

A large yellow square containing the letters 'JS' in a bold, black, sans-serif font.

JS

exceções

THIAGO DELGADO PINTO
thiago_dp (at) yahoo (dot) com (dot) br

versão: 2020.10.01

o objetivo desse conteúdo é **abordar a sintaxe** do tratamento de exceções em JavaScript e da criação de exceções

os **conceitos** sobre tratamento de exceções já foram tratados em **disciplinas anteriores**

em caso de dúvidas, busque dirimí-las com o professor

revisão dos
conceitos

uma **exceção** representa a ocorrência de um **estado de execução indesejado** para a aplicação ou parte dela

essa ocorrência é geralmente **incomum**
ou pelo menos deveria ser

lançamento e tratamento

quando uma **exceção** é lançada, o **fluxo de execução** da aplicação é **alterado**

o interpretador/compilador procura um **bloco de tratamento da exceção**

ao **não encontrá-lo**, a **execução** da aplicação é passada para o **tratador de exceção padrão da linguagem**

tratador de exceção padrão

geralmente o tratador de exceção padrão da linguagem:

1. **exibe** a exceção na **saída padrão** como sendo um erro; e
2. **termina** a execução.

tratador de exceção do usuário

se o interpretador/compilador **encontrar** um bloco de tratamento de exceção, **desvia** o **fluxo de execução** para o respectivo **bloco**

geralmente as linguagens usam a palavra **catch** para ele

esse bloco **recebe a exceção** que foi lançada

mais de um tratador de exceção do usuário

em **linguagens tipadas**, é possível declarar **mais de um** tratador de exceção

não é o caso de JavaScript, por exemplo

nelas, cada **tratador** captura **exceções** de um **certo tipo** que, por polimorfismo, é compatível com tipos filhos

o interpretador/compilador **analisa** a **compatibilidade** da **exceção** com **cada tipo**, na ordem das declarações (cima para baixo) e **desvia o fluxo** para o **primeiro tratador compatível**, caso haja

fluxo adicional de finalização

o tratador de exceção do usuário também pode acrescentar um **bloco de finalização de execução**

geralmente as linguagens usam a palavra **finally** para ele

esse **bloco** é **sempre executado** pelo interpretador/compilador, tendo a **exceção** sido **lançada** ou não

sintaxe

lançamento de exceção

JavaScript usa a palavra **throw** para lançar uma exceção

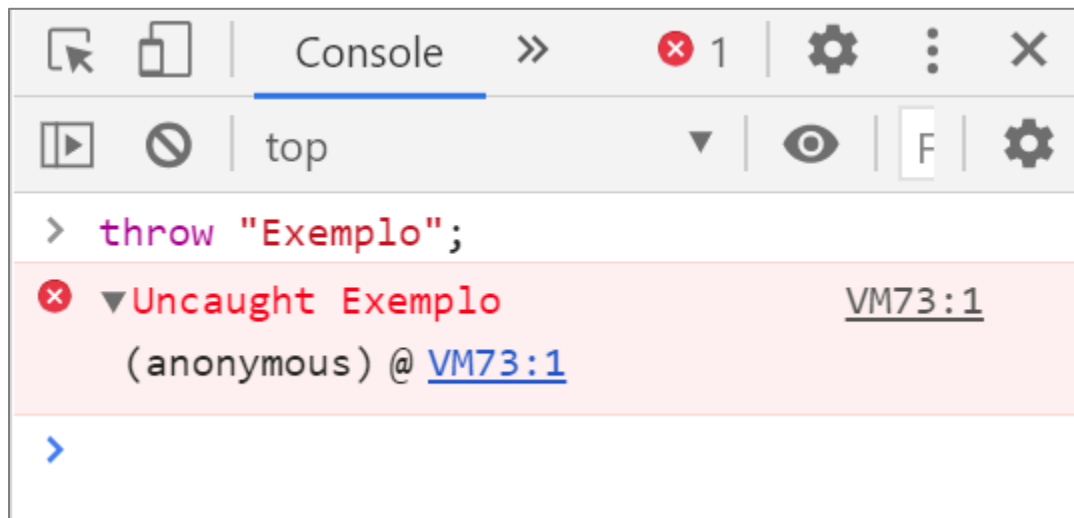
ela pode lançar um **valor** de qualquer tipo

exemplo:

```
function dividir( x, y ) {  
  if ( 0 === y ) {  
    throw 'Dividir por zero não é permitido.';  
  }  
  return x / y;  
}
```

tratador de exceções padrão – no navegador

exemplo de execução no console de um navegador



mudando o tratador de exceção no navegador

```
<html><meta charset="utf8" /><body><script>
  const meuTratador = function(
    mensagemOriginal, url, linha, coluna, excecao
  ) {
    alert( `Ocorreu um problema na aplicação:\n
      ERRO      : ${ 'object' === typeof excecao ? excecao.message : excecao }
      LOCAL     : ${url}:${linha}:${coluna}
      Por favor, contate o suporte e informe o ocorrido.`
    );
  };

  window.onerror = meuTratador;

  throw 'Exemplo';
</script></body></html>
```

tratador de exceções do usuário

```
try {  
  
    // código síncrono que pode gerar uma exceção  
  
} catch ( e ) { // variável que recebe a exceção gerada  
  
    // código de tratamento da exceção  
}
```

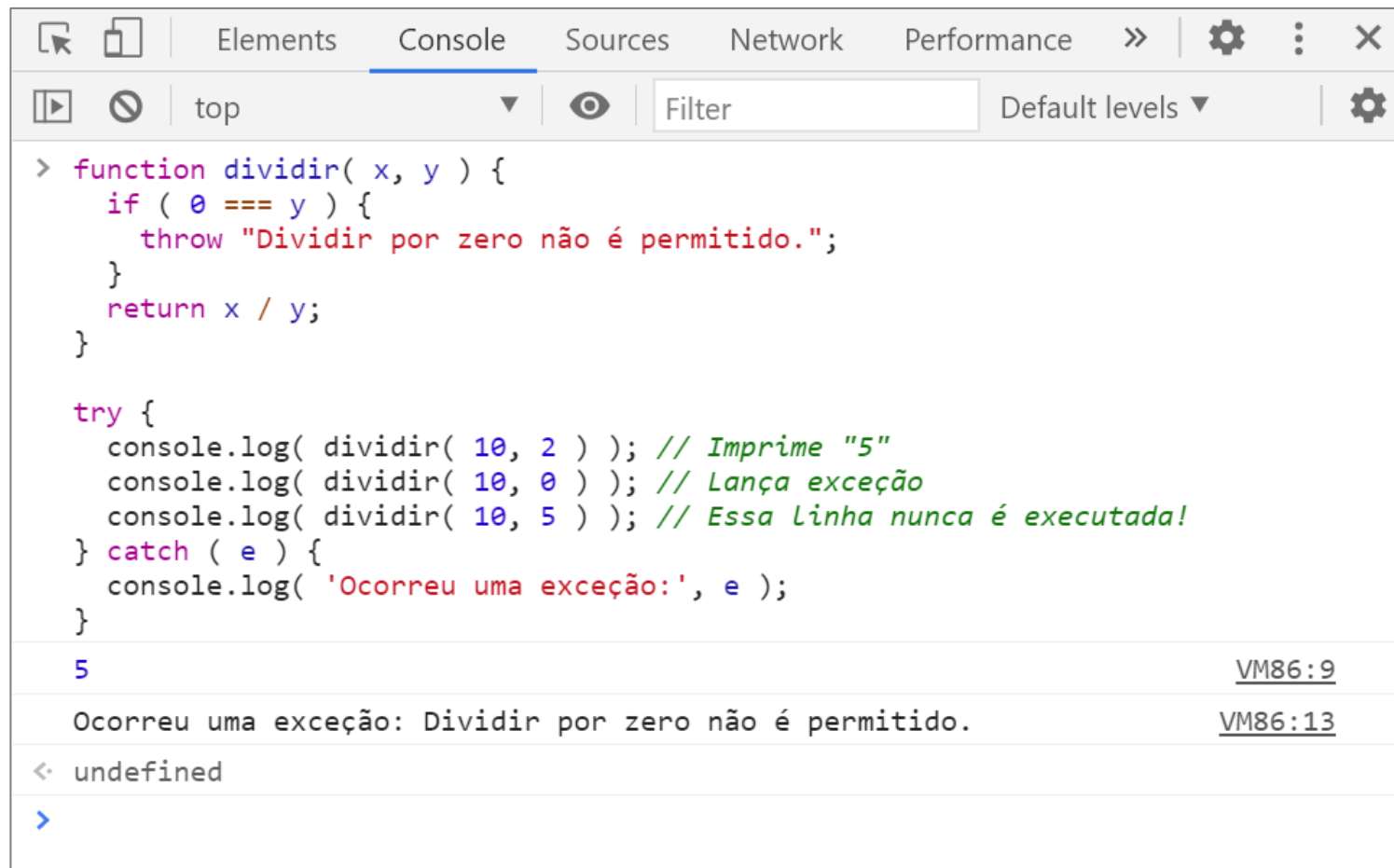
tratador de exceções do usuário – exemplo 1

1 de 2

```
function dividir( x, y ) {  
  if ( 0 === y ) {  
    throw 'Dividir por zero não é permitido.';  
  }  
  return x / y;  
}  
  
try {  
  console.log( dividir( 10, 2 ) ); // Imprime "5"  
  console.log( dividir( 10, 0 ) ); // Lança exceção  
  console.log( dividir( 10, 5 ) ); // Essa linha nunca é executada!  
} catch ( e ) {  
  console.log( 'Ocorreu uma exceção:', e );  
}
```

tratador de exceções do usuário – exemplo 1

2 de 2



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its execution results:

```
> function dividir( x, y ) {  
    if ( 0 === y ) {  
        throw "Dividir por zero não é permitido.";  
    }  
    return x / y;  
}  
  
try {  
    console.log( dividir( 10, 2 ) ); // Imprime "5"  
    console.log( dividir( 10, 0 ) ); // Lança exceção  
    console.log( dividir( 10, 5 ) ); // Essa linha nunca é executada!  
} catch ( e ) {  
    console.log( 'Ocorreu uma exceção:', e );  
}
```

The console output shows the following:

- Line 5: `5` (result of `dividir(10, 2)`) VM86:9
- Line 6: `Ocorreu uma exceção: Dividir por zero não é permitido.` VM86:13
- Line 7: `undefined`
- Line 8: `>`

classe **Error**

JavaScript representa uma exceção com a classe **Error**

principais propriedades:

message: **string** contém a mensagem da exceção

name: **string** contém o nome da classe

stack: **string** contém o rastro da pilha de execução

classe `Error` – observação

executando no navegador e fora do console, **Error** também pode ter:

fileName: string contém o nome do arquivo

lineNumber: number contém a linha do arquivo

columnNumber : number contém a coluna do arquivo

classe Error – exemplo

```
function dividir( x, y ) {  
  if ( 0 === y ) {  
    throw new Error( 'Dividir por zero não é permitido.' );  
  }  
  return x / y;  
}  
  
try {  
  console.log( dividir( 10, 2 ) ); // Imprime "5"  
  console.log( dividir( 10, 0 ) ); // Lança exceção  
  console.log( dividir( 10, 5 ) ); // Essa linha nunca é executada!  
} catch ( e ) {  
  console.log( 'Ocorreu uma exceção:', e.message );  
}
```

principais filhas de Error

Classe	Motivação
RangeError	Um parâmetro numérico está fora de seus limites válidos
ReferenceError	Uma referência se tornou inválida
SyntaxError	Um código que está sendo interpretado tem sintaxe inválida
TypeError	Uma variável, constante ou argumento não é de um tipo válido
URIError	Foram passados parâmetros inválidos para <code>encodeURIComponent()</code> ou <code>decodeURIComponent()</code>

exemplo

```
class Produto {  
    // ...  
    set preco( valor ) {  
        if ( valor < 0.01 ) {  
            throw new RangeError('Preço deve ser de no mínimo um centavo.');        }  
        this._valor = valor;  
    }  
}
```

diferenciando o tipo de exceção (se necessário)

```
try {  
    // ... código que pode lançar exceção ...  
} catch ( e ) {  
    switch ( typeof e ) {  
        case 'string': alert( e ); break;  
        case 'number': alert( 'Erro com código ' + e ); break;  
        case 'object': {  
            if ( e instanceof RangeError ) {  
                alert( 'Faixa incorreta. ' + e.message ); break;  
            } else if ( e instanceof Error ) {  
                alert( e.message ); break;  
            } // senão continua  
        }  
        default: alert( 'Erro desconhecido.' );  
    }  
}
```

bloco de finalização

```
try {  
    // código síncrono que pode gerar uma exceção  
} catch ( e ) { // variável que recebe a exceção gerada  
    // código de tratamento da exceção  
} finally {  
    // código que sempre será executado, mesmo com exceção  
}
```

bloco de finalização – exemplo

```
const PRECO_MINIMO_LOJA = 10.00;
const p = new Produto(); // Inicia com preco 0.01
try {
    p.preco = Number( inputPreco.value ); // Obtém do usuário
} catch ( e ) { // Ocorre se preço menor que 0.01
    alert( e.message );
} finally {
    // Garante o preço mínimo da loja
    if ( p.preco < PRECO_MINIMO_LOJA ) {
        p.preco = PRECO_MINIMO_LOJA;
    }
}
```


criando e usando
novos tipos

criando uma exceção

um **novo tipo** de exceção pode ser criado para **representar** um **contexto específico**

esse contexto tem **relação** com o **nível de detalhes do problema**

nível de abstração importa!

nível de abstração

exceções **mais abstratas** tendem a prover **mensagens** um
uma **visão geral** do problema – *sem detalhes técnicos*

geralmente é direcionada ao usuário final

detalhes técnicos costumam não importar

exceções **menos abstratas** tendem a possuir **detalhes
técnicos**

importam aos desenvolvedores ou usuários técnicos

relançamento de exceção

geralmente ocorre para tornar o tipo **mais abstrato**

ocultar detalhes que não importam ao público alvo

exemplo

```
class RepositorioProdutoEmArquivo extends RepositorioProduto {  
  //...  
  produtoComId( id ) {  
    try {  
      // Pode lançar FileReadError  
      const conteudo = this._arquivo.ler( id + '.json' );  
      // Pode lançar SyntaxError  
      return JSON.parse( conteudo );  
    } catch ( e ) {  
      throw new RepositorioError(  
        `Erro ao ler o produto ${id} de um arquivo.` );  
    }  
  }  
}
```

exemplo de criação de exceção

```
class RepositorioError extends Error {  
    // Necessário para a propriedade name não ficar "Error"  
    name = 'RepositorioError';  
}
```

o uso de **exceções** simplifica o **controle** do **fluxo de execução** de uma aplicação

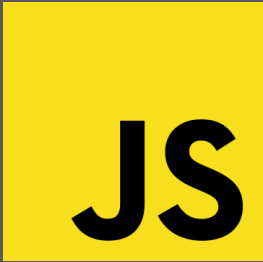
o **tratamento** de uma exceção deve ser realizado pela **camada de abstração** mais apropriada

referências usadas

MDN. *try...catch* – *JavaScript*. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/try...catch>

MDN. *Error* – *JavaScript*. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Error

JavaScript.info. *Error Handling*. Disponível em: <https://javascript.info/error-handling>



fim

Versão 1: 2020.10.01



Licença Creative Commons 4

ESTE MATERIAL PERTENCE AO PROFESSOR THIAGO DELGADO PINTO
E ESTÁ DISPONÍVEL SOB A LICENÇA CREATIVE COMMONS VERSÃO 4.
AO SE BASEAR EM QUALQUER CONTEÚDO DELE, POR FAVOR, CITE-O.