**06\_useEffect 훅 정리**

**1. 기본 구조**

|  |
| --- |
| **useEffect(() => {**  **// 실행할 코드**  **});** |

1. **모든 렌더링마다 실행**됨.
2. DOM 업데이트 직후 실행되며, 비동기적으로 동작.
3. 주로 **렌더링 후 처리**가 필요한 코드 작성.

**2. 의존성 배열(dependency array)**

* 두 번째 인자로 의존성 배열을 받음.
* [] (빈 배열) → 마운트 시 1번만 실행.
* [value] → 특정 상태값이 변경될 때마다 실행.

**→ 클래스형 컴포넌트의 생명주기 메서드와 대응**

* **componentDidMount → useEffect(..., [])**
* **componentDidUpdate → useEffect(..., [deps])**

**3. Cleanup 함수**

|  |
| --- |
| **useEffect(() => {**  **const timer = setInterval(() => {**  **console.log("타이머 실행");**  **}, 1000);**  **// cleanup**  **return () => {**  **clearInterval(timer);**  **};**  **}, []);** |

1. **함수 반환** → 다음 effect 실행 전이나 컴포넌트 언마운트 시 호출.
2. 이벤트 핸들러 제거, 타이머 해제, 리소스 정리 같은 작업에 사용.
3. 클래스형의 componentWillUnmount에 해당.

**4. 사용 예시**

(1) 로컬스토리지 동기화

|  |
| --- |
| const [todos, setTodos] = useState([]);  useEffect(() => {  localStorage.setItem("todos", JSON.stringify(todos));  }, [todos]); |

상태가 바뀔 때마다 자동으로 저장.

(2) API 호출

|  |
| --- |
| useEffect(() => {  fetch("/api/data")  .then(res => res.json())  .then(data => setItems(data));  }, []); |

최초 마운트 시 데이터 가져오기.

비동기 요청과 결합해 사용.

(3) 이벤트 리스너 등록/해제

|  |
| --- |
| useEffect(() => {  const handleResize = () => console.log(window.innerWidth);  window.addEventListener("resize", handleResize);  return () => window.removeEventListener("resize", handleResize);  }, []); |

#### (4) 타이머/D-Day 구현

|  |
| --- |
| useEffect(() => {  const id = setInterval(() => setNow(new Date()), 1000);  return () => clearInterval(id);  }, []); |

 컴포넌트 살아있는 동안 매 초 갱신.

 언마운트 시 메모리 누수 방지.