리액트 중간 정리: 6-7

Chapter 6: State와 생명주기

State

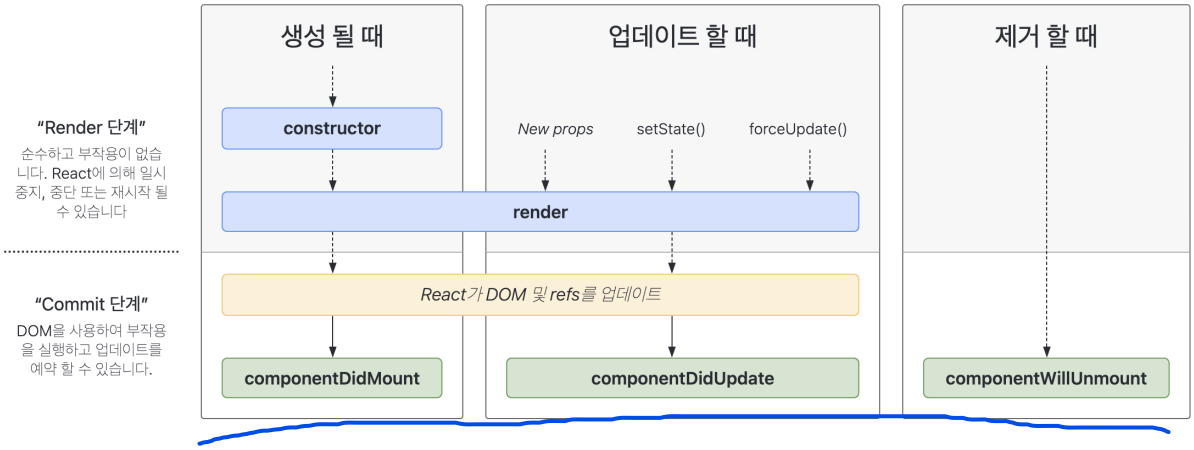
- State란?
  - 리액트 컴포넌트의 변경 가능한 데이터
  - state는 개발자가 자유롭게 정의할 수 있으며, 일반적인 자바스크립트 객체임
  - 어떤 컴포넌트에만 한정하여 사용되는 데이터를 포함, 해당 데이터는 시간이 지남에 따라 변경될 수 있음
  - state가 변경될 경우 컴포넌트가 재랜더링됨
  - 랜더링이나 데이터 흐름에 사용되지 않는 값은 state에 넣지 않아야 함
  - state 정의 후 직접 수정할 수 없고, setState(), useState() 함수를 사용
- 클래스 컴포넌트에서의 state
  - 생성자에서 모든 state를 한번에 정의
  - state를 변경할 때는 setState( ) 함수를 사용해야 함
  - setState()
    - 컴포넌트 state의 변경 사항을 대기열에 집어넣고, React에게 해당 컴포넌트와 그 자식들이 갱신된 state를 사용하여 다시 렌더링되어야 한다고 알림
    - 이벤트 핸들러와 서버 응답 등에 따라 UI를 갱신할 때 주로 사용
- 함수 컴포넌트에서의 state
  - useState() 훅을 사용하여 각각의 state를 정의
  - 각 state별로 주어지는 set함수를 사용하여 state 값을 변경
- State 업데이트는 비동기적일 수 있음
  - 컴포넌트가 즉각적으로 갱신되는 것을 보장하지 않음 (여러 변경 사항과 함께 일괄적으로 갱신하거나 나중으로 미뤄질 수 있음)

- setState()를 호출하자마자 this.state에 접근하는 것은 바람직하지 않음 → componentDidUpdate() 또는 setState의 콜백을 사용

```
componentDidUpdate(prevProps, prevState) {
  if (this.state.num !== prevState.num) {
    console.log("state : " + this.state.num);
  }
}
```

생명주기(Lifecycle)

- 클래스 컴포넌트의 생명주기 함수



- Mount 단계
  - constructor()

    - 컴포넌트가 마운트되기 전에 호출됨.

    다른 구문에 앞서 super(props)를 호출해야 this.props가 정상적으로 정의됨

    this.state에 객체를 할당하여 state 초기화

    이벤트 처리 메서드를 바인딩 - bind()

  - render()

    - 클래스 컴포넌트에서 반드시 구현돼야하는 유일한 메서드

    React 엘리먼트를 반환

    컴포넌트의 state를 변경하지 않고, 호출될 때마다 동일한 결과를 반환하며, 브라우저와 직접적으로 상호작용하지 않음

    - componentDidMount()

      - 컴포넌트 마운트 직후(DOM 트리에 삽입된 직후) 호출됨

      DOM 노드와 관련된 초기화 작업, 외부 데이터 요청(네트워크) 수행

      데이터 구독 설정 등의 작업 수행 (구독 해제는 componentWillUnmount())

- Update 단계
    - render()
    - componentDidUpdate()

      - 갱신이 일어난 직후에 호출됨. 컴포넌트 갱신 시 DOM을 조작하기 위해 활용

      이전과 현재의 props를 비교하여 네트워크 요청 등의 작업 수행

- Unmount 단계
    - componentWillUnmount()

      - 컴포넌트가 마운트 해제되어 제거되기 직전에 호출됨

      타이머 제거, 네트워크 요청 취소, 구독 해제 등 정리 작업 수행

Chapter 7: Hook

Hook 개요

- Hook은 함수 컴포넌트에서 React state와 생명주기 기능(lifecycle features)을 “연동(hook into)“할 수 있게 해주는 특별한 함수
- Hook은 class 안에서는 동작하지 않음
- React 16.8.0에서 추가된 기능
- Hook이 추가되기 전에는 함수 컴포넌트에 state를 추가하고 싶을 때 클래스 컴포넌트로 바꿔야 했지만, 이제는 Hook을 이용하여 state를 사용할 수 있게 됨

State Hook

- useState()
    - 함수 컴포넌트에서 state를 사용할 수 있게 해주는 Hook

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

function Example() {
  // 새로운 state 변수 count를 선언하고 0으로 초기화
  const [count, setCount] = useState(0);
  // 여러 개의 state 변수를 선언할 수 있음
  const [fruit, setFruit] = useState('banana');
```

- useState() 호출
  - state 변수를 선언, state 변수 이름은 개발자가 정의
  - 클래스 컴포넌트의 this.state와 동일한 기능
  - 일반 변수는 함수가 끝날 때 사라지지만, state 변수는 React에 의해 사라지지 않음
- useState()의 인자
  - state의 초기값
  - 숫자, 문자 타입을 가질 수 있음
- useState()의 리턴값
  - state 변수, 해당 변수를 업데이트할 수 있는 함수 쌍을 반환
  - 클래스 컴포넌트의 this.state.count, this.setState()와 유사
- state 가져오기
  - state 변수 직접 사용 - `<p>You clicked {count} times</p>`
  - 클래스 컴포넌트는 this.state.count를 사용 - `<p>You clicked {this.state.count} times</p>`
- state 갱신하기
  - setCount 함수 사용하여 state 변수를 갱신

```
<button onClick={() => setCount(count + 1)}>Click me</button>
```

  - 클래스 컴포넌트는 this.setState()를 호출

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>
Click me
</button>
```

- useState() Hook 예제

```
import React, { useState } from 'react'; // useState Hook을 React에서 가져옴

function Example() {
  // useState Hook을 사용하여 state변수와 set함수 생성, count 변수를 0으로 초기화
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      /* 사용자가 버튼을 클릭하면 setCount 함수 호출하여 state 변수 갱신
      React는 새로운 count 변수를 Example 컴포넌트에 넘기며 해당 컴포넌트를 리렌더링 */
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

> 배열 구조 분해 - 배열을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식
>
> ```
> var foo = ["one", "two", "three"];
>
> var [red, yellow, green] = foo;
> console.log(red); // "one"
> console.log(yellow); // "two"
> console.log(green); // "three"
> ```

Effect Hook

- useEffect()
  - 함수 컴포넌트에서 side effect를 수행할 수 있게 해주는 Hook
  - 클래스 컴포넌트에서 제공하는 생명주기 함수 componentDidMount(), componentDidUpdate(), ComponentWillMount()와 동일한 기능을 하나로 통합해서 제공

- useEffect Hook을 이용하여 React에게 컴포넌트가 렌더링 이후에 수행할 일을 알려줌
- React는 DOM 업데이트를 수행한 이후에 useEffect 함수를 호출
-

- side effects
  - 데이터 가져오기, 구독(subscription) 설정하기, 수동으로 React 컴포넌트의 DOM 수정 등의 작업
  - 정리(Clean-up)를 이용하지 않는 Effects
    - 네트워크 리퀘스트, DOM 수동 조작, 로깅 등은 정리(clean-up)가 필요 없는 경우, 이러한 예들은 실행 이후 신경 쓸 것이 없음

```
function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });
}
```

    - 기본적으로 첫번째 렌더링과 이후의 모든 업데이트에서 수행됨
  - 정리(Clean-up)를 이용하는 Effects
    - 외부 데이터에 구독(subscription)을 설정해야 하는 경우, 이런 경우에 메모리 누수가 발생하지 않도록 정리(clean-up)하는 것이 중요

```
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    // effect 이후에 어떻게 정리(clean-up)할 것인지 표시
    return function cleanup() {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });
```

    - 정리를 위한 함수를 반환

      - effect를 위한 추가적인 정리(clean-up) 메커니즘

      named function 또는 arrow function을 반환할 수 있음
    - React는 컴포넌트가 마운트 해제되는 때에 정리(clean-up)를 실행
    - 관심사 분리

        - Hook을 이용하면 생명주기 메소드에 따라서가 아니라, 코드가 무엇을 하는지에 따라 나눌 수 있음
        - 구독의 추가와 제거가 하나의 effect 구성
        -
    - Effect를 건너뛰어 성능 최적화하기
        - 모든 렌더링 이후에 effect를 정리(clean-up)하거나 적용하는 것이 때때로 성능 저하를 발생시킴
        - 클래스 컴포넌트의 경우, componentDidUpdate에서 prevProps나 prevState와의 비교를 통해 이러한 문제를 해결
        - 특정 값들이 리렌더링 시에 변경되지 않으면 effect를 건너뛰도록 설정

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]); // count가 바뀔 때만 effect를 재실행
```

        - effect를 마운트와 마운트 해제 시 한 번씩만 실행하고 싶을 경우, 빈 배열([])을 두 번째 인자로 넘김

Hook의 규칙

- 최상위(Top Level)에서만 Hook을 호출해야 함
    - 반복문, 조건문 혹은 중첩된 함수 내에서 hook을 호출하면 안됨
    - 컴포넌트가 렌더링 될 때마다 항상 동일한 순서로 Hook이 호출되는 것이 보장됨
        - Hook의 상태를 올바르게 유지할 수 있도록 해줌
    - 조건부 effect를 실행하려면 조건문을 Hook 내부에 적용
- React 함수 컴포넌트에서 Hook 호출
    - 함수 컴포넌트, Custom Hook에서만 호출

나만의 Hook 만들기

- 사용자 정의 Hook (Custom Hook)
    - 리액트에서 기본 제공되는 훅 이외에 추가로 필요한 기능을 직접 만들어서 사용

    - 여러 컴포넌트에서 반복적으로 사용되는 로직을 함수로 뽑아내어 재사용 할 수 있음
    - 사용자 정의 Hook은 이름이 use로 시작하는 자바스크립트 함수. 내부에서 다른 Hook을 호출
- Custom Hook 규칙
    - 커스텀 훅은 특별한 규칙이 없음
        - 파라미터, 리턴 값을 개발자가 직접 정의해서 사용
    - 이름은 use로 시작해야 함
        - 이를 따르지 않으면 특정한 함수가 그 안에서 Hook을 호출하는지를 알 수 없기 때문에 Hook 규칙의 위반 여부를 자동으로 체크할 수 없음
- 사용자 정의 Hook 추출하기
    - 두 개의 자바스크립트 함수에서 같은 로직을 공유하고자 할 때는 또 다른 함수로 분리
    - 컴포넌트와 Hook 또한 함수이기 때문에 같은 방법을 사용하여 다른 함수로 분리