パフォーマンス最適化

学習目標

- Reactの仮想DOMと再レンダリングの仕組みを理解する
- useCallbackとuseMemoフックの使い方を学ぶ
- パフォーマンス測定ツールの使用方法を知る

Reactの仮想DOMと再レンダリング

- 仮想DOM: 実際のDOMの軽量なコピー
- 差分検出: 仮想DOMと実際のDOMを比較
- 効率的な更新: 必要な部分のみ更新

useCallbackフック

- 関数のメモ化に使用
- 不要な再生成を防ぎ、パフォーマンスを向上

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useMemoフック

- 計算結果のメモ化に使用
- 高コストな計算の不要な再実行を防ぐ

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

パフォーマンス測定ツール

- React DevTools
  - Profiler機能でコンポーネントの再レンダリングを分析
- Chrome DevTools
  - Performance タブでJavaScriptの実行時間を分析

実践演習

1. React DevToolsをインストールし、Profilerを使用してアプリを分析
2. useCallbackを使用して、タスク操作関数をメモ化
3. useMemoを使用して、フィルタリングされたタスクリストをメモ化
4. 最適化前後でのパフォーマンスの違いを比較