Error Boundary

かつて、コンポーネント内で発生した JavaScript エラーは React の内部状態を破壊し、以降のレンダーで不可解な エラーを 引き起こして いました。このようなエラーはアプリケーションコード中のどこか前の段階で発生したエラーによって引き起こされますが、React はエラーをコンポーネント内で適切に処理する方法を提供していなかったため回復できませんでした。

---

error boundary とは

UI の一部に JavaScript エラーがあってもアプリ全体が壊れてはいけません。React ユーザーがこの問題に対応できるように、React 16 では "error boundary" という新しい概念を導入しました。

error boundary は**自身の子コンポーネントツリーで発生した JavaScript エラーをキャッチ**し、**エラーを記録**し、クラッシュしたコンポーネントツリーの代わりに**フォールバック用の UI を表示**する React コンポーネントです。error boundary は配下のツリー全体のレンダー中、ライフサイクルメソッド内、およびコンストラクタ内で発生したエラーをキャッチします。

> **補足**
>
> error boundary は以下のエラーをキャッチ**しません**：
>
> - イベントハンドラ（詳細）
> - 非同期コード（例： `setTimeout` や `requestAnimationFrame` のコールバック）
> - サーバーサイドレンダリング
> - （子コンポーネントではなく）error boundary 自身がスローしたエラー

クラスコンポーネントに、ライフサイクルメソッドの `static getDerivedStateFromError()` か `componentDidCatch()` のいずれか（または両方）を定義すると、error boundary になります。`static getDerivedStateFromError()` はエラーがスローされた後にフォールバック UI をレンダーするために使用します。`componentDidCatch()` はエラー情報をログに記録するために使用します。

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, info) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

使用する際は通常のコンポーネントとして扱います：

```
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

error boundary はコンポーネントに対して JavaScript の `catch {}` ブロックのように動作します。error boundary になれるのはクラスコンポーネントだけです。実用上、一度だけ error boundary を定義してそれをアプリケーションの至るところで使用することがよくあります。

error boundary は配下のツリー内のコンポーネントで発生したエラーのみをキャッチすることに注意してください。error boundary は自身で起こるエラーをキャッチできません。error boundary がエラーメッセージのレンダーに失敗した場合、そのエラーは最も近い上位の error boundary に伝搬します。この動作もまた、JavaScript の catch {} ブロックの動作と似ています。

ライブデモ

React 16 で error boundary を宣言して利用する例を確認してください。

error boundary を配置すべき場所

error boundary の粒度はあなた次第です。サーバサイドフレームワークがクラッシュを処理する際によく見られるように、最上位のルートコンポーネントをラップしてユーザーに “Something went wrong” メッセージを表示してもいいでしょう。各ウィジェットを個別にラップしてアプリケーションの残りの部分をクラッシュから守るのもいいでしょう。

エラーがキャッチされなかった場合の新しい動作

この変更には重要な意味があります。**React 16 から、どの error boundary でもエラーがキャッチされなかった場合に React コンポーネントツリー全体がアンマウントされるようになりました。**
この決定については議論がありましたが、我々の経験上、壊れた UI をそのまま表示しておくことは、完全に削除してしまうよりももっと悪いことです。例えば、Messenger のような製品において壊れた UI を表示したままにしておくと、誰かが誤って別の人にメッセージを送ってしまう可能性があります。同様に、支払いアプリで間違った金額を表示することは、何も表示しないよりも悪いことです。
この変更のため、React 16 に移行すると、これまで気付かれていなかったアプリケーションの既存の不具合が明らかになることでしょう。error boundary を追加することで、問題が発生したときのユーザー体験を向上できます。
例えば、Facebook Messenger はサイドバー、情報パネル、会話ログ、メッセージ入力欄といったコンテンツを個別の error boundary でラップしています。これらの UI エリアの一部のコンポーネントがクラッシュしても、残りの部分はインタラクティブなままです。
また、本番環境で発生したキャッチされなかった例外について知って修正できるように、JS エラー報告サービスを利用（もしくは自身で構築）することもお勧めします。

コンポーネントのスタックトレース

React 16 は開発時に、レンダー中に起こった全てのエラーをコンソールに出力します（アプリケーションが誤ってエラーを握り潰してしまっても出力します）。そこではエラーメッセージと JavaScript のスタックに加えて、コンポーネントのスタックトレースも提供します。これにより、コンポーネントツリーのどこでエラーが発生したのかが正確にわかります：

```
▶React caught an error thrown by BuggyCounter. You should fix this error in your code.     react-dom.development.js:7708
React will try to recreate this component tree from scratch using the error boundary you provided, ErrorBoundary.

Error: I crashed!

The error is located at:
    in BuggyCounter (created by App)
    in ErrorBoundary (created by App)
    in div (created by App)
    in App
```

コンポーネントスタックトレースにはファイル名と行番号も出力できます。Create React App のプロジェクトではこれがデフォルトで有効になっています：

```
▶React caught an error thrown by BuggyCounter. You should fix this error in your code.     react-dom.development.js:7708
React will try to recreate this component tree from scratch using the error boundary you provided, ErrorBoundary.

Error: I crashed!

The error is located at:
    in BuggyCounter (at App.js:26)
    in ErrorBoundary (at App.js:21)
    in div (at App.js:8)
    in App (at index.js:5)
```

Create React App を使用しない場合は、このプラグインを手動で Babel の設定に追加してください。ただし、この機能は開発専用であり、**本番では必ず無効化しなければならない**ことに注意してください。

> **補足**
> スタックトレースで表示されるコンポーネント名は `Function.name` プロパティに依存します。このプロパティをネイティブで提供しない古いブラウザやデバイス（IE 11 など）をサポートする場合は、アプリケーションバンドルに `Function.name` のポリフィル（`function.name-polyfill` など）を含めることを検討してください。もしくは、全てのコンポーネントに `displayName` プロパティを明示的に設定することもできます。

try/catch について

`try` / `catch` は素晴らしいですが、命令型のコードでのみ動作します：

```
try {
  showButton();
} catch (error) {
  // ...
}
```

一方、React コンポーネントは宣言型であり、**何が**レンダーされるべきなのかを指定します：

```
<Button />
```

error boundary は React の宣言型という性質を保持しつつ、期待通りの動作をします。例えば、`componentDidUpdate` メソッドで発生したエラーがツリー内のどこか深い場所にある `setState` によって引き起こされていた場合でも、最も近い error boundary にそのことが正しく伝播します。

イベントハンドラについて

error boundary はイベントハンドラ内で発生したエラーをキャッチ**しません。**
イベントハンドラ内のエラーから回復するのに error boundary は不要です。レンダーメソッドやライフサイクルメソッドとは異なり、イベントハンドラはレンダー中には実行されません。そのためイベントハンドラ内でエラーが発生しても、React が画面に表示する内容は変わりません。
イベントハンドラ内のエラーをキャッチする必要がある場合は、普通の JavaScript の `try` / `catch` 文を使用してください：

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { error: null };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    try {
      // Do something that could throw
    } catch (error) {
      this.setState({ error });
    }
  }

  render() {
    if (this.state.error) {
      return <h1>Caught an error.</h1>
    }
    return <div onClick={this.handleClick}>Click Me</div>
  }
}
```

上記の例は標準の JavaScript の動作説明であって error boundary を使用していないことに注意してください。

React 15 からの命名の変更

React 15 は error boundary を異なるメソッド名（`unstable_handleError`）で非常に限定的にサポートしていました。このメソッドはもう動作しないため、16 ベータ版リリース以降はコードを `componentDidCatch` に変更する必要があります。

この変更について、自動的にコードを移行できる codemod が提供されています。

このページを編集する