refs

Validates correct usage of refs, not reading/writing during render. See the "pitfalls" section in `useRef()` usage.

Rule Details

Refs hold values that aren't used for rendering. Unlike state, changing a ref doesn't trigger a re-render. Reading or writing `ref.current` during render breaks React's expectations. Refs might not be initialized when you try to read them, and their values can be stale or inconsistent.

How It Detects Refs

The lint only applies these rules to values it knows are refs. A value is inferred as a ref when the compiler sees any of the following patterns:

- Returned from `useRef()` or `React.createRef()`.

  ```
  const scrollRef = useRef(null);
  ```

- An identifier named `ref` or ending in `Ref` that reads from or writes to `.current`.

  ```
  buttonRef.current = node;
  ```

- Passed through a JSX `ref` prop (for example `<div ref={someRef} />`).

  ```
  <input ref={inputRef} />
  ```

Once something is marked as a ref, that inference follows the value through assignments, destructuring, or helper calls. This lets the lint surface violations even when `ref.current` is accessed inside another function that received the ref as an argument.

Common Violations

- Reading `ref.current` during render
- Updating `refs` during render
- Using `refs` for values that should be state

Invalid

Examples of incorrect code for this rule:

```
// ❌ Reading ref during render
function Component() {
  const ref = useRef(0);
  const value = ref.current; // Don't read during render
  return <div>{value}</div>;
}

// ❌ Modifying ref during render
function Component({value}) {
  const ref = useRef(null);
  ref.current = value; // Don't modify during render
  return <div />;
}
```

Valid

Examples of correct code for this rule:

```
// ✅ Read ref in effects/handlers
function Component() {
  const ref = useRef(null);

  useEffect(() => {
    if (ref.current) {
      console.log(ref.current.offsetWidth); // OK in effect
    }
  });

  return <div ref={ref} />;
}

// ✅ Use state for UI values
function Component() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      {count}
    </button>
  );
}

// ✅ Lazy initialization of ref value
function Component() {
  const ref = useRef(null);

  // Initialize only once on first use
  if (ref.current === null) {
    ref.current = expensiveComputation(); // OK - lazy initialization
  }

  const handleClick = () => {
    console.log(ref.current); // Use the initialized value
  };

  return <button onClick={handleClick}>Click</button>;
}
```

Troubleshooting

The lint flagged my plain object with `.current`

The name heuristic intentionally treats `ref.current` and `fooRef.current` as real refs. If you're modeling a custom container object, pick a different name (for example, `box`) or move the mutable value into state. Renaming avoids the lint because the compiler stops inferring it as a ref.

이전
purity

다음
set-state-in-effect