React Components

개요

함수형 프로그래밍이란 함수가 항상 같은 결과를 반환하기를 기대하고 코드를 작성하는 프로그래밍 기법이다. 이를 멱등성이라고 한다. React도 이런 개념을 기반으로 설계되었다.

함수형 컴포넌트

React의 모든 컴포넌트는 순수 함수로 가정한다. 이런 가정은 같은 입력이 주어졌을때 항상 같은 JSX를 반환한다는 것을 의미한다.

```
function Recipe({ drinkers }) {
  return (
    <ol>
      <li>Boil {drinkers} cups of water.</li>
      <li>Add {drinkers} spoons of tea and {0.5 * drinkers} spoons of spice.</li>
      <li>Add {0.5 * drinkers} cups of milk to boil and sugar to taste.</li>
    </ol>
  );
}

export default function App() {
  return (
    <section>
      <h1>Spiced Chai Recipe</h1>
      <h2>For two</h2>
      <Recipe drinkers={2} />
      <h2>For a gathering</h2>
      <Recipe drinkers={4} />
    </section>
  );
}
```

수학 공식과도 같은 순수 함수는 다음과 같은 특징이 있다.

- 함수의 호출로 인해서 기존에 존재했던 객체나 변수의 상태를 변경하지 않는다.
- 같은 입력이 주어졌다면 항상 같은 결과를 반환한다.

Side Effects

사이드 이펙트란 React의 렌더링 과정이 순수하지 않다는 의미이다. 컴포넌트는 JSX만 반환해야 하며 렌더링 이전에 존재했던 객체나 변수를 변경하지 않아야 한다. 아래 코드 조각은 사이드 이펙트를 발생시키는 안 좋은 예제이다.

```jsx
function Cup() {
 // 나쁜 지점: 이미 존재했던 변수를 변경하고 있습니다!
 guest = guest + 1;
 return <h2>Tea cup for guest #{guest}</h2>;
}

export default function TeaSet() {
 return (
  <>
   <Cup />
   <Cup />
   <Cup />
  </>
 );
}
```

함수 안에서 `guest` 라는 변수를 읽고 수정하고 있다. 이런 경우 컴포넌트를 여러 번 호출하면 다른 JSX를 생성한다는 것을 의미한다. 해당 컴포넌트가 `guest` 를 읽고 다른 컴포넌트 또한 `guest` 를 읽어서 렌더링 한다면 해당 컴포넌트 또한 다른 JSX를 생성할 것이고 결과를 예측하기 어려워진다. 이런 코드는 버그를 발생시킬 수 있다.

`guest` 변수를 프로퍼티로 넘겨 줌으로써 컴포넌트를 순수하게 작성할 수 있다.

```jsx
function Cup({ guest }) {
 return <h2>Tea cup for guest #{guest}</h2>;
}

export default function TeaSet() {
 return (
  <>
   <Cup guest={1} />
   <Cup guest={2} />
   <Cup guest={3} />
  </>
 );
}
```

Strict Mode로 순수하지 않은 함수 감지

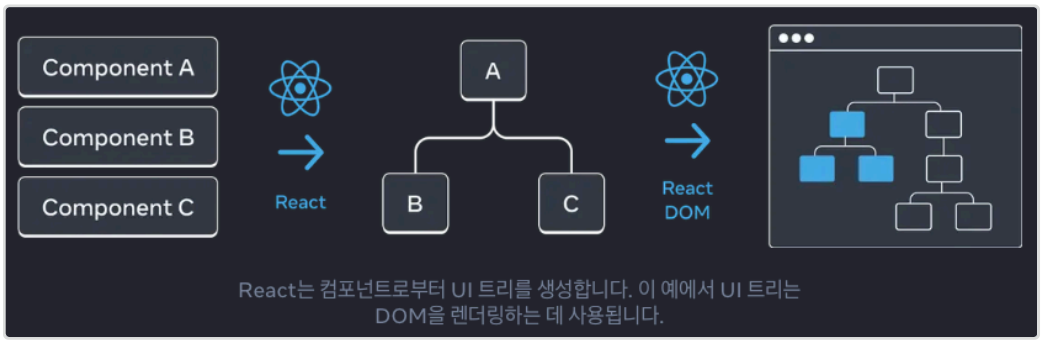
React는 개발 중에 각 컴포넌트의 함수를 두 번 호출하는 Strict Mode를 제공한다. 컴포넌트 함수를 두 번 호출함으로써 순수 함수 규칙을 위반하는 컴포넌트를 찾는데 도움을 받을 수 있다. (순수 함수는 N번 호출해도 항상 같은 결과를 반환하기 때문에 아무 것도 변경하지 않는다.) React의 Strict Mode는 최상단 컴포넌트를 <React.StrictMode>로 감싸면 된다. Strict Mode는 프로덕션에 영향을 주지 않는다.

Components Tree

React 앱은 서로 중첩된 다수의 컴포넌트로 구성되어 있다. React는 UI를 트리로 모델링한다.

UI Tree

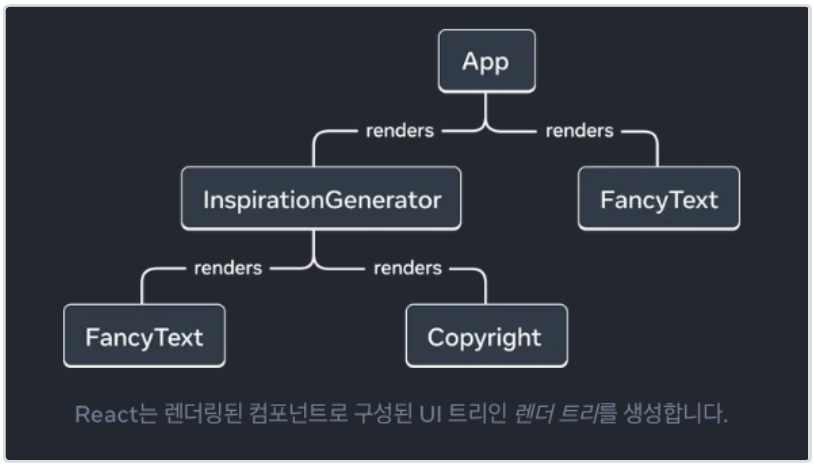
Tree는 Component와 UI 사이의 관계 모델이며 UI는 종종 Tree 구조로 표현된다. 예를 들어, 브라우저는 HTML(DOM)과 CSS(CSSOM)를 모델링하기 위해 트리 구조를 사용한다.



Render Tree

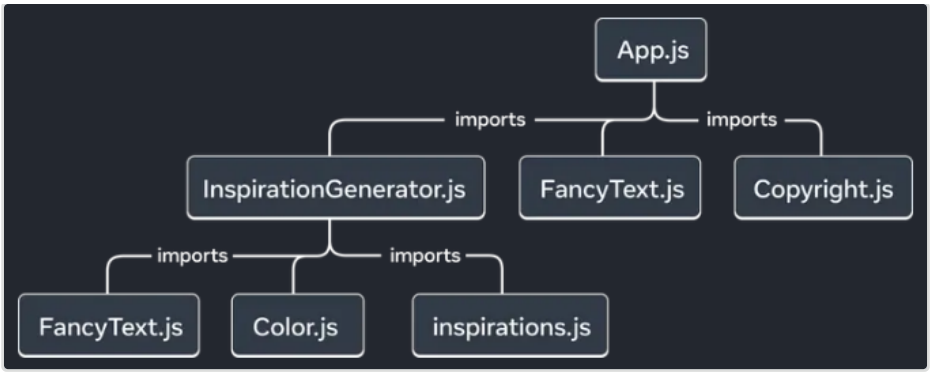
컴포넌트의 특징은 다른 컴포넌트를 구성한다는 것이다. 컴포넌트를 중첩하면 부모 자식 관계가 생기며 각 부모 컴포넌트 또한 다른 컴포넌트의 자식이 될 수 있다. React 앱을 렌더링할 때 컴포넌트 관계를 Render Tree라고 알려진 Tree로 모델링할 수 있다.

```
export default function App() {
  return (
    <>
      <FancyText title text="Get Inspired App" />
      <InspirationGenerator>
        <Copyright year={2004} />
      </InspirationGenerator>
    </>
  );
}
```



모듈 의존성 트리

트리로 모델링 할 수 있는 React 앱의 다른 관계는 앱의 모듈 의존성이다. 컴포넌트를 분리하고 로직을 별도의 파일로 분리하면 컴포넌트, 함수 또는 상수를 내보내는 JS 모듈을 만들 수 있다. 모듈 의존성 트리의 각 노드는 모듈이며, 각 가지는 해당 모듈의 `import` 문을 나타낸다.



참고 자료 :

- 컴포넌트 순수하게 유지하기 – React
- 트리로서 UI 이해하기 – React