Object.create(null)

0116
2022

状態は単一の経路を使って参照しよう

プログラミング

React アプリケーションにおいて single source of truth と言った場合, 複数のコンポーネントで同じ値が必要なときは, それぞれのコンポーネントで独立に状態を管理して互いに同期をとるのではなく, ただ一つの場所で状態を管理し, 全てのコンポーネントはそれを参照すべき, という設計のプラクティスとして説明されます.

> There should be a single “source of truth” for any data that changes in a React application. Usually, the state is first added to the component that needs it for rendering. Then, if other components also need it, you can lift it up to their closest common ancestor. Instead of trying to sync the state between different components, you should rely on the top-down data flow.
> https://reactjs.org/docs/lifting-state-up.html#lessons-learned

このプラクティスは状態をどう管理すべきかという視点に立ったもので, 実際これを守らないと各コンポーネントの状態の同期のために余計なコストがかかり, とても簡単にコードを機能不全に陥らせることができます.

このようにして単一の場所で状態を管理するところまでは良いのですが, そういった状態がしばしばコンポーネントツリーの外側に配置されるためか, 一つのコンポーネントが一つの状態を複数の経路で参照してしまうという現象が見られます. このような設計をしてしまった場合, single source of truth が守られなかった場合ほど深刻な問題にはなりにくいものの, 不必要な複雑さを招いてしまう可能性があります.

例

1. Props が単一の経路となっている場合

まずは素朴に, コンポーネントツリーに従った props のバケツリレーのみを使っている場合を考えます.

TypeScript

```
function Counter(props: {
  count: number;
  increment: () => void;
}): React.ReactElement {
  const { count, increment } = props;
  return (
    <p>
      <span>{count}</span>
      <button
        type="button"
        onClick={() => {
          increment();
        }}
      >
        +
      </button>
```

```
      </p>
    );
  }
```

`Counter` コンポーネントは, props という単一の経路から, 状態への参照 (読み書き) である `count` と `increment` を受け取っています.

このコンポーネントは, 親コンポーネントが整合性を持った `count` と `increment` のペアを渡すことを要求します. とはいえコンポーネントのインターフェース (props) からこの要求を読み取ることはさほど難しいことではないでしょう.

この場合については (追加の文脈がなければ) 特に挙げられるような設計上の問題はなさそうです.

2. Hook が単一の経路となっている場合

続いて, 例えば `useContext` や Recoil などを用いて, コンポーネントツリーによらずに (props のバケツリレーから解放されて) 状態を参照できるような場合を考えます.

TypeScript

```
declare function useGlobalCount(): {
  count: number;
  increment: () => void;
};

function Counter(): React.ReactElement {
  const { count, increment } = useGlobalCount();
  return (
    <p>
      <span>{count}</span>
      <button
        type="button"
        onClick={() => {
          increment();
        }}
      >
        +
      </button>
    </p>
  );
}
```

この例でも, `Counter` コンポーネントは, `useGlobalCount` という単一の経路から, 状態への参照である `count` と `increment` を受け取っています. そしてやはり設計上に特に問題は見られません.

3. Props と Hook の両方の経路が使われている場合

同じくコンポーネントツリーによらない状態の参照を行う場合ですが, props と組み合わせるとどうでしょうか?

TypeScript

```
declare function useGlobalCount(): {
  count: number;
  increment: () => void;
};

function Counter(props: { count: number }): React.ReactElement {
  const { count } = props;
  const { increment } = useGlobalCount();
  return (
    <p>
      <span>{count}</span>
      <button
        type="button"
        onClick={() => {
          increment();
        }}
```

```
      >
        +
      </button>
    </p>
  );
}
```

`Counter` コンポーネントは props と hook の二つの経路から, それぞれ状態への参照である `count` と `increment` を取得しています. こういった実装は, 最初は `count` の表示だけを行なっていたコンポーネントに対し, 後から `increment` を行う機能が追加された場合などに, 不注意によって生まれてしまうことがあります. 何度も見たことがあります. 何度も...

このコンポーネントが正しく動作するためには, 親コンポーネントが `useGlobalCount` を使って取得した `count` を, そのままこのコンポーネントに渡すことが要求されます.

ところがこの要求はコンポーネントのインターフェースを見ただけではわからず, 具体的な実装やコメントなどの周辺的な情報まで立ち入って初めて読み取れることです. そのため, コンポーネントの利用者 (親コンポーネントの実装者) はそのことを理解するために余計な手間を強いられることになります.

さらに場合によっては親コンポーネントやそのまた親コンポーネントも `count` を props の一つとして受け取っており, この見えない要求はそういった先祖まで伝播してしまうことがあります. こうなるとコンポーネントの利用者は疑心暗鬼になり, 任意のコンポーネントに対してどういった props を渡すべきかを知るために全ての子孫コンポーネントの実装を読み解くことになるか, あるいは面倒になって開発をやめてしまうでしょう.

このようなコンポーネントの要求に伴う複雑さは, 同等の機能を持つ先の 2 つの例では全く存在しなかったことで, 本来不要なもののはずです. 状態への参照は props のみ, または hooks のみのように, 必ず経路を単一にすることで問題を回避しましょう.

最初に書いたような, コンポーネントに後から実装を追加するような場合には, 要求が満たされた状態から開始するため, 要求が増えたこと自体意識されないこともあるのかもしれません. しかし上記の通り, このようにして増えた要求はコードの理解の妨げになったり, あるいは理解しないまま変更することで将来的な不具合につながってしまう可能性が高いため, 十分に注意を払う必要があります.

まとめ

状態は単一の経路を使って参照しよう.

ところでたまたま React のコードで見かけたので React を使って説明しましたがこれはあくまで例で, こういった話は (single source of truth も含めて) Web フロントエンド, GUI アプリケーションといったものにも限らない, ごく一般的なプログラミングのプラクティスのはずです. 一般化大好き.

by.Susisu (id:susisu) 190日前

関連記事

2022-07-03
共通化すれば良いとは限らない
ここのところ偶然なのか「共通化」という言葉を多く聞いている…

2022-05-06
プログラムの複雑さ・表面積・グラフの構造
特に何かしらの出典はありません. プログラムの複雑さに対する…

2021-12-15
CloudWatch Logs Insights を使って Mackerel 上にアプリケーションの…
この記事は Mackerel Advent Calendar 2021 の 15 日目の記事で…

2021-07-13
exactOptionalPropertyTypes によせて
TypeScript 4.4 に exactOptionalPropertyTypes というオプショ…

2021-02-13