Lifecycle of Reactive Effects – React JS Bangla Tutorial

react-bangla.vercel.app/learn-react-escape-hatches/lifecycle-of-reactive-effects Updated on: August 23, 2025

Effect-এর জীবনচক্র

React-এর প্রতিটি Component-এর একটি জীবনচক্র থাকে, যা তিনটি ধাপে বিভক্ত:

1. **Mount** (যখন প্রথম স্ক্রিনে আসে),
2. **Update** (যখন state বা props বদলায়) এবং
3. **Unmount** (যখন স্ক্রিন থেকে চলে যায়)।

কিন্তু `useEffect` হুককে এভাবে ভাবা ঠিক নয়। Effect-এর জীবনচক্র এর চেয়েও সহজ। এর কাজ শুধু দুটো:

1. **কিছু একটা শুরু করা** (Start synchronization)
2. **সেটা আবার বন্ধ করা** (Stop synchronization)

এই শুরু এবং বন্ধ হওয়ার চক্রটি useEffect এ একবার বা একাধিকবার ঘটতে পারে। State এবং Props চেঞ্জের মাধ্যমে।

একটি চ্যাটরুম

ভাবুন, আপনি একটি চ্যাটরুম বানাচ্ছেন। যখন একজন ইউজার কোনো রুমে প্রবেশ করে, তখন আমাদের সার্ভারের সাথে একটি কানেকশন তৈরি করতে হবে।

```
functionChatRoom({ roomId }) {
useEffect(() => {
// কাজ শুরু: সার্ভারের সাথে কানেক্ট করা
console.log(`✔️ Connecting to "${roomId}" room...`);
constconnection=createConnection(serverUrl, roomId);
connection.connect();

// কাজ বন্ধ: সার্ভার থেকে ডিসকানেক্ট করা
return () => {
console.log(`❌ Disconnected from "${roomId}" room.`);
connection.disconnect();
    };
  }, [roomId]); // এই Effect-টি roomId-এর উপর নির্ভরশীল

return <h1>Welcome to the {roomId} room!</h1>;
}
```

এই `useEffect`-টি React-কে বলছে:

- **কীভাবে কানেক্ট করতে হবে:** `connection.connect()` ব্যবহার করে।
- **কীভাবে ডিসকানেক্ট করতে হবে:** `return` ফাংশনের ভেতরে থাকা `connection.disconnect()` ব্যবহার করে।

React নিজে থেকেই বাকিটা সামলে নেবে।

যখন রুম পরিবর্তন হয়

ধরুন, ইউজার প্রথমে `"general"` রুমে ছিল। এরপর সে ড্রপডাউন থেকে `"travel"` রুম সিলেক্ট করল। তখন কী ঘটবে?

React দেখবে যে `roomId` পরিবর্তন হয়েছে। যেহেতু আমরা `[roomId]` dependency দিয়েছি, React বুঝবে যে তাকে আবার কাজটি করতে হবে। তাই সে:

1. **পুরোনো কাজটি বন্ধ করবে:** আগের `"general"` রুম থেকে ডিসকানেক্ট করার জন্য `return` ফাংশনটি চালাবে।
2. **নতুন কাজটি শুরু করবে:** নতুন `"travel"` রুমের জন্য আবার Effect-টি চালাবে এবং কানেক্ট করবে।

ইউজার যদি আবার রুম পরিবর্তন করে `"music"`-এ যায়, একই ঘটনা ঘটবে। React পুরোনো `"travel"` রুম থেকে ডিসকানেক্ট করে নতুন `"music"` রুমে কানেক্ট করবে।

সবশেষে, যখন ইউজার চ্যাটরুম Component থেকেই বেরিয়ে যাবে (unmount), React শেষবারের মতো ডিসকানেক্ট ফাংশনটি চালিয়ে সবকিছু পরিষ্কার করে দেবে।

এভাবে চিন্তা করলে আপনার কাজ অনেক সহজ হয়ে যাবে। আপনাকে শুধু React-কে **"শুরু"** এবং **"বন্ধ"** করার নিয়ম বলে দিতে হবে।

Dependency Array: Effect-এর চালক

`useEffect`-এর শেষে যে `[]` অ্যারেটি থাকে, তাকে **dependency array** বলে। এটিই ঠিক করে দেয় যে Effect-টি কখন আবার চলবে।

- **`[roomId]`**: এর মানে হলো, যখনই `roomId`-এর মান পরিবর্তন হবে, তখনই Effect-টি আগের কাজ বন্ধ করে নতুন করে আবার চলবে।
- **`[]` (খালি অ্যারে)**: এর মানে হলো, Effect-টির কোনো কিছুর ওপর নির্ভরতা নেই। তাই এটি শুধু Component প্রথমবার স্ক্রিনে আসার সময় (mount) একবার চলবে এবং স্ক্রিন থেকে চলে যাওয়ার সময় (unmount) একবার বন্ধ হবে।
- **কোনো অ্যারে না দেওয়া (না রাখা)**: এটি করলে প্রতিবার Component রেন্ডার হওয়ার সময় Effect চলবে, যা সাধারণত আমরা চাই না এবং এতে পারফরম্যান্স সমস্যা হতে পারে।

Reactive Value কী?

সহজ কথায়, **reactive value** হলো এমন কোনো মান যা রেন্ডারিং প্রক্রিয়ার সময় পরিবর্তন হতে পারে। এর মধ্যে রয়েছে:

- **Props**: যেমন, `roomId`।
- **State**: `useState` বা `useReducer` দিয়ে তৈরি করা যেকোনো ভ্যারিয়েবল।
- **Component-এর ভেতরে তৈরি ভ্যারিয়েবল**: যা props বা state-এর ওপর ভিত্তি করে তৈরি হয়।

নিয়ম হলো, `useEffect`-এর ভেতরে যদি কোনো reactive value ব্যবহার করা হয়, তবে তাকে অবশ্যই dependency array-তে রাখতে হবে।

React-এর সাথে থাকা **linter** টুলটি স্বয়ংক্রিয়ভাবে এই ভুলগুলো ধরে ফেলে এবং আপনাকে সতর্ক করে। এই সতর্কতাকে কখনোই এড়িয়ে যাওয়া উচিত নয়, কারণ এটি প্রায়ই অ্যাপ্লিকেশনের বাগ নির্দেশ করে।

প্রতিটি কাজের জন্য আলাদা Effect

ধরুন, আপনি চ্যাটরুমে ঢোকার সময় একটি অ্যানালিটিক্স ইভেন্ট পাঠাতে চান। আপনি হয়তো কানেকশনের কোডের সাথেই সেটা যোগ করে দেবেন:

```
// 🚫 ভালো অভ্যাস নয়
useEffect(() => {
logVisit(roomId); // অ্যানালিটিক্স
constconnection=createConnection(serverUrl, roomId);
connection.connect();
return () =>connection.disconnect();
}, [roomId, serverUrl]);
```

সমস্যা হলো, এখানে দুটি ভিন্ন কাজ (লগিং এবং কানেকশন) একসাথে মিশে আছে। যদি `serverUrl` পরিবর্তন হয়, তাহলেও অপ্রয়োজনে `logVisit` আবার চলবে।

সঠিক উপায় হলো, প্রতিটি আলাদা কাজের জন্য আলাদা `useEffect` ব্যবহার করা।

```
// ✔️ সঠিক উপায়
useEffect(() => {
logVisit(roomId);
}, [roomId]); // শুধু roomId বদলালে চলবে

useEffect(() => {
constconnection=createConnection(serverUrl, roomId);
connection.connect();
return () =>connection.disconnect();
}, [roomId, serverUrl]); // roomId বা serverUrl বদলালে চলবে
```

এভাবে ভাবুন: প্রতিটি `useEffect` হলো একটি স্বাধীন কাজের অংশ। একটিকে সরিয়ে ফেললেও যেন অন্যটির কোনো সমস্যা না হয়।

সারসংক্ষেপ

- Component-এর lifecycle (mount, update, unmount) নিয়ে না ভেবে Effect-এর **"শুরু"** এবং **"বন্ধ"** হওয়ার চক্র নিয়ে ভাবুন।
- `useEffect` বাইরের কোনো সিস্টেমের সাথে আপনার Component-কে **synchronize** বা সিঙ্ক করে।
- **Dependency array** ঠিক করে দেয় কখন একটি Effect আবার re-synchronize (বন্ধ হয়ে চালু) হবে।
- `useEffect`-এর ভেতরে ব্যবহৃত সব **reactive value** (props, state) অবশ্যই dependency array-তে উল্লেখ করতে হবে।
- প্রতিটি সম্পর্কহীন কাজের জন্য **আলাদা** `useEffect` ব্যবহার করুন।