Manipulating the dom with Refs - React JS Bangla Tutorial

react-bangla.vercel.app/learn-react-escape-hatches/manipulating-dom-with-refs Last Updated: August 23, 2025

Refs দিয়ে ডম নিয়ন্ত্রণ

React নিজের মতো করে ওয়েবপেজ (DOM) আপডেট করে। তাই সাধারণভাবে আপনাকে DOM-ম্যানুপুলেশনের এর কিছু করতে হয় না।

কিন্তু কিছু কিছু সময় আপনি নিজেই DOM- নিয়ন্ত্রণ করতে চাইতে পারেন। যেমন:

- একটা ইনপুট বক্সে ফোকাস দেওয়া
- ইউজারকে স্ক্রল করে কোনো জায়গায় নিয়ে যাওয়া
- কোনো একটি ডিভ-এর উচ্চতা বা অবস্থান জানা

এই কাজগুলো করতে হলে আপনাকে ওই DOM এলিমেন্টটাকে আগে ধরতে হবে। আর এজন্য লাগে **ref**।

এই লিসনে আপনি যা শিখবেন:

- কীভাবে React-এ `ref` দিয়ে DOM এলিমেন্ট ধরা যায়
- `ref` অ্যাট্রিবিউট আর `useRef()` হুক কীভাবে কাজ করে
- কীভাবে অন্য কম্পোনেন্টের DOM এলিমেন্ট ধরা যায়
- কখন React-এর DOM-এ নিজে হাত দেওয়া নিরাপদ

আরো সহজ করে বললে:

> **ref** হলো একটা টুল, যেটা দিয়ে আপনি React-এর ভেতরে থাকা HTML এলিমেন্টকে ধরতে পারেন এবং দরকারমতো কিছু কাজ করতে পারেন।

DOM এলিমেন্টে রেফারেন্স (ref) দেওয়া

React যখন কোনো DOM এলিমেন্ট তৈরি করে, তখন সেটা ধরার জন্য আমাদের একটা **ref** লাগে।

ধাপ ১: প্রথমে `useRef` হুকটি ইমপোর্ট করুন:

```
import { useRef } from"react";
```

ধাপ ২: কম্পোনেন্টের ভেতরে `useRef` দিয়ে একটি রেফারেন্স তৈরি করুন:

```
constmyRef=useRef(null);
```

ধাপ ৩: আপনি যেই DOM এলিমেন্টে অ্যাক্সেস পেতে চান, সেখানে `ref={myRef}` দিয়ে সেটি যুক্ত করুন:

```
<divref={myRef}>
```

`useRef` একটি অবজেক্ট রিটার্ন করে, যার মধ্যে `current` নামে একটি প্রপার্টি থাকে। শুরুতে `myRef.current` হয় `null`। React যখন `<div>` তৈরি করে, তখন সেটার DOM এলিমেন্টটা `myRef.current`-এর মধ্যে রেখে দেয়।

এখন আপনি এই DOM এলিমেন্টকে ইভেন্ট হ্যান্ডলারের ভিতরে ব্যবহার করতে পারবেন।

উদাহরণ:

```
// স্ক্রল করে DOM এলিমেন্টকে দৃশ্যমান করা
myRef.current.scrollIntoView();
```

উদাহরণ: ইনপুটে ফোকাস দেওয়া

এই উদাহরণে বোতামে ক্লিক করলে ইনপুট বক্সে ফোকাস চলে যাবে।

```
import { useRef } from"react";

exportdefaultfunctionForm() {
constinputRef=useRef(null);

functionhandleClick() {
inputRef.current.focus(); // ইনপুটে ফোকাস দেওয়া
  }

return (
    <>
      <inputref={inputRef} />
      <buttononClick={handleClick}>Click to Focus the input</button>
    </>
  );
}
```

কীভাবে কাজ করে:

1. `useRef` দিয়ে `inputRef` বানালাম।
2. `<input ref={inputRef}>` লিখে ইনপুট ফিল্ডের রেফারেন্স তৈরি করলাম।
3. `handleClick` ফাংশনে `inputRef.current.focus()` দিয়ে ইনপুটে ফোকাস দিলাম।
4. `onClick={handleClick}` দিয়ে বাটনের সাথে ইভেন্ট যুক্ত করলাম।

উদাহরণ: কোনো ছবিতে স্ক্রল করা

এই উদাহরণে তিনটি বিড়ালের ছবি আছে। প্রতিটি বোতামে ক্লিক করলে নির্দিষ্ট ছবিতে স্ক্রল হবে।

```
import { useRef } from"react";

exportdefaultfunctionCatFriends() {
constfirstCatRef=useRef(null);
constsecondCatRef=useRef(null);
constthirdCatRef=useRef(null);

functionhandleScrollToFirstCat() {
firstCatRef.current.scrollIntoView({
      behavior:"smooth",
      block:"nearest",
      inline:"center",
    });
  }

functionhandleScrollToSecondCat() {
secondCatRef.current.scrollIntoView({
      behavior:"smooth",
      block:"nearest",
      inline:"center",
    });
  }

functionhandleScrollToThirdCat() {
thirdCatRef.current.scrollIntoView({
      behavior:"smooth",
      block:"nearest",
      inline:"center",
    });
  }

return (
    <>
      <nav>
        <buttononClick={handleScrollToFirstCat}>Tom</button>
        <buttononClick={handleScrollToSecondCat}>Maru</button>
        <buttononClick={handleScrollToThirdCat}>Jellylorum</button>
      </nav>
      <div>
        <ul>
          <li>
            <img
src="https://placekitten.com/g/200/200"
alt="Tom"
ref={firstCatRef}
            />
          </li>
          <li>
            <img
src="https://placekitten.com/g/300/200"
alt="Maru"
ref={secondCatRef}
            />
          </li>
          <li>
            <img
```

```
src="https://placekitten.com/g/250/200"
alt="Jellylorum"
ref={thirdCatRef}
            />
          </li>
        </ul>
      </div>
    </>
  );
}
```

CSS (শুধু ডিজাইনের জন্য):

```
div {
width:100%;
overflow:hidden;
}

nav {
text-align:center;
}

button {
margin:0.25rem;
}

ul,
li {
list-style:none;
white-space:nowrap;
}

li {
display:inline;
padding:0.5rem;
}
```

এইভাবেই `useRef` ব্যবহার করে আপনি DOM-এ ফোকাস, স্ক্রল, অথবা window এর height/width মাপার মতো কাজ সহজেই করতে পারেন।

এখানে React-এ `ref` ব্যবহারের একটি উন্নত টপিক ব্যাখ্যা করা হয়েছে—

কিভাবে একটি ডায়নামিক লিস্টের জন্য অনেকগুলো `ref` ম্যানেজ করবেন, `ref callback` ব্যবহার করে।

উপরের উদাহরণগুলোতে আমরা নির্দিষ্ট সংখ্যক `ref` ব্যবহার করেছি। কিন্তু অনেক সময় এমন হতে পারে যে, আপনার একটি তালিকার প্রতিটি আইটেমের জন্য `ref` দরকার, এবং আপনি জানেন না আইটেম কতগুলো হবে। নিচের কোডটি **চলবে না**:

```
<ul>
  {items.map((item) => {
// চলবে না!
constref=useRef(null);
return <liref={ref} />;
  })}
</ul>
```

কারণ হলো: **React-এর Hooks সবসময় কম্পোনেন্টের টপ-লেভেলে কল করতে হয়।** আপনি `useRef` কে কোনো লুপ, শর্ত (if), অথবা `map()` এর ভিতরে ব্যবহার করতে পারবেন না।

এর একটি সমাধান হতে পারে—তালিকার প্যারেন্ট এলিমেন্টের একটি `ref` রাখা এবং তারপর DOM ম্যানিপুলেশন করে [`querySelectorAll`]() ব্যবহার করে প্রতিটি চাইল্ডকে খুঁজে বের করা। কিন্তু এই পদ্ধতি ঝুঁকিপূর্ণ, কারণ DOM স্ট্রাকচার পরিবর্তন হলে এই কোড ভেঙে যেতে পারে।

আরেকটি ভালো সমাধান হলো: `ref` অ্যাট্রিবিউটে সরাসরি একটি **function (callback)** পাঠানো। এটিকে বলে [`ref callback`]()। React যখন DOM নোড তৈরি করে তখন এই ফাংশনটি কল করে এবং `node` পাঠায়। যখন DOM থেকে সরানো হয়, তখন `null` পাঠায়। আপনি এইভাবে `Map` বা অ্যারে ব্যবহার করে সব `ref` ট্র্যাক রাখতে পারবেন।

নিচের উদাহরণটি দেখায় কিভাবে আপনি এই পদ্ধতি ব্যবহার করে একটি দীর্ঘ তালিকার যেকোনো নোডে স্ক্রল করতে পারেন:

```
import { useRef, useState } from"react";

exportdefaultfunctionCatFriends() {
constitemsRef=useRef(null); // Map রাখার জন্য
const [catList,setCatList] =useState(setupCatList); // ১০টা বিড়ালের লিস্ট

functionscrollToCat(cat) {
constmap=getMap();
constnode=map.get(cat); // সংশ্লিষ্ট DOM node পাওয়া
node.scrollIntoView({
      behavior:"smooth",
      block:"nearest",
      inline:"center",
    });
  }

functiongetMap() {
if (!itemsRef.current) {
// প্রথমবার ব্যবহার হলে Map তৈরি করো
itemsRef.current =newMap();
    }
returnitemsRef.current;
  }

return (
    <>
      <nav>
        <buttononClick={() =>scrollToCat(catList[0])}>Tom</button>
        <buttononClick={() =>scrollToCat(catList[5])}>Maru</button>
        <buttononClick={() =>scrollToCat(catList[9])}>Jellylorum</button>
      </nav>
      <div>
        <ul>
          {catList.map((cat) => (
            <li
key={cat}
ref={(node) => {
constmap=getMap();
if (node) {
map.set(cat, node); // Map-এ DOM node সেট করো
                } else {
map.delete(cat); // node না থাকলে Map থেকে সরাও
                }
              }}
            >
              <imgsrc={cat} />
            </li>
          ))}
        </ul>
      </div>
    </>
  );
}

functionsetupCatList() {
constcatList= [];
```

```
for (let i =0; i <10; i++) {
catList.push("https://loremflickr.com/320/240/cat?lock="+ i);
  }

return catList;
}
```

ব্যাখ্যা:

এখানে `itemsRef` আসলে কোনো DOM node রাখে না। এটি একটি [Map](#) রাখে যেখানে `key` হচ্ছে ক্যাট আইডি আর `value` হচ্ছে সংশ্লিষ্ট DOM নোড।

নিচের কোডটি হলো `ref callback` যেটা প্রতিটি `li`-তে ব্যবহার করা হয়েছে:

```
<li
key={cat}
ref={node => {
constmap=getMap();
if (node) {
map.set(cat, node); // Map-এ যোগ করো
    } else {
map.delete(cat); // Map থেকে মুছে ফেলো
    }
  }}
>

<li
key={cat.id}
ref={node => {
constmap=getMap();
map.set(cat, node);

return () => {
map.delete(cat); // ক্লিনআপ: আনমাউন্ট হলে Map থেকে সরাও
    };
  }}
>
```

এইভাবে আপনি ডাইনামিক সংখ্যক `ref` খুব সহজে এবং নিরাপদভাবে ম্যানেজ করতে পারবেন।

অন্য একটি কম্পোনেন্টের DOM নোডে অ্যাক্সেস পাওয়া

যখন আপনি কোনও বিল্ট-ইন কম্পোনেন্টে (যেমন `<input />`) `ref` ব্যবহার করেন, তখন React সেই `ref` এর `current` প্রপার্টিতে সেই DOM এলিমেন্টটি সেট করে দেয় (যেমন বাস্তবের `<input>` এলিমেন্টটি)।

কিন্তু আপনি যদি নিজের তৈরি করা কম্পোনেন্টে (যেমন `<MyInput />`) `ref` ব্যবহার করতে চান, তাহলে **ডিফল্টভাবে সেটা কাজ করবে না**, এবং `ref.current` হবে `null`। নিচের উদাহরণে দেখানো হয়েছে, যেখানে বাটনে ক্লিক করলেও ইনপুট ফোকাস হয় না:

```
import { useRef } from"react";

functionMyInput(props) {
return <input {...props} />;
}

exportdefaultfunctionMyForm() {
constinputRef=useRef(null);

functionhandleClick() {
inputRef.current.focus();
  }

return (
    <>
      <MyInputref={inputRef} />
      <buttononClick={handleClick}>Focus the input</button>
    </>
  );
}
```

React কনসোলে এ সময় একটি সতর্কবার্তাও দেখায়:

> ⚠️ Warning: Function components cannot be given refs... Did you mean to use React.forwardRef()?

এই সমস্যাটি হয় কারণ React ডিফল্টভাবে অন্য কম্পোনেন্টের DOM নোডে সরাসরি অ্যাক্সেস দেয় না—even যদি সেটা নিজের চাইল্ড কম্পোনেন্টও হয়! এটি ইচ্ছাকৃত, কারণ `ref` মূলত "escape hatch" যা সতর্কভাবে ব্যবহার করা উচিত।

সমাধান: `forwardRef` ব্যবহার করুন

যদি আপনি চান যে আপনার কম্পোনেন্ট তার DOM রেফারেন্স এক্সপোজ করুক, তাহলে আপনাকে `forwardRef` দিয়ে সেই ইচ্ছা প্রকাশ করতে হবে। নিচে `MyInput`-কে `forwardRef` দিয়ে তৈরি করা হয়েছে:

```
constMyInput=forwardRef((props, ref) => {
return <input {...props} ref={ref} />;
});
```

এখন এইভাবে কাজ করে:

1. `<MyInput ref={inputRef} />` — React এখন বুঝবে যে `inputRef.current` এ ইনপুট DOM এলিমেন্ট থাকবে।
2. `MyInput` কম্পোনেন্ট `forwardRef` দিয়ে তৈরি, তাই এটি দ্বিতীয় আর্গুমেন্ট হিসেবে `ref` পায়।
3. সেই `ref` ইনপুট DOM নোডে ফরওয়ার্ড করা হয়।

```
import { forwardRef, useRef } from"react";

constMyInput=forwardRef((props, ref) => {
return <input {...props} ref={ref} />;
});

exportdefaultfunctionForm() {
constinputRef=useRef(null);

functionhandleClick() {
inputRef.current.focus();
  }

return (
    <>
      <MyInputref={inputRef} />
      <buttononClick={handleClick}>Focus the input</button>
    </>
  );
}
```

এই টেকনিকটি সাধারণত ডিজাইন সিস্টেমের জন্য ব্যবহৃত হয়, যেখানে লো-লেভেল কম্পোনেন্ট (যেমন `Button`, `Input`) তাদের DOM রেফারেন্স এক্সপোজ করে। তবে হাই-লেভেল কম্পোনেন্ট (যেমন `Form`, `Page`) সাধারণত তা করে না, যাতে DOM স্ট্রাকচারে ডিপেন্ডেন্সি না তৈরি হয়।

`useImperativeHandle` দিয়ে কাস্টম API এক্সপোজ করা

উপরে `MyInput` কম্পোনেন্ট পুরো ইনপুট DOM এলিমেন্ট এক্সপোজ করছিল। ফলে প্যারেন্ট কম্পোনেন্ট শুধু `focus()` না, বরং অন্য কাজও করতে পারত যেমন স্টাইল বদলানো। যদি আপনি চান কেবল কিছু নির্দিষ্ট ফাংশনই এক্সপোজ করতে, তাহলে `useImperativeHandle` ব্যবহার করতে পারেন:

```
import { forwardRef, useRef, useImperativeHandle } from"react";

constMyInput=forwardRef((props, ref) => {
constrealInputRef=useRef(null);
useImperativeHandle(ref, () => ({
// শুধু focus ফাংশন এক্সপোজ করছি
focus() {
realInputRef.current.focus();
    },
  }));
return <input {...props} ref={realInputRef} />;
});

exportdefaultfunctionForm() {
constinputRef=useRef(null);

functionhandleClick() {
inputRef.current.focus();
  }

return (
    <>
      <MyInputref={inputRef} />
      <buttononClick={handleClick}>Focus the input</button>
    </>
  );
}
```

এখানে `realInputRef` হচ্ছে ইনপুটের আসল DOM নোড। কিন্তু `useImperativeHandle` ব্যবহার করে আমরা এমন একটি অবজেক্ট তৈরি করেছি যেটা শুধু `focus()` ফাংশন এক্সপোজ করে। ফলে `inputRef.current` হচ্ছে একটা কাস্টম অবজেক্ট, ইনপুট DOM নোড নয়।

কখন React `ref` সেট করে

React-এ প্রতিবার আপডেট দুই ধাপে হয়:

1. **Render ধাপে**, React নির্ধারণ করে UI-তে কী দেখাতে হবে।
2. **Commit ধাপে**, React DOM-এ পরিবর্তনগুলো বাস্তবে প্রয়োগ করে।

> 👉 তাই আপনি যখন কম্পোনেন্টে কোড লিখছেন, তখন `ref.current` এ DOM node থাকবে না। প্রথমবার রেন্ডার হলে `ref.current` হবে `null`।

React `ref.current`-কে তখনই সেট করে যখন DOM আপডেট হয় (মানে **commit ধাপে**)। এর আগে সেটাকে `null` করে দেয়, পরে আবার সেট করে DOM node-এ।

✔️ কখন `ref` ব্যবহার করবেন?

- সাধারণত `ref` আপনি **event handler** (যেমন `onClick`) বা `useEffect()` এর ভেতরে ব্যবহার করবেন।
- কারণ এই সময়েই DOM তৈরি বা আপডেট হয়ে গেছে, তাই আপনি নিরাপদে সেটা ব্যবহার করতে পারবেন।

🎯 উদাহরণ: যখন স্ক্রল এক ধাপ পিছিয়ে যায়

ধরুন আপনি একটি টুডু লিস্ট বানালেন, নতুন আইটেম যোগ করে লিস্টের একদম নিচে স্ক্রল করতে চান। কিন্তু নিচের কোডে স্ক্রল হয় এক ধাপ পিছনের আইটেমে:

```
setTodos([...todos, newTodo]);
listRef.current.lastChild.scrollIntoView();
```

সমস্যা কী? `setTodos` সঙ্গে সঙ্গে DOM আপডেট করে না — বরং React এটি "queue" করে। তাই যখন আপনি স্ক্রল করতে যান, তখনও নতুন আইটেম DOM-এ যুক্ত হয়নি।

✔️ সমাধান: `flushSync` ব্যবহার করুন

`flushSync()` React-কে বলে — এখনই DOM আপডেট করো!

```
flushSync(() => {
setTodos([...todos, newTodo]);
});
listRef.current.lastChild.scrollIntoView();
```

এভাবে করলে React সঙ্গে সঙ্গে DOM আপডেট করবে, তারপর আপনি সঠিকভাবে স্ক্রল করতে পারবেন নতুন টুডু-তে।

📌 সারাংশ:

বিষয়	**ব্যাখ্যা**
`ref.current`	রেন্ডার ফেজে পাওয়া যায় না, কমিট ফেজে সেট হয়
`setState`	তাৎক্ষণিক DOM আপডেট করে না
`flushSync`	DOM-এ সাথে সাথে আপডেট করতে সাহায্য করে
`ref` ব্যবহারের সময়	Event handler বা `useEffect()`-এ

🔍 রেফারেন্স (refs) দিয়ে DOM কন্ট্রোল করার সঠিক নিয়ম

React-এ `ref` হল এক ধরনের **বিকল্প দরজা** (escape hatch)। আপনি যখন React-এর সাধারণ নিয়ম দিয়ে কোনো কাজ করতে পারেন না, তখন `ref` ব্যবহার করা যায়।

🎯 কখন `ref` ব্যবহার করবেন?

- ফোকাস (focus) কন্ট্রোল করতে
- স্ক্রল পজিশন ঠিক করতে
- কোনো ব্রাউজার API কল করতে যা React সরাসরি দেয় না

তবে সাবধান! যদি আপনি শুধু ফোকাস বা স্ক্রল করার মতো নিরীহ কাজ করেন, তাহলে সমস্যা হয় না। কিন্তু আপনি যদি **DOM নিজে থেকে পরিবর্তন** করতে যান (React কে না জানিয়ে), তখন React-এর সাথে আপনার কোডের সংঘর্ষ (conflict) হতে পারে।

❌ খারাপ উদাহরণ: DOM নিজে থেকে মুছে ফেলা

নিচের উদাহরণে দুটি বাটন আছে:

- প্রথম বাটন React এর `setState` ব্যবহার করে টেক্সট দেখায় বা লুকায়।
- দ্বিতীয় বাটন সরাসরি DOM থেকে `remove()` দিয়ে মুছে দেয়, React কে না জানিয়ে।

👇 কোডটা দেখুন:

```
import { useState, useRef } from"react";

exportdefaultfunctionCounter() {
const [show,setShow] =useState(true);
constref=useRef(null);

return (
    <div>
      <buttononClick={() =>setShow(!show)}>Toggle with setState</button>
      <buttononClick={() =>ref.current.remove()}>Remove from the DOM</button>
      {show && <pref={ref}>Hello world</p>}
    </div>
  );
}
```

যখন আপনি `Remove from the DOM` ক্লিক করবেন, তখন `Hello world` প্যারাগ্রাফ React এর বাইরে থেকে DOM থেকে মুছে যাবে।

এরপর আবার যদি আপনি `Toggle with setState` দিয়ে সেটা ফেরত আনার চেষ্টা করেন — তখন অ্যাপ ক্র্যাশ করবে!

⚠️ কেন এমন হয়?

কারণ আপনি React এর জানা DOM গঠন ভেঙে দিয়েছেন। React জানেই না, DOM থেকে ওই প্যারাগ্রাফ উধাও হয়ে গেছে। তাই সে ঠিকভাবে আপডেট করতে পারে না।

✔️ কী করা উচিত?

করবেন	করবেন না
`ref` দিয়ে ফোকাস, স্ক্রল, মাপ নেওয়ার মতো কাজ করুন	DOM থেকে নিজে হাতে `remove()` করবেন না
React যেসব DOM এলিমেন্টে হাত দেয় না, সেখানে পরিবর্তন করুন	React-এর ম্যানেজ করা DOM এলিমেন্ট পরিবর্তন করবেন না

✔️ উদাহরণ: কখন আপনি নিরাপদে DOM পরিবর্তন করতে পারেন?

ধরুন আপনার JSX-এ এমন একটা `<div>` আছে যেটা সবসময় খালি থাকবে:

```
<divref={myRef}></div>
```

এখানে React কিছু রেন্ডার করে না, মানে এটা React-এর "রাডারে" নেই। আপনি চাইলে এখানেই নিজে DOM যোগ বা পরিবর্তন করতে পারেন — এটা নিরাপদ।

📌 সংক্ষেপে মনে রাখুন

✔️ রেফ ব্যবহার করে কী করবেন আর কী করবেন না:

- `ref` ব্যবহার করে আপনি DOM এলিমেন্ট ধরে রাখতে পারেন → `myRef.current`
- React এর JSX-এ `<div ref={myRef}>` দিয়ে সেট করতে হয়
- ফোকাস, স্ক্রল, মাপ নেওয়া — এসব `ref` দিয়ে ভালোভাবে করা যায়
- `forwardRef` ব্যবহার করলে কম্পোনেন্ট বাইরের রেফারেন্স নিতে পারে
- React যে DOM ম্যানেজ করে, সেটাকে নিজে থেকে মডিফাই করবেন না
- যদি করতেই হয়, তাহলে এমন DOM বদলান যেখানে React কিছু করে না

এইভাবে আপনি React এর `ref` সিস্টেমকে বুঝে-শুনে ব্যবহার করতে পারবেন — **বিনাশ না করে, সঠিক উপায়ে।**

💙 স্পনসর করুন

আপনি আমার উন্নত কাজকে সমর্থন করেন 🤝