Try catch

Dù code có tuyệt vời đến đâu thì thỉnh thoảng nó vẫn xảy ra lỗi. Lỗi này có thể đến từ những sai lầm của bản, lỗi người dùng nhập vào những giá trị không mong đợi hoặc có hàng ngàn lý do khác.

Bình thường thì khi một đoạn code thực thi bị lỗi, các đoạn code phía dưới của bạn sẽ không được chạy và ứng dụng sẽ bị dừng lại ngay lập tức.

Để xử lý vấn đề này ta thường dùng `try...catch`

Cú pháp `try...catch`

```
try {
  alert('Bắt đầu chạy') // (1) <--
  lalala // Lỗi, biến chưa được khai báo
  alert('Dòng này sẽ không bao giờ chạy') // (2)
} catch (err) {
  alert(`Đã xảy ra lỗi`) // (3) <--
}
```

Theo ví dụ trên, đoạn (1) và (3) sẽ được thực thi

`try...catch` chỉ hoạt động với runtime errors (lỗi thực thi code)

Ví dụ đoạn code dưới đây bị lỗi cú pháp, `try...catch` sẽ không hoạt động

```
try {
  {{{{{{{{{{{{
} catch(e) {
  alert("Dòng này sẽ không được in ra");
}
```

`try...catch` chỉ hoạt động đồng bộ

`try...catch` sẽ không bắt được lỗi trong hàm `setTimeout` phía dưới đây

```
try {
  setTimeout(function () {
    noSuchVariable // Biến chưa được khai báo, lỗi!
  }, 1000)
} catch (e) {
  alert('Không hoạt động')
}
```

Để bắt được lỗi đó, ta phải thực hiện `try...catch` bên trong function của `setTimeout`

```
setTimeout(function () {
  try {
    noSuchVariable // try..catch sẽ bắt được lỗi chưa khai báo biến!
  } catch {
    alert('Lỗi được bắt!')
  }
}, 1000)
```

Error Object

Khi xảy ra lỗi, JS sẽ tạo ra một object chứa thông tin về lỗi đó Object lỗi này có 3 thuộc tính chính

- `name`: Tên lỗi
- `message`: Thông báo lỗi
- `stack`: Thông tin chi tiết về lỗi như thế nào, tại vị trí nào

```
try {
  lalala // Lỗi, biến chưa được khai báo!
} catch (err) {
  alert(err.name) // ReferenceError
  alert(err.message) // lalala is not defined
  alert(err.stack) // ReferenceError: lalala is not defined at (...call stack)

  // Khi dùng như thế này thì alert sẽ tự hiểu
  // và convert object để show ra như một string
  alert(err) // ReferenceError: lalala is not defined
}
```

Chủ động quăng lỗi với `throw`

Để tạo ra một lỗi chủ động,ta có thể dùng `throw`

```
throw 'Lỗi'
throw new Error('lỗi')
throw new SyntaxError('lỗi')
```

`throw` gì thì khi khi dùng `try...catch(e)`, `e` sẽ là định dạng đó. `throw` string thì `e` sẽ là string Vì thế khuyến khích `throw` object dạng Error như `Error`, `SyntaxError`, `ReferenceError`

```
try {
  throw new Error('lỗi')
} catch (e) {
```

```
  console.log(e.message)
}
```

```
let json = '{ "age": 30 }' // incomplete data
try {
  let user = JSON.parse(json)
  if (!user.name) {
    throw new SyntaxError('Incomplete data: no name')
  }
  blabla() // Lỗi không mong đợi
  alert(user.name)
} catch (e) {
  if (e instanceof SyntaxError) {
    alert('JSON Error: ' + e.message)
  } else {
    throw e // rethrow (quăng lại lỗi)
  }
}
```

try…catch…finally

Khi muốn luôn luôn thực thi một đoạn code sau khi `try...catch` thì ta dùng `finally`

```
try {
  // Code có thể lỗi
} catch (e) {
  // Xử lý lỗi
} finally {
  // Code tại đây luôn luôn được thực thi
}
```

Lưu ý:

- Biến khai báo bằng `let`, `const` trong block `try` thì chỉ được dùng trong `try`, không thể dùng trong `catch` hay `finally`. Tương tự với `catch` và `finally`
- Dù cho bạn có return trong `try` thì code trong `finally` vẫn hoạt động
- Ta có thể dùng `try...finally` mà không cần `catch`