useEffect 조지기

부제: 그러나 조져지는 건 나였다

React 컴포넌트와 바깥 세상의 연결고리

- "React 컴포넌트가 렌더링 결과(화면에 보이는 것) 외에, 브라우저 탭의 제목을 바꾸거나 채팅 서버에 연결하는 등 '바깥 세상'과 소통해야 할 땐 어떻게 할까요?"

React 컴포넌트와 바깥 세상의 연결고리

- "React 컴포넌트가 렌더링 결과(화면에 보이는 것) 외에, 브라우저 탭의 제목을 바꾸거나 채팅 서버에 연결하는 등 '바깥 세상'과 소통해야 할 땐 어떻게 할까요?"
- useEffect는 컴포넌트의 상태(state)와 속성(props)을 기준으로 외부 시스템과 동기화(synchronize)하는 도구

React 컴포넌트와 바깥 세상의 연결고리

- useEffect의 기본 개념, 이벤트 핸들러와의 차이점, 그리고 '동기화'라는 핵심 원리 살짝 이해하자!

1. "Effect"란 무엇일까요? 사이드 이펙트와 동기화의 개념

- useEffect는 “리액트 컴포넌트의 생명주기에 맞춰 부수 효과(side effect)를 실행하기 위한 훅”
- React에서 'Effect' 또는 'Side Effect(부수 효과)'란, 컴포넌트의 주된 임무인 UI 렌더링 중에 하면 안 되는 작업을 의미

1. "Effect"란 무엇일까요? 사이드 이펙트와 동기화의 개념

```
function Greeting({ name }) {
  useEffect(() => {
    document.title = '안녕하세요, ' + name;
  });
  return <h1>안녕하세요, {name}</h1>;
}
```

- **React의 내부 세계:** <h1> 태그는 React가 직접 관리하고 렌더링하는 DOM의 일부입니다. React는 name prop이 바뀔 때마다 <h1>의 내용을 알아서 업데이트합니다.
- **외부 세계:** 브라우저 탭의 제목(document.title)은 React가 직접 관리하지 않는 영역입니다. React는 document.title이 무엇인지, 어떻게 바뀌는지 알지 못합니다.

1. "Effect"란 무엇일까요? 사이드 이펙트와 동기화의 개념

- useEffect는 바로 이 두 세계를 연결하는 다리 역할
- 컴포넌트의 name prop(내부 상태)이 바뀔 때마다, 이와 관련된 외부 세계(브라우저 탭 제목)를 최신 상태로 동기화

2. 렌더링과 동기화: useEffect의 핵심 원리

- "모든 렌더링은 자신만의 값을 가진다”
- 컴포넌트가 렌더링될 때, 그 시점의 props와 state 값을 '사진'처럼 찍어두고(캡처하고) 그 값은 해당 렌더링 주기 안에서는 절대 변하지 않는다!

2. 렌더

원리

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

  const showAlert = () => {
    setTimeout(() => {
      alert('클릭 시점의 count: ' + count);
    }, 3000);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
      <button onClick={showAlert}>3초 후 알림 표시</button>
    </div>
  );
}
```

2. 렌더링과 동기화: useEffect의 핵심 원리

- React는 매 렌더링마다 새로운 Effect 함수를 기억
- useEffect는 기본적으로 모든 렌더링이 끝나고 화면 업데이트가 완료된 후에 실행
- React가 화면을 그리는 중요한 작업을 먼저 마친 후, 부수적인 동기화 작업을 처리하여 사용자 경험을 해치지 않도록 보장하기 위함

2. 렌더링과 동기화: useEffect의 핵심 원리

- React는 각 렌더링을 독립적인 스냅샷으로 취급함
- → Effect가 특정 시점의 상태를 기준으로 외부 세계와 정확하게 동기화할 수 있도록 보장

2. 렌더링과 동기화: useEffect의 핵심 원리

- 사용자가 버튼을 클릭했을 때 특정 작업을 수행하는 것도 외부와 상호작용하는 건데?

3. 이벤트 핸들러 vs useEffect: 언제 무엇을 써야 할까요?

- 코드가 실행되어야 하는 이유가 무엇인가?

구분	이벤트 핸들러 (Event Handler)	`useEffect`
실행 이유	**사용자의 특정 상호작용** (예: 버튼 클릭, 입력 등) 때문에 실행됩니다.	**컴포넌트가 화면에 렌더링**되었기 때문에 실행됩니다.
핵심 예시	'구매' 버튼을 눌렀을 때, `product`를 장바구니에 추가하는 `addToCart(product)` API를 호출합니다.	채팅방 컴포넌트가 화면에 나타났을 때 채팅 서버에 연결합니다.
정신 모델	"사용자가 **이것을 했기 때문에** 저것을 실행한다."	"화면이 **이렇게 보이기 때문에** 이것과 동기화한다."

4. useEffect 심화: 동기화 제어하기

- 채팅방 컴포넌트가 화면에서 사라졌는데도 서버 연결을 끊지 않으면 메모리 누수나 버그의 원인

→ 정리하는 과정 필요

```jsx
import { useEffect } from 'react';
import { createConnection } from './chat.js';

export default function ChatRoom() {
  useEffect(() => {
    const connection = createConnection();
    connection.connect();
  }, []);
  return <h1>채팅에 오신걸 환영합니다!</h1>
}
```

제어하기

끊지 않으면 메모리 누

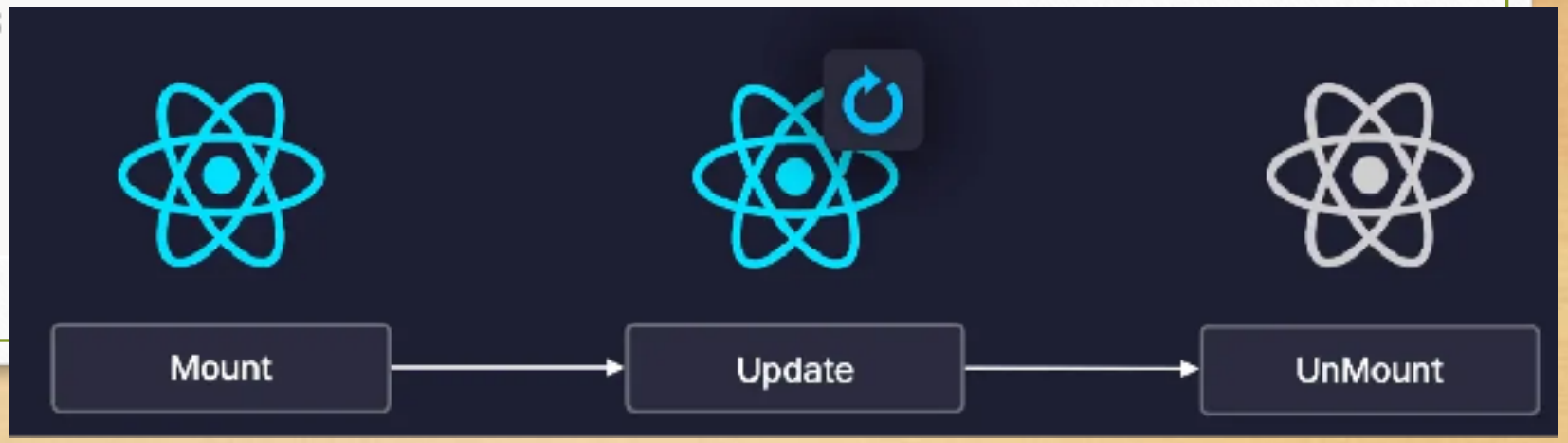
```jsx
export function createConnection() {
  // 실제 구현은 정말로 채팅 서버에 연결하는 것이 되어야 합니다.
  return {
    connect() {
      console.log('✅ 연결 중...');
    },
    disconnect() {
      console.log('❌ 연결이 끊겼습니다.');
    }
  };
}
```

4. useEffect 심화: 동기화 제어하기

```
useEffect(() => {
  // 1. 구독 설정 (Effect 실행)
  const connection = createConnection(serverUrl, roomId);
  connection.connect();

  // 2. 구독 해제 (Cleanup 실행)
  return () => {
    connection.disconnect();
  };
}, [roomId, serverUrl]);
```

- 클린업 함수는,
  - 다음 렌더링의 Effect가 실행되기 전에 실행
- 컴포넌트가 언마운트될 때 마지막으로 한 번 더 실행



4. useEffect 심화: 동기화 제어하기

- 기본적으로 useEffect는 매 렌더링마다 실행
- 때로는 특정 값이 변경될 때만 동기화를 다시 실행하고 싶을 수 있음

→ **의존성 배열**

4. useEffect 심화: 동기화 제어하기

- **배열 생략 시:** 모든 렌더링 후에 Effect가 실행됩니다.
- **[] (빈 배열):** 컴포넌트가 처음 화면에 나타날 때(마운트) **한 번만** 실행됩니다. 초기 설정에 주로 사용됩니다.
- **[dep1, dep2, ...]:** 배열 안의 의존성 중 하나라도 이전 렌더링과 값이 다를 때 Effect가 다시 실행됩니다.

5. useEffect가 필요 없는 경우: 흔한 실수 피하기

- 1. 렌더링 중 계산 가능한 값 (파생 상태)
  - firstName과 lastName state를 조합해 fullName을 만들어야 한다고 가정

5. useEffect가 필요 없는 경우: 흔한 실수 피하기

```
// 🔴 안티패턴: 불필요한 Effect와 state
const [fullName, setFullName] = useState('');
useEffect(() => {
  setFullName(firstName + ' ' + lastName);
}, [firstName, lastName]);

// ✅ 좋은 패턴: 렌더링 중 직접 계산
const fullName = firstName + ' ' + lastName;
```

- fullName에 대한 오래된 값으로 전체 렌더링 패스를 수행한 다음, 업데이트된 값으로 다시 렌더링

5. useEffect가 필요 없는 경우: 흔한 실수 피하기

- 2. Prop 변경에 따른 State 초기화
  - userId라는 prop이 바뀔 때마다 댓글 입력창(comment state)을 비워야 하는 경우

```jsx
export default function ProfilePage({ userId }) {
  const [comment, setComment] = useState('');

  // 🔴 피하세요: Effect에서 prop 변경 시 state 초기화
  useEffect(() => {
    setComment('');
  }, [userId]);
  // ...
}
```

없는 경우:
기

- 컴포넌트에 key={userId}를 전달하는 패턴을 사용
- key가 변경되면, React는 이전 컴포넌트 인스턴스를 파괴하고 완전히 새로운 인스턴스를 생성
- 내부의 모든 state가 자동으로 초기화

```jsx
export default function ProfilePage({ userId }) {
  // Profile 컴포넌트에 userId를 key로 전달
  return <Profile userId={userId} key={userId} />;
}

function Profile({ userId }) {
  // key가 변경되면 이 컴포넌트의 모든 state는 자동으로 리셋됩니다.
  const [comment, setComment] = useState('');
  // ...
}
```

5. useEffect가 필요 없는 경우: 흔한 실수 피하기

- 3. 사용자 이벤트 처리
  - 알림을 띄우거나, API에 데이터를 전송하는 등의 작업이 사용자의 '클릭'이나 '제출' 같은 특정 행동의 결과라면, 해당 로직은 이벤트 핸들러 함수 안에 있어야

결론: "동기화" 관점으로 useEffect 바라보기

- useEffect는 '컴포넌트가 마운트될 때', '업데이트될 때' 실행되는 도구가 아니라, "React 컴포넌트의 상태를 외부 세계와 일치시키는 도구"

결론: "동기화" 관점으로 useEffect 바라보기

- "이 코드는 왜 실행되어야 하는가?
- 특정 상호작용 때문인가, 아니면 화면에 무언가가 표시되었기 때문인가?
- 그리고 이 코드가 React 컴포넌트의 상태와 외부 시스템을 일치시키려고 하는가?"