preserve-manual-memoization

Validates that existing manual memoization is preserved by the compiler. React Compiler will only compile components and hooks if its inference matches or exceeds the existing manual memoization.

Rule Details

React Compiler preserves your existing `useMemo`, `useCallback`, and `React.memo` calls. If you've manually memoized something, the compiler assumes you had a good reason and won't remove it. However, incomplete dependencies prevent the compiler from understanding your code's data flow and applying further optimizations.

Invalid

Examples of incorrect code for this rule:

```
// ❌ Missing dependencies in useMemo
function Component({ data, filter }) {
  const filtered = useMemo(
    () => data.filter(filter),
    [data] // Missing 'filter' dependency
  );

  return <List items={filtered} />;
}

// ❌ Missing dependencies in useCallback
function Component({ onUpdate, value }) {
  const handleClick = useCallback(() => {
    onUpdate(value);
  }, [onUpdate]); // Missing 'value'
  return <button onClick={handleClick}>Update</button>;
}
```

Valid

Examples of correct code for this rule:

```
// ✅ Complete dependencies
function Component({ data, filter }) {
  const filtered = useMemo(
    () => data.filter(filter),
    [data, filter] // All dependencies included
  );

  return <List items={filtered} />;
}

// ✅ Or let the compiler handle it
function Component({ data, filter }) {
  // No manual memoization needed
  const filtered = data.filter(filter);
  return <List items={filtered} />;
}
```

Troubleshooting

Should I remove my manual memoization?

You might wonder if React Compiler makes manual memoization unnecessary:

```
// Do I still need this?
function Component({items, sortBy}) {
  const sorted = useMemo(() => {
    return [...items].sort((a, b) => {
      return a[sortBy] - b[sortBy];
    });
  }, [items, sortBy]);
  return <List items={sorted} />;
}
```

You can safely remove it if using React Compiler:

```
// ✅ Better: Let the compiler optimize
function Component({items, sortBy}) {
  const sorted = [...items].sort((a, b) => {
    return a[sortBy] - b[sortBy];
  });

  return <List items={sorted} />;
}
```

이전
incompatible-library

다음
purity

React 학습하기

빠르게 시작하기

설치하기

UI 표현하기

상호작용성 더하기

State 관리하기

탈출구

API 참고서

React APIs

React DOM APIs

커뮤니티

더 보기