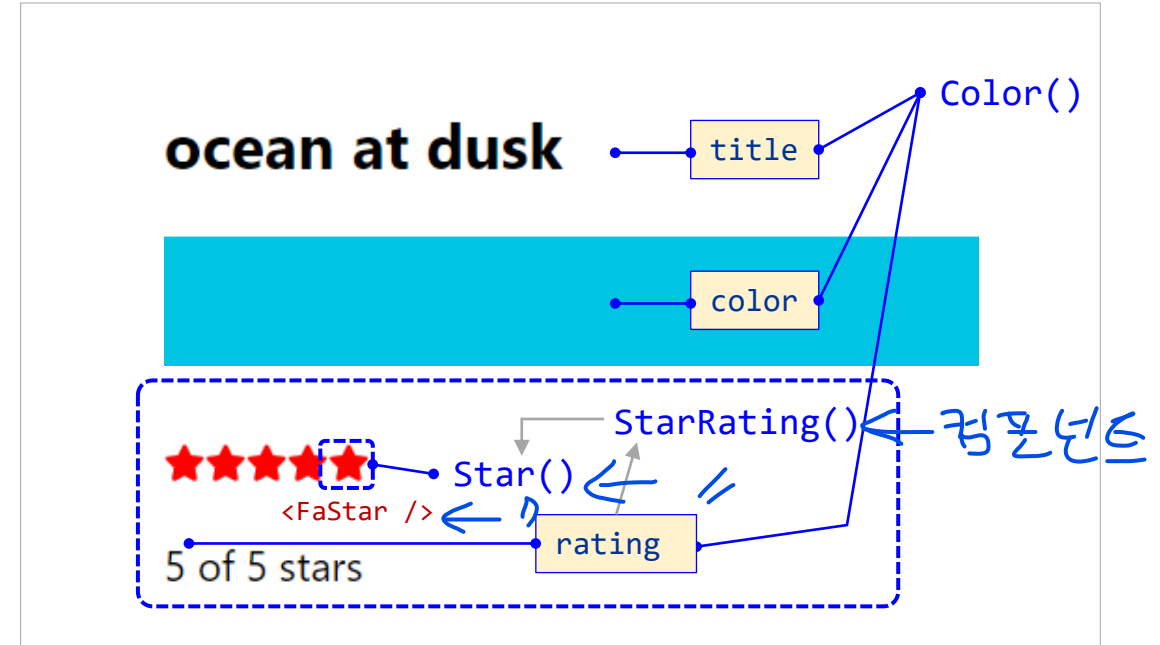
• 컴포넌트 트리

- DOM이 트리 구조를 갖는 것처럼
컴포넌트도 트리 구조에 따라 엘리먼트를 구성

※ 상태: 리액트 컴포넌트의 데이터를 표현하는 객체
→ 컴포넌트 내부에서 변경될 수 있는 값

• 컴포넌트의 상태 관리

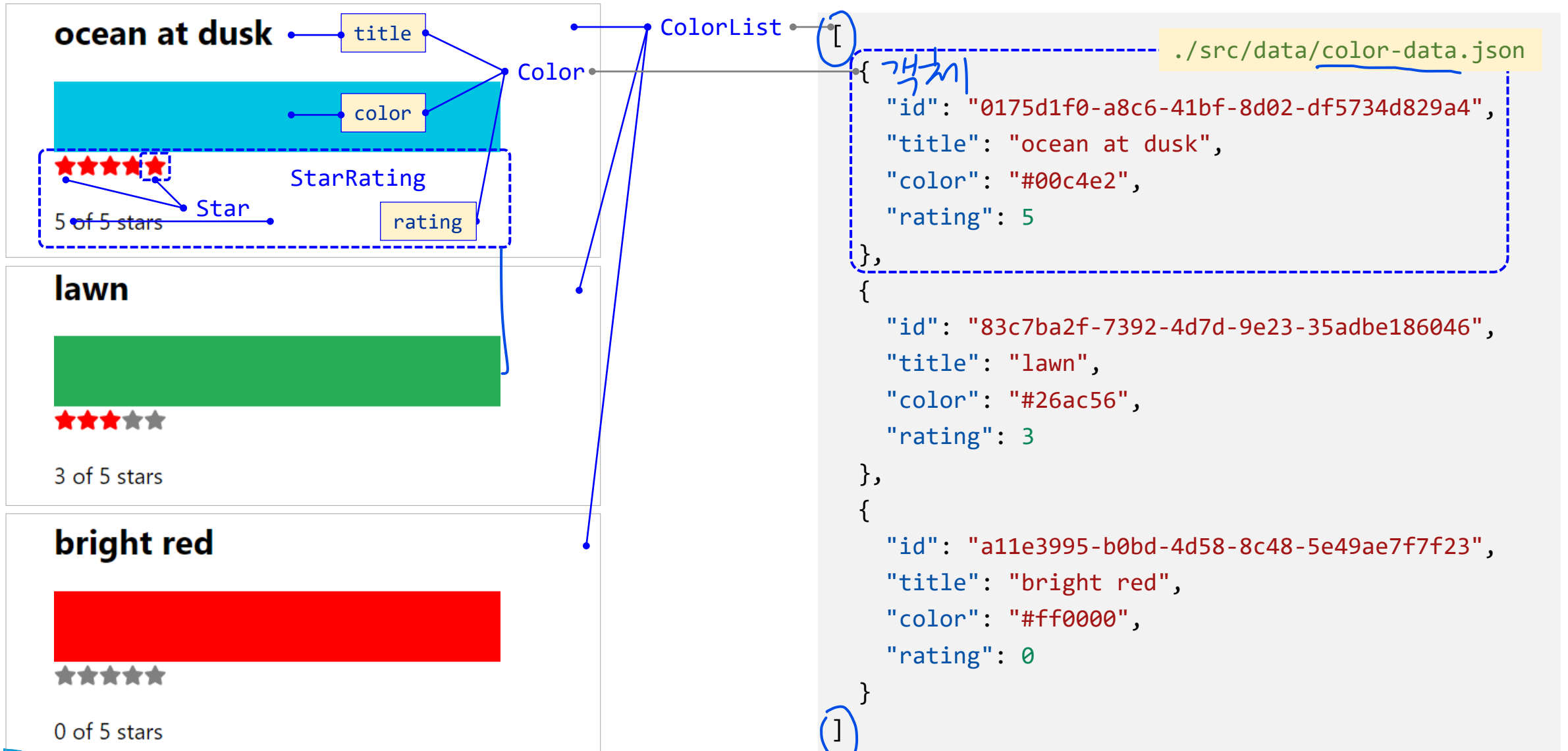
- 각 컴포넌트가 상태를 가질 수도 있고
- 컴포넌트 트리에 따라 데이터를 전달할 수도 있음



```
{
  "id": "0175d1f0-a8c6-41bf-8d02-df5734d829a4",
  "title": "ocean at dusk",
  "color": "#00c4e2",
  "rating": 5
},
```

App() → StarRating() → Star() → <FaStar />

rating → true/false → red/grey

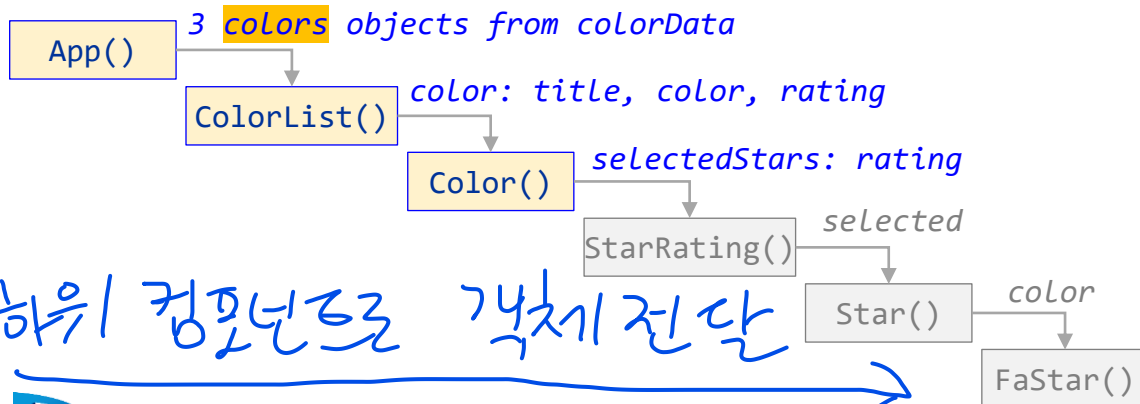


```
/* color-org/src/App.js */
import { useState } from "react";
import ColorList from "../components/ColorList";
import colorData from "../data/color-data.json"
```

```
function App() {
  const [colors] = useState(colorData);
  return <ColorList colors={colors} />;
}
```

앱의 메인 콘텐츠는 ColorList

상태: colors
json 형태의 데이터 colorData의 값을 배열의 구조분해 할당으로 변수 생성



하위 컴포넌트로 객체 전달

```
/* color-org/src/components/ColorList.js */
import React from "react";
import Color from "../Color";
const ColorList = function( { colors = [] } ){
  return (
    <div>
      { colors.map( color =>
        <Color key={color.id} {...color} /> ) }
    </div>
  );
}
export default ColorList;
```

```
/* color-org/src/components/Color.js */
import React from "react";
import StarRating from "../StarRating";
const Color = function( { title, color, rating } ){
  return (
    <section>
      <h1>{title}</h1>
      <div style={{ backgroundColor: color, height: 50 }} />
      <StarRating selectedStars={rating} />
    </section>
  );
}
export default Color;
```

```
/* color-org/src/components/StarRating.js */
import React from "react";
import Star from "../Star";
const createArray = length => [...Array(length)];

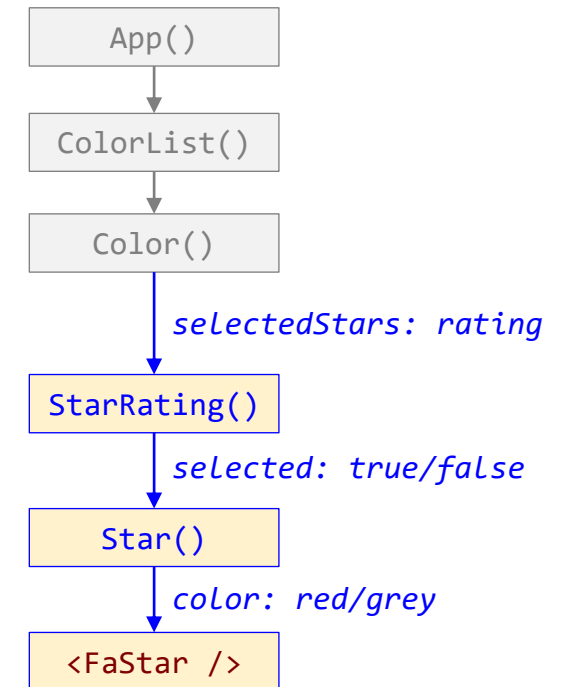
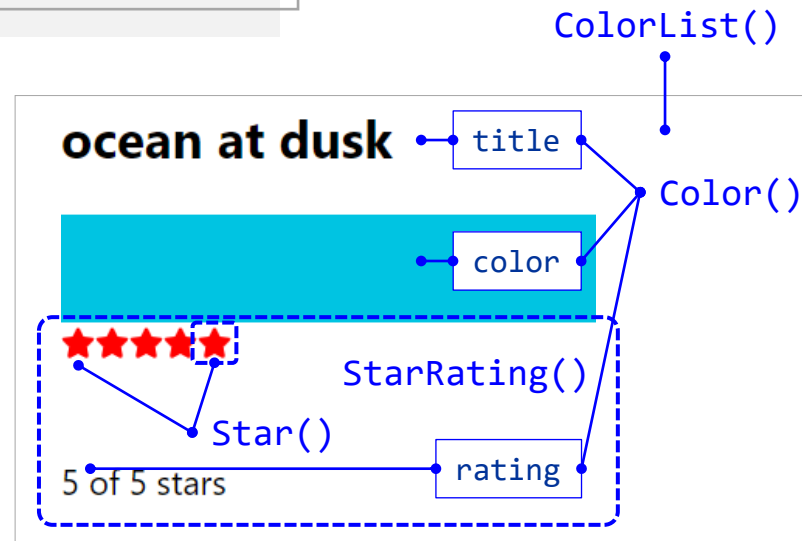
const StarRating = function( { totalStars=5, selectedStars=0 }){
  return (
    <>
      {createArray(totalStars).map( (n, i) => (
        <Star
          key={i}
          selected={selectedStars > i}
        /> )) }
      <p>{selectedStars} of {totalStars} stars</p>
    </>
  );
}

export default StarRating;
```

```
/* color-org/src/components/Star.js */
import { FaStar } from "react-icons/fa";

const Star = ({ selected = false }) => (
  <FaStar color={selected ? "red" : "grey"} />
);

export default Star;
```



색상 관리 앱 -- 아이템 삭제: onClick() → onRemove()

color-org-06

```
/* color-org/src/components/Color.js */
```

```
import React from "react";
```

```
import { FaTrash } from "react-icons/fa";
```

```
import StarRating from "../StarRating"
```

```
const Color = function( { id, title, color, rating, onRemove = f => f } ){
```

```
  return (
```

```
    <section>
```

```
      <h1>{title}</h1>
```

```
      <button onClick={ ( ) => onRemove(id) }>
```

```
        <FaTrash />
```

```
      </button>
```

```
      <div style={{ backgroundColor: color, height: 50 }} />
```

```
      <StarRating selectedStars={rating} />
```

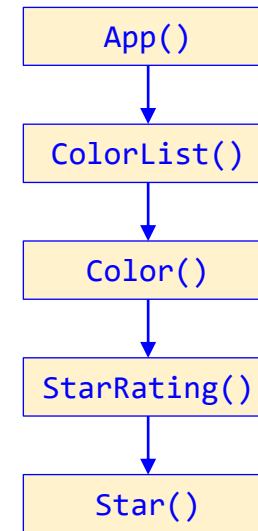
```
    </section>
```

```
  );
```

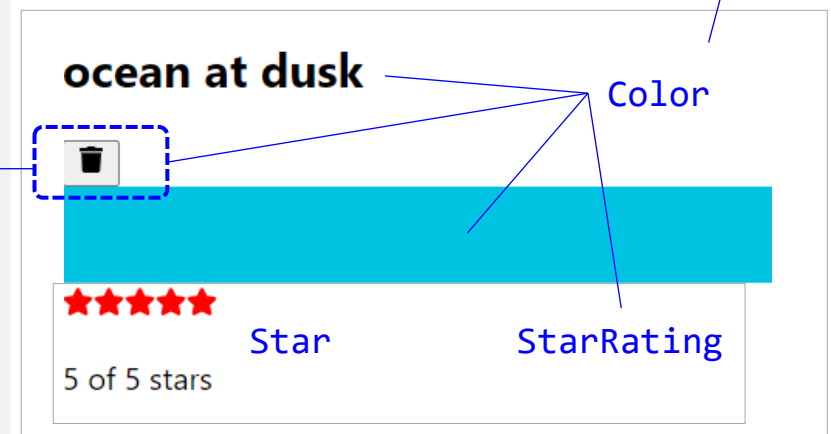
```
}
```

```
export default Color;
```

리스트에서 무언가를 삭제하려면
상위 컴포넌트가
호출하도록 해야함



상위 컴포넌트에서
하위 컴포넌트를
생성하면서,
하위 컴포넌트가
필요로 하는
데이터를 넘겨줌



색상 관리 앱 -- 아이템 삭제: onClick() → onRemove() → onRemoveColor() → setColors()

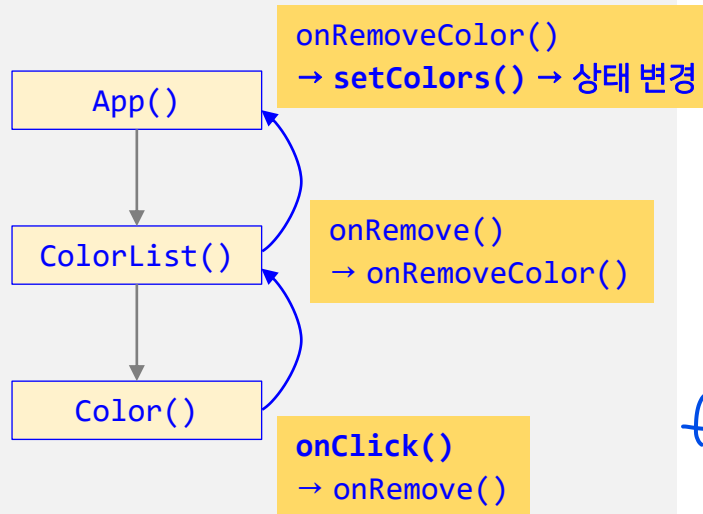
color-org-06

```
/* color-org/src/components/ColorList.js */
import React from "react";
import Color from "../Color";

const ColorList = function({ colors = [],
                             onRemoveColor = f => f }){

  return (
    <div>
      { colors.map( color =>
        <Color key={color.id} {...color}
          onRemove={onRemoveColor} /> ) }
    </div>
  );
}
```

```
export default ColorList;
```



```
/* color-org/src/App.js */
import { useState } from "react";
import ColorList from "../components/ColorList";
import colorData from "../data/color-data.json"

function App() {
  const [colors, setColors] = useState(colorData);
  return (
    <ColorList
      colors={colors}
      onRemoveColor={id => {
        const newColors = colors.filter(
          color => color.id !== id);
        setColors(newColors);
      }}
    />);
}
```

```
export default App;
```

삭제할 데이터는 filter() 처리
※ 데이터는 존재함 원본은 유지

필터링 한 후, 현재의 상태 colors를 newColors로 업데이트
> colors 상태가 변경되면 App()은 다시 렌더링 됨

컴포넌트의

filter: 통과하는 것만 모아
새로운 배열 반환

색상 관리 앱 -- 평점 변경: onClick() → onSelect() → onRate()

color-org-07

```
/* color-org/src/components/Star.js */
```

```
const Star = function( {  
  selected = false,  
  onSelect = f => f  
}) {  
  return (  
    <FaStar  
      color={selected ? "red" : "grey"}  
      onClick={onSelect}  
    />  
  );  
};  
  
export default Star;
```

ocean at dusk



4 of 5 stars

onClick

```
/* color-org/src/components/StarRating.js */
```

```
const createArray = length => [...Array(length)];  
const StarRating = function( {  
  totalStars=5, selectedStars=0,  
  onRate = f => f  
}){  
  return (  
    <>  
      {createArray(totalStars).map( (n, i) => (  
        <Star  
          key={i}  
          selected={selectedStars > i}  
          onSelect={() => onRate(i+1)}  
        /> )) }  
      <p>{selectedStars} of {totalStars} stars</p>  
    </>  
  );  
}  
  
export default StarRating;
```

색상 관리 앱 -- 평점 변경: onRate() → onRateColor() → rateColor()

color-org-07

```
/* color-org/src/components/Color.js */

const Color = function( {
  id, title, color, rating,
  onRemove = f => f,
  onRate = f => f
}){
  return (
    <section>
      <h1>{title}</h1>
      <button onClick={ () => onRemove(id) }>
        <FaTrash />
      </button>
      <div style={{ backgroundColor: color,
                    height: 50 }} />
      <StarRating
        selectedStars={rating}
        onRate={rating => onRate(id, rating)}
      />
    </section>
  );
}

export default Color;
```

```
/* color-org/src/components/ColorList.js */

const ColorList = function( {
  colors = [],
  onRemoveColor = f => f,
  onRateColor = f => f
}){
  return (
    <div className="color-list">
      { colors.map( color =>
        <Color
          key={color.id}
          {...color}
          onRemove={onRemoveColor}
          onRate={onRateColor}
        /> ) }
    </div>
  );
}

export default ColorList;
```

색상 관리 앱 -- 평점 변경: onRateColor() → rateColor() → setColors()

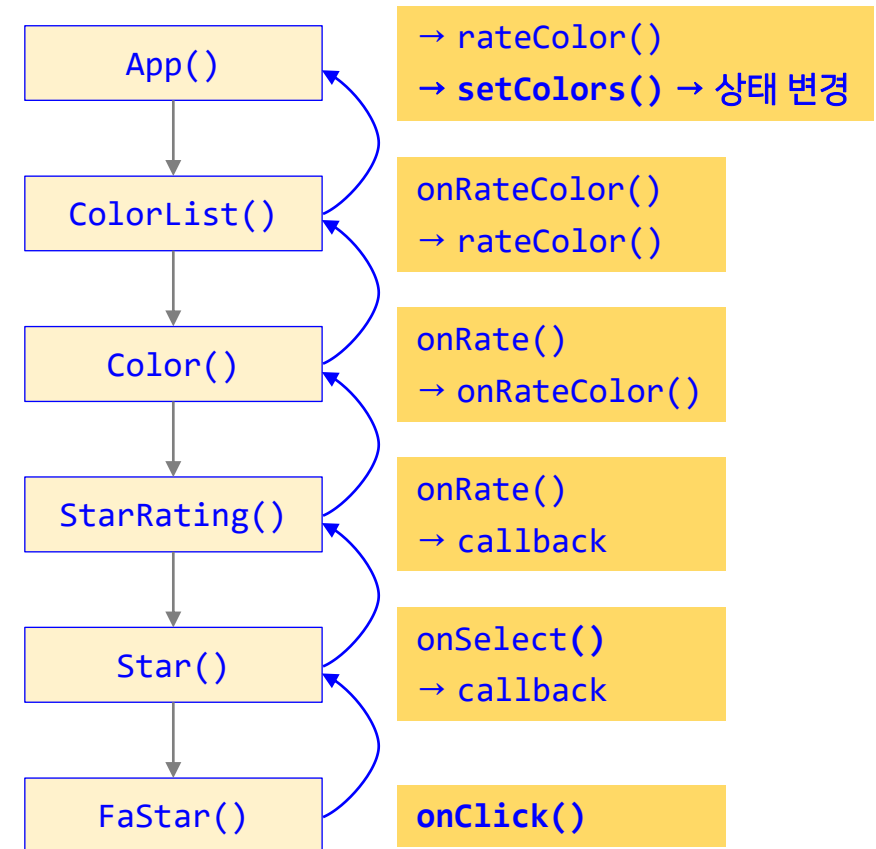
color-org-07

```
/* color-org/src/App.js */
```

```
function App() {  
  const [colors, setColors] = useState(colorData);  
  const removeColor = id => {  
    const newColors = colors.filter( color => color.id !== id );  
    setColors(newColors);  
  };  
  const rateColor = (id, rating) => {  
  const newColors = colors.map( color => color.id === id ?  
    {...color, rating } : color );  
  setColors(newColors);  
};  
return (  
  <ColorList  
    colors={colors} onRemoveColor={ removeColor }  
    onRateColor={ rateColor }  
  />);  
}
```

```
export default App;
```

color-org-05

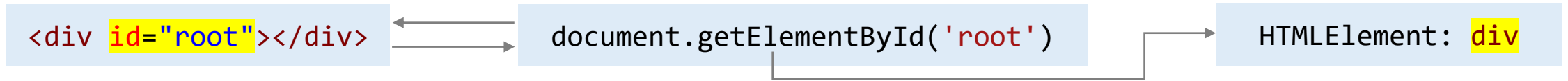


이렇게 해야 side effect를 방지

참조: ref 떠넘기지 않고 직접 참조해야 할 때

- DOM의 id와 리액트 컴포넌트에서의 ref

- 엘리먼트를 참조하기 위해서 사용 → 유일한 값이어야 객체를 참조하여 활용할 수 있음



- 리액트 컴포넌트에서 DOM 객체를 직접 참조해야 하는 경우 useRef Hook 활용

- 참조 객체 선언: `useRef()` → `const __ref = useRef();`

- 엘리먼트에 ref 속성 추가 → `<input ref={__ref} type="text" ... />`

- 참조 객체의 current 필드 활용 → `__ref.current.value`


- 리액트 컴포넌트 안에서도 id를 사용할 수 있으나, 컴포넌트를 재사용 할 경우 id가 중복될 수 있으므로 사용하지 않음

※ 참조 객체 ref는 컴포넌트 내부에서만 동작

ADD

`<input type="text" />`

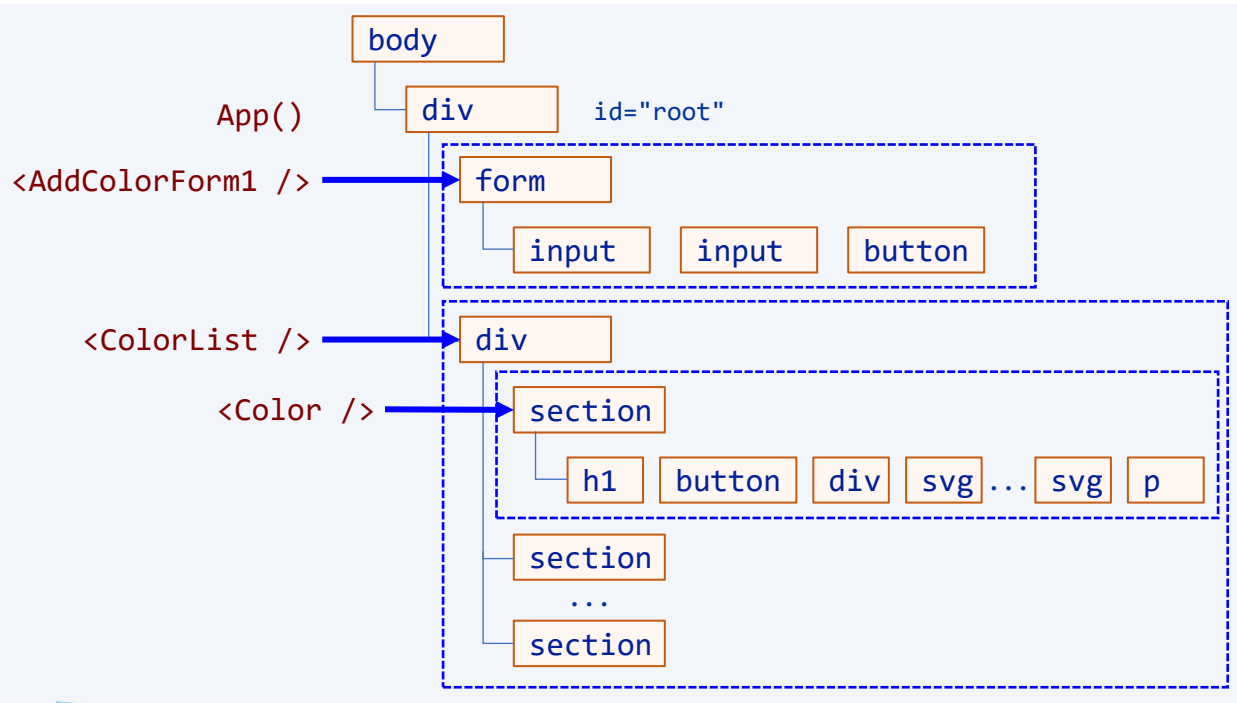
ocean at dusk



★★★★★

4 of 5 stars

`<input type="color" />`



```
<form>
  <input type="text" placeholder="color title..." required />
  <input type="color" required />
  <button>ADD</button>
</form>
```

`<form />`

- 2개의 입력, 1개의 버튼으로 구성
- 사용자의 입력을 받음

```
/* color-org/src/App.js */
import AddColorForm1 from "../components/AddColorForm1";
import ColorList from "../components/ColorList";
...
function App() {
  ...
  return (
    <>
      <AddColorForm1
        onNewColor={(title, color) =>
          alert(`TODO: Create ${title} - ${color}`)} />
      <ColorList ... />
    </>);
}

export default App;
```

```
/* color-org/src/components/AddColorForm1.js */
import React, {useRef} from "react";
const AddColorForm1 = function({ onNewColor = f => f }){
  const txtTitle = useRef();
  const hexColor = useRef();
  const submit = e => {
    e.preventDefault();
    const title = txtTitle.current.value;
    const color = hexColor.current.value;
    onNewColor(title, color);
    txtTitle.current.value = "";
    hexColor.current.value = "";
  };
  return (
    <form onSubmit={submit}>
      <input ref={txtTitle}
        type="text" placeholder="color title..." required />
      <input ref={hexColor}
        type="color" required />
      <button>ADD</button>
    </form>
  );
}
export default AddColorForm1;
```

submit 이벤트를 리스너

```
/* color-org/src/App.js */
import { useState } from "react";
import ColorList from "../components/ColorList";
import colorData from "../data/color-data.json";
import AddColorForm1 from "../components/AddColorForm1";

function App() {
  const [colors, setColors] = useState(colorData);
  return (
    <>
      <AddColorForm1
        onNewColor={(title, color) => {
          const newColors = [ ...colors, {
            id: v4(),
            rating: 0,
            title,
            color
          } ];
          setColors(newColors);
        }}
      />
      <ColorList ... />
    </>
  );
}
export default App;
```

현재 상태 colors에 {id, rating, title, color} 객체를 추가한 newColors로 업데이트

useRef() 대신 useState() 사용

color-org-09

fisk

```
/* color-org/src/components/AddColorForm1.js */
import React, {useRef} from "react";
const AddColorForm1 = function({ onNewColor = f => f }){
  const txtTitle = useRef();
  const hexColor = useRef();
  const submit = e => {
    e.preventDefault();
    const title = txtTitle.current.value;
    const color = hexColor.current.value;
    onNewColor(title, color);
    txtTitle.current.value = "";
    hexColor.current.value = "";
  };

  return (
    <form onSubmit={submit}>
      <input ref={txtTitle}
        type="text" placeholder="color title..." required />
      <input ref={hexColor}
        type="color" required />
      <button>ADD</button>
    </form>
  );
}
export default AddColorForm1;
```

DOM을 직접 변경

good

```
/* color-org/src/components/AddColorForm2.js */
import React, {useState} from "react";
const AddColorForm2 = function({ onNewColor = f => f }){
  const [title, setTitle] = useState("");
  const [color, setColor] = useState("#000000");
  const submit = e => {
    e.preventDefault();
    onNewColor(title, color);
    setTitle("");
    setColor("");
  };
  return (
    <form onSubmit={submit}>
      <input
        value={title}
        onChange={event => setTitle(event.target.value)}
        type="text" placeholder="color title..." required />
      <input value={color}
        onChange={event => setColor(event.target.value)}
        type="color" required />
      <button>ADD</button>
    </form>
  );
}
export default AddColorForm2;
```

상태 변경 함수가 상태를 변경하면, 다시 렌더링이 일어남

<input /> 상태가 변경됨에 따라
title 업데이트 (예: 키 입력)

학습 정리: 6장. 리액트 상태 관리

- 리액트 상태
 - 예제 (별점 프로젝트: StarRating, Star 컴포넌트 구성)
- 컴포넌트 상태와 상태 변경을 위한 `useState()`
- 컴포넌트 트리
- 폼만들기
- 리액트 컨텍스트