React Hooksと

Reduxと

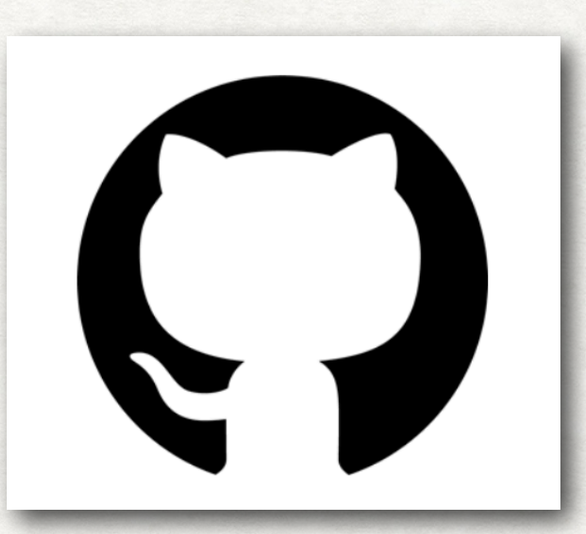
Proxy

自己紹介

- 加藤大志
  - フリーランスプログラマー@お仕事募集中

@dai_shi

@dai_shi



dai-shi

React

- React 0.14から使い始めた
- 理由：function componentsが使えるようになったから
- class componentsは使いたくなかったので、Reduxで全てのstateを保持するスタイル

React Hooks

- React Hooksはfunction componentsで全て完結するようにする仕組み
- 0.14の時代からすぐにできるだろうと思ったが、だいぶ時間が経った
- function componentsが好きな人には嬉しい限り

React Hooks好きすぎて

- 小さいライブラリをいっぱい作った
  - react-hooks-global-state
  - react-hooks-render-props
  - react-hooks-easy-redux ←今日のトピック
  - react-hooks-fetch
  - react-hooks-async
  - react-hooks-worker

ReduxとReact Redux

- Redux自体は独立したライブラリ
  - 原理部分は99行で実装できる https://gist.github.com/gaearon/ffd88b0e4f00b22c3159
- React ReduxがReactに依存したライブラリ
  - パフォーマンスを意識してチューンされている
- パフォーマンスを意識して使うにはobject identityやselectorの理解が必要
  - 初心者には少し難しい

React Hooksによって様々な代替ライブラリが登場

- Reduxを使わずにhooksだけで類似のことを実現するライブラリ
  - “reducer-based global state”
- Reduxを使いつつReact Reduxとは別のbindingsをhooksで実現するライブラリ
  - “hook-based Redux bindings” ←今日のトピック

残り構成

- 一般的にどのように実装されるか
- その問題点は何か
- どんな解決策があるか
- Proxyを使った解決策
- ベンチマーク

コード例: Redux Store作成

```
const initialState = {
  counter: 0,
  text: 'hello',
};

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case 'increment': return { ...state, counter: state.counter + 1 };
    case 'decrement': return { ...state, counter: state.counter - 1 };
    case 'setText': return { ...state, text: action.text };
    default: return state;
  }
};

const store = createStore(reducer);
```

ごくごく簡単なRedux Storeの作成

コード例: 単純なHooks

```
const Counter = () => {
  const state = useReduxState();
  const dispatch = useReduxDispatch();
  const increment = () => dispatch({ type: 'increment' });
  return (
    <div>
      <span>Count: {state.counter}</span>
      <button type="button" onClick={increment}>+1</button>
    </div>
  );
};

const App = () => (
  <ReduxProvider store={store}>
    <Counter />
    <TextBox />
  </ReduxProvider>
);
```

Counterはstate.counterを表示するだけだが、state.textが変更された場合もレンダリングされてしまう

コード例: 素朴な実装

```
const ReduxStoreContext = createContext();

const ReduxProvider = ({ store, children }) => (
  <ReduxStoreContext.Provider value={store}>
    {children}
  </ReduxStoreContext.Provider>
);

const useReduxState = () => {
  const store = useContext(ReduxStoreContext);
  const [state, setState] = useState(store.getState());
  useEffect(() => {
    const callback = () => {
      setState(store.getState());
    };
    const unsubscribe = store.subscribe(callback);
    return unsubscribe;
  }, [store]);
  return state;
};
```

様々な点で限定的であり、
実際にこのまま使うケースは
ほぼない

変更検知問題

- Redux stateは一般的に大きいため、一部の変更によって無関係なコンポーネントまでレンダリングすることは避けたい
  - stateが小さかったり、パフォーマンスが問題にならなければ、素朴な実装でも実は問題ないかも
- 解決法
  - 1. selectorを指定する
  - 2. memoizeする
  - 3. auto-detectする **←今日のトピック**

解決策1: selector

```
const Counter = () => {
  const selector = state => ({ counter: state.counter });
  const state = useReduxStateWithSelector(selector);
  return (
    <div>
      <span>Count: {state.counter}</span>
    </div>
  );
};
```

- React ReduxのconnectのmapStateToPropsを指定するのと同様

解決策2: memoization

```
const Counter = () => {
  const state = useReduxState();
  return useMemo(() => (
    <div>
      <span>Count: {state.counter}</span>
    </div>
  ), [state.counter]);
};
```

- function componentのrender関数は実行されるが、結果は再利用される
- 利用側が注意しなければならない
- 場合によってはパフォーマンスの問題が残る可能性がある

解決策3: auto-detect

```
const Counter = () => {
  const state = useReduxState();
  return (
    <div>
      <span>Count: {state.counter}</span>
    </div>
  );
};
```

- 利用側では何もしなくても、selectorと同様の動作をする
- そんなことができるのか？

Proxy

構文

```
var p = new Proxy(target, handler);
```

引数

`target`
ターゲットのオブジェクト (ネイティブの配列、関数、あるいは他の Proxy も含め、どのような種類のオブジェクトでもかまいません) または、`Proxy` でラップする関数。

`handler`
関数をプロパティとして持つオブジェクトで、その関数で、Proxy に対して操作が行われた場合の挙動を定義します。

- Proxyを使うとオブジェクトへの操作をトラップできる
- stateのうちどの部分が使われたかを知ることができる

Proxyを使って解決策3を実装してみた

依存ライブラリは“proxyequal”というProxyを再帰的に適用してくれるもの

react-hooks-easy-redux

0.9.1 • Public • Published 2 days ago

Readme　　1 Dependencies

react-hooks-easy-redux

Easy React bindings for Redux with Hooks API

https://www.npmjs.com/package/react-hooks-easy-redux

ベンチマークテスト

- react-redux-benchmarksを使って比較実験
- たった一つの特殊な例でのテストなので限定的なことに注意

react-redux 5.0.7

Avg FPS	Scripting	Rendering	Painting
50.73	9262.02	7520.57	576.89

react-hooks-easy-redux 0.8.0

Avg FPS	Scripting	Rendering	Painting
8.37	28016.50	693.61	51.04

- 当初、全然パフォーマンスが出なかった
- 調査によりproxyequalの問題であることが判明
- 作者とともにproxyequalのボトルネックを解消した

react-hooks-easy-redux
v0.9.1

Avg FPS	Scripting	Rendering	Painting
38.96	17988.73	4321.41	303.09

最後に

- 興味ある方は触ってみてください
  - reduxを使って簡単に書きたい人
  - custom hooksを作ってみたい人
  - ベンチマークが好きな人

Thank you!