


Programação Web Back-end

Adriano Rivolli

rivolli@utfpr.edu.br


Universidade Tecnológica Federal do Paraná (UTFPR)
Câmpus Cornélio Procópio
Departamento de Computação

Conteúdo


- 
- 1 Introdução
 - 2 Tratamento de exceções:
captura e lançamento
 - 3 Classes de erros
 - 4 Tratando erros no back-end

Introdução

Objetivo

- 
- Compreender os tipos de erros
 - Realizar o tratamento de erros adequadamente
 - Gerar erros personalizados
 - Identificar erros que não podem ser tratados
 - Criar projetos web com tratamento de erros

Por que tratar erros?

- 
- Evitar que erros quebrem a aplicação
 - Melhorar a experiência do usuário
 - Organizar o código em relação aos possíveis problemas
 - Customizar as mensagens de erros
 - Controle de logs customizados

Erros por tipo de execução

■ Erros de sintaxe

- ▶ O programa não funciona (erros básicos)

■ Erros de execução

- ▶ Ocorre durante a execução do programa
- ▶ Uma situação imprevista ocorre (é o erro que devemos tratar)

■ Erros de semântica

- ▶ Está relacionado com o resultado incorreto
- ▶ Pode gerar grandes prejuízos

Erros por origem

■ Erros Nativos

- ▶ São os erros inerentes à linguagem, como `ReferenceError`, `TypeError`, `RangeError`, `SyntaxError`

■ Erros personalizados

- ▶ São erros criados pelo desenvolvedor para informar situações específicas

Como identificar erros

- Examinar a *stack* de erros no terminal
- Arquivo de logs
- Documentação
- Experiência prática

Tratamento de exceções: captura e lançamento

Captura de erros

- Ao executar uma ação que pode gerar error
- **try..catch**
 - ▶ Permite especificar uma ou mais respostas para uma exceção lançada
 - ▶ O primeiro bloco (**try**) contém o fluxo normal que pode gerar um erro
 - ▶ O segundo bloco (**catch**) contém as ações alternativas para os casos de erros
 - ▶ O bloco final (**finally**) executa em ambos os casos (sucesso ou falha)

Exemplo de try..catch

JS

```
openMyFile();  
try {  
  writeMyFile(theData); //Isso pode lançar um erro  
} catch (e) {  
  handleError(e); // Se temos um erro temos que lidar com ele  
} finally {  
  closeMyFile(); // Sempre feche o recurso  
}
```

Obtendo informações do erro

- Para obter os detalhes do erro basta utilizar o objeto recebido por parâmetro no `catch`
- O objeto possui as seguintes propriedades:
 - ▶ `console.log(e.name)` *Representa o tipo do erro*
 - ▶ `console.log(e.message)` *Mais detalhes do erro*
 - ▶ `console.log(e.stack)` *Stack do erro*

Tratar ou capturar os erros?

- Existem cenários em que não é possível prever/evitar o erro
 - ▶ Acesso a recursos externos (normalmente assíncronos)
 - ▶ Erros de rede ou de ambiente
- Os demais cenários podem ser previstos e evitados
 - ▶ Divisão de um número por 0
 - ▶ Entradas dos usuários
 - ▶ Acesso a índices inválidos
- Em termos de desempenho é sempre melhor evitar o erro

AVISO!

Nunca use `try..catch`
desnecessariamente

Gerando um erro

- A declaração **throw** lança uma exceção definida pelo usuário
- As instruções após o **throw** não serão executadas
- O controle será passado para o primeiro bloco **catch**
 - ▶ Se não houver um bloco, o programa irá parar de funcionar

Exemplo de geração de erro

■ Erro simples

▶ `throw "Falha ao executar a ação desejada"`


■ Erro completo

```
function UserException(message) {  
  this.name = "UserErro"  
  this.message = message  
}
```

▶ `throw new UserException("Dados inválidos")`

Classes de erros

Principais classes de erros

- 
- Error
 - ReferenceError
 - TypeError
 - SyntaxError
 - RangeError

Error

- Classe genérica que representa todos os erros

- Como lançar um erro personalizado:

- ▶ `throw new Error("Ops!");`

- ▶ *Nota: exemplo didático, não faça isso em produção :)*

ReferenceError

- Ocorre quando se tenta acessar uma variável que não foi declarada ou está fora do escopo
- Como gerar o erro:
 - ▶ `let abc = variavel_nao_declarada`
- Como lançar um erro personalizado:
 - ▶ `throw new ReferenceError("Erro de referência");`

TypeError

- Surge quando se tenta realizar uma operação em um valor que não é do tipo esperado
- Como gerar o erro:
 - ▶ `numero.toUpperCase()`
- Como lançar um erro personalizado:
 - ▶ `throw new TypeError("Erro de tipo");`

RangeError

- Ocorre quando um valor não está dentro do intervalo permitido
- Como gerar o erro:
 - ▶ `new Array(-1)`
- Como lançar um erro personalizado:
 - ▶ `throw new RangeError("Erro de tipo");`

Capturando um erro específico

```
try {  
    Objeto.Metodo();  
} catch (e) {  
    if (e instanceof EvalError) {  
        alert(e.name + ": " + e.message);  
    } else if (e instanceof RangeError) {  
        alert(e.name + ": " + e.message);  
    }  
    // ... etc  
}
```

Tratando erros no back-end

Lista de erros

- Lista de erros que podem ocorrer em uma aplicação web:
 - ▶ <https://nodejs.org/api/errors.html#nodejs-error-codes>
- Alguns casos são exceções que não encerram o servidor

Gerenciador de erros

- Um *middleware* que recebe todos os erros gerados
- Deve ser registrado ao final dos demais
 - ▶ Caso contrário, os próximos registros não serão cobertos por este recurso
 - ▶ Parâmetros: (**erro**, **req**, **res**, **next**)
- É possível especificar alguma rota em particular
- Neste caso, não se usa o **try..catch**