React State basic

개요

컴포넌트는 상호 작용의 결과로 화면 내용을 변경하는 경우가 많다. 예를 들어, Form에 입력하면 입력 필드가 업데이트되고 이미지 갤러리에서 다음을 누르면 표시되는 이미지가 벼경되거나, 상품 구매 버튼을 클릭하면 상품이 장바구니에 담겨야 한다. 컴포넌트는 현재 입력값을 기억해야 하는데, React는 이런 종류의 컴포넌트별 메모리를 state라고 부른다.

일반 변수의 한계

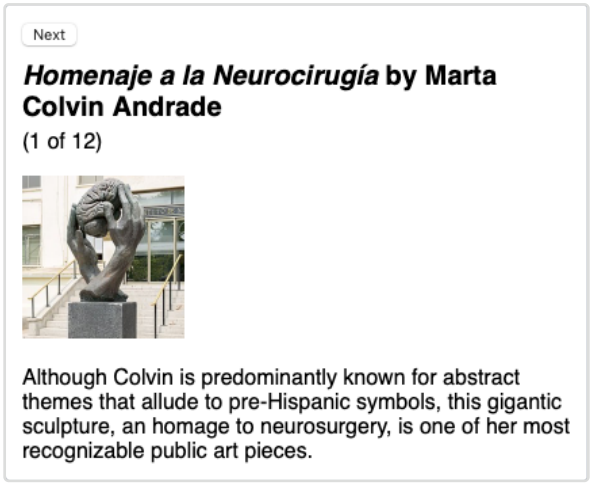
예를 들어, 아래 예제와 같이 Next 버튼을 클릭하면 다음 Index 이미지를 보여주는 코드가 있다고 가정하자.

```
export default function Gallery() {
  let index = 0;

  function handleClick() {
    index = index + 1;
  }

  let sculpture = sculptureList[index];
  return (
    <>
      <button onClick={handleClick}>
        Next
      </button>
      <h2>
        <i>{sculpture.name} </i>
        by {sculpture.artist}
      </h2>
      <h3>
        ({index + 1} of {sculptureList.length})
      </h3>
      <img
        src={sculpture.url}
        alt={sculpture.alt}
      />
      <p>
        {sculpture.description}
```

```
      </p>
    </>
  );
}
```



그러나 위 코드는 정상적으로 동작하지 않는다. (Next 버튼을 클릭해도 아무 일도 발생하지 않는다.) `handleClick` 이벤트 핸들러는 지역 변수 `index`를 업데이트하고 있는데 정상 동작하지 않는 이유는 두 가지가 있다.

- 지역 변수는 렌더링 발생 시 유지되지 않는다. React는 해당 컴포넌트를 두 번째로 업데이트할 때 지역 변수에 대한 변경 사항은 고려하지 않고 처음부터 렌더링한다.
- 지역 변수가 변경되어도 렌더링은 발생하지 않는다. React는 새로운 데이터로 컴포넌트를 다시 렌더링해야 한다는 것을 인식할 수 없다.

결과적으로 위 문제점은 다음과 같은 결론에 도달한다.

- 렌더링 사이에 데이터를 유지해야 한다.
- React가 새로운 데이터로 컴포넌트를 렌더링하도록 유발해야 한다.

`useState` React hook은 이 두 가지가 가능하다.

- 렌더링 사이에 데이터를 유지하기 위한 **state 변수**
- React 컴포넌트를 다시 렌더링하도록 유발하는 **state setter 함수**

state 변수

state 변수를 추가하기 위해 파일 최상단에 React `useState`를 import 한다.

```
import { useState } from 'react';
```

앞에서 선언했던 `index` 지역 변수를 아래와 같은 코드로 변경한다. 새로 작성한 코드에서 `index`는 변수이고 `setIndex`는 setter 함수이다.

```
const [index, setIndex] = useState(0);
```

이제 함수를 다시 실행하면 정상 동작하는 것을 확인할 수 있다.

```
export default function Gallery() {
  const [index, setIndex] = useState(0);

  function handleClick() {
```

```
    setIndex(index + 1);
  }

  let sculpture = sculptureList[index];
  return (
    <>
      <button onClick={handleClick}>
        Next
      </button>
      <h2>
        <i>{sculpture.name} </i>
        by {sculpture.artist}
      </h2>
      <h3>
        ({index + 1} of {sculptureList.length})
      </h3>
      <img
        src={sculpture.url}
        alt={sculpture.alt}
      />
      <p>
        {sculpture.description}
      </p>
    </>
  );
}
```

React hook

useState 와 같이 `use`로 시작하는 함수를 Hook이라고 부른다. Hook은 React가 렌더링 중일 때만 사용할 수 있는 특별한 함수를 일컫는다. 이를 통해 다양한 React 기능을 연결할 수 있다.

> Hook은 컴포넌트의 최상위 수준 혹은 Custom Hook 안에서만 호출할 수 있다. 조건문, 반복문 등 기타 중첩 함수 내부에서는 Hook을 호출할 수 없다.

useState

`useState`를 호출하는 이유는 React에 특정 컴포넌트가 어떤 데이터를 갖고 있기를 원한다고 알려주는 것이다.

```
const [index, setIndex] = useState(0);
```

위와 같은 경우 React가 `index`변수를 기억하기를 원한다. `useState`의 유일한 인수는 state 변수의 초깃값이다. 실제 작동 방식은 다음과 같다.

1. 컴포넌트가 처음 렌더링된다. `index`의 초깃값으로 `useState`를 사용해 0을 전달했기 때문에 `[o, setIndex]`를 반환한다. React는 `state`의 최신 값을 0으로 기억한다.
2. `state`를 업데이트 한다. 사용자의 상호작용으로 `setIndex(index + 1)`을 호출한다. 이 때 React는 `index`를 1로 변경하여 기억하고 다른 렌더링을 유발한다.
3. 컴포넌트가 두 번째로 렌더링된다. React는 `state`변수를 이제 `[1, setIndex]`라고 기억한다.
4. 반복

> **React Hook에 대한 아이디어**
>
> ```
> let componentHooks = [];
> let currentHookIndex = 0;
> ```

```
3
4  // How useState works inside React (simplified).
5  function useState(initialState) {
6    let pair = componentHooks[currentHookIndex];
7    if (pair) {
8      // This is not the first render,
9      // so the state pair already exists.
10     // Return it and prepare for next Hook call.
11     currentHookIndex++;
12     return pair;
13   }
14
15   // This is the first time we're rendering,
16   // so create a state pair and store it.
17   pair = [initialState, setState];
18
19   function setState(nextState) {
20     // When the user requests a state change,
21     // put the new value into the pair.
22     pair[0] = nextState;
23     updateDOM();
24   }
25
26   // Store the pair for future renders
27   // and prepare for the next Hook call.
28   componentHooks[currentHookIndex] = pair;
29   currentHookIndex++;
30   return pair;
31 }
```

State의 지역성

State는 컴포넌트 인스턴스에 지역적일 수 있다. 풀어서 이야기하면 동일한 컴포넌트에서 두 번 렌더링 한다고 해도 각 복사본은 완전히 격리된 서로 다른 State를 갖는다.

```
1  export default function Gallery() {
2    const [index, setIndex] = useState(0);
3    const [showMore, setShowMore] = useState(false);
4
5    function handleNextClick() {
6      setIndex(index + 1);
7    }
8
9    function handleMoreClick() {
10     setShowMore(!showMore);
11   }
12   ...
13 }
```

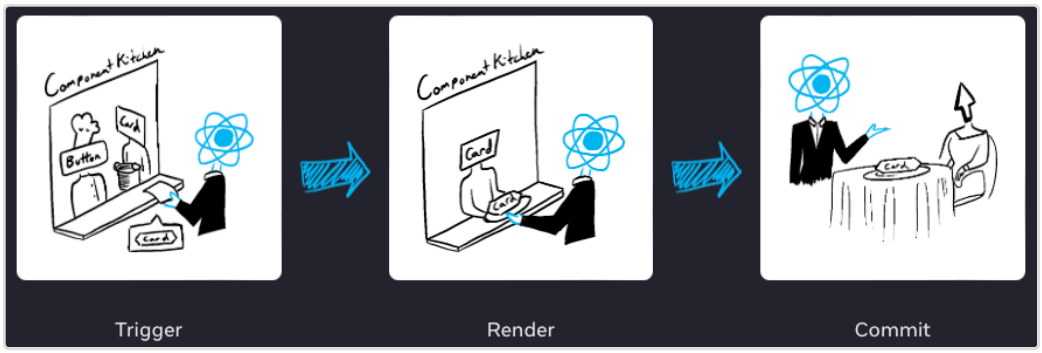
State가 `Gallery()` 안에서 지역적으로 선언되었기 때문에 별도로 저장되며 해당 State에 대해서 다른 컴포넌트는 알 수 없다. 만약 하나의 동기화된 State가 필요하다면 가장 가까운 공통 부모 컴포넌트에 State를 추가하여 사용할 수 있다.

렌더링 트리거

컴포넌트를 화면에 표시하기 전에 React에서 렌더링을 해야 한다. 총 세 가지 과정을 거친다.

1. 렌더링 트리거 (주문을 주방으로 전달)

2. 컴포넌트 렌더링 (주방에서 주문 준비하기)
3. DOM에 커밋 (테이블에 주문 요리 내놓기)



1단계 : 렌더링 트리거

컴포넌트 렌더링이 발생하는 데에는 두 가지 이유가 있다.

1. 컴포넌트의 초기 렌더링인 경우
2. 컴포넌트의 state가 업데이트된 경우

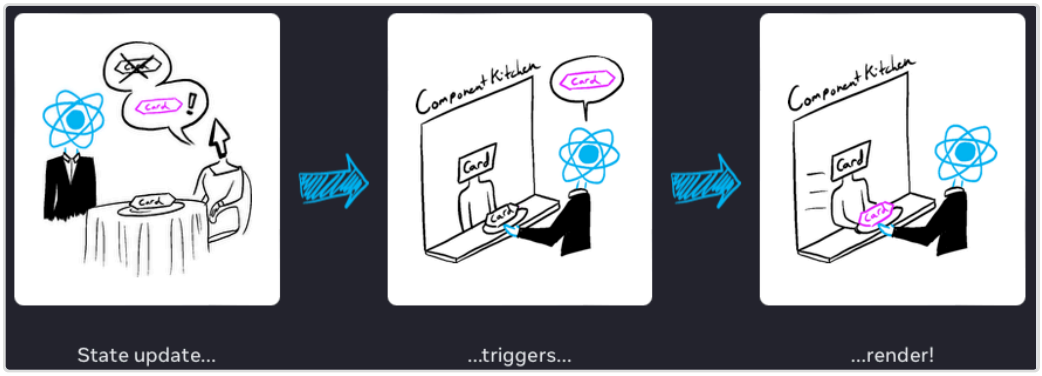
초기 렌더링

앱을 시작할 때 초기 렌더링을 트리거 해야 한다. 대상 DOM 노드와 함께 `createRoot`를 호출한 다음 해당 컴포넌트로 `render` 메서드를 호출하면 이 작업이 완료된다.

```
const root = createRoot(document.getElementById('root'))
root.render(<Image />);
```

state 업데이트 렌더링

컴포넌트가 렌더링 된 후 `set`함수를 통해서 상태를 업데이트하면 추가 렌더링 작업을 트리거 할 수 있다. 컴포넌트의 상태를 업데이트 하면 자동으로 렌더링 대기열에 추가된다.



2단계 : React Component rendering

렌더링을 트리거 한 후 React는 컴포넌트를 호출하여 화면에 표시할 내용을 파악한다. 렌더링은 React에서 컴포넌트를 호출하는 작업이다.

- 초기 렌더링에서 React는 루트 컴포넌트를 호출한다.
- 이후 렌더링에서 React는 state 업데이트가 일어나 렌더링을 트리거한 컴포넌트를 호출한다.

업데이트된 컴포넌트가 다른 컴포넌트를 반환하면 React는 해당 컴포넌트를 다음으로 렌더링한다. 해당 컴포넌트 또한 컴포넌트를 반환하면 반환된 컴포넌트를 다음으로 렌더링하며 이 작업은 더 이상 중첩된 컴포넌트가 없을 때까지 반복된다.

3단계 : React가 변경사항을 DOM에 커밋

컴포넌트를 렌더링한 후 React는 DOM을 수정한다. React는 렌더링 간에 차이가 있는 경우에만 DOM 노드를 변경한다.

- 초기 렌더링의 경우 React는 `appendChild()` DOM API를 사용하여 생성한 모든 DOM 노드를 화면에 출력한다.
- 리렌더링의 경우 React는 필요한 최소한의 작업을 적용하여 DOM이 최신 렌더링 출력과 일치하도록 한다.

참고 자료 :

State: 컴포넌트의 기억 저장소 – React

구조 분해 할당