In the past, JavaScript errors inside components used to corrupt React's internal state and cause it to emit cryptic errors on next renders. These errors were always caused by an earlier error in the application code, but React did not provide a way to handle them gracefully in components, and could not recover from them.

Introducing Error Boundaries

A JavaScript error in a part of the UI shouldn't break the whole app. To solve this problem for React users, React 16 introduces a new concept of an "error boundary".

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

Note

Error boundaries do not catch errors for:

- Event handlers (learn more)
- Asynchronous code (e.g. `setTimeout` or `requestAnimationFrame` callbacks)
- Server side rendering
- Errors thrown in the error boundary itself (rather than its children)

A class component becomes an error boundary if it defines either (or both) of the lifecycle methods `static getDerivedStateFromError()` or `componentDidCatch()`. Use `static getDerivedStateFromError()` to render a fallback UI after an error has been thrown. Use `componentDidCatch()` to log error information.

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

```
static getDerivedStateFromError(error) {

  // Update state so the next render will show the fallback UI.

  return { hasError: true };

}
```

```
componentDidCatch(error, errorInfo) {

  // You can also log the error to an error reporting service

  logErrorToMyService(error, errorInfo);

}
```

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

Then you can use it as a regular component:

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

Error boundaries work like a JavaScript `catch {}` block, but for components. Only class components can be error boundaries. In practice, most of the time you'll want to declare an error boundary component once and use it throughout your application.

Note that error boundaries only catch errors in the components below them in the tree. An error boundary can't catch an error within itself. If an error boundary fails trying to render the error message, the error will propagate to the closest

error boundary above it. This, too, is similar to how the `catch {}` block works in JavaScript.