BACHARELADO EM SISTEMAS DE INFORMAÇÃO
CEFET-RJ NOVA FRIBURGO

Programação de Clientes Web

PROF. THIAGO DELGADO PINTO
thiago.pinto@cefet-rj.br

módulos

versão: 2021.08.16

IIFE

immediately-invoked function expression (IIFE)

expressão de função invocada imediatamente após declarada

usada em **JS5 ou anterior** para evitar o escopo global

não é mais usado em JS6+

permite modularizar o código

exemplo 1

```
'use strict';

( function() {
  var nome = 'Ana';
  console.log( nome );
} )();

console.log( nome ); // nome is not defined
```

exemplo 2

```
'use strict';

// Argumentos
( function( $, usuario ) {

  var nome = ( ! usuario || ! usuario.nome )
      ? 'desconhecido' : usuario.nome;
  var mensagem = 'Olá, ' + nome;
  $( '#usuario' ).html( mensagem );

} )( this.jQuery, this.usuarioLogado );
```

modularização com IIFE

permite que **módulos inteiros do sistema evitem o escopo global**

pode-se ter um único global para o sistema

ex., um objeto **app**

os módulos podem ser inseridos nesse objeto único

exemplo 3

```
'use strict';

( function( app, isPushAvailable ) {

  if ( ! app || !! app.config ) {
    return;
  }

  app.config = {
    secondsToPull: 60,
    isPushAvailable: isPushAvailable
  };

} )( this.app, ( 'PushManager' in this ) );
```

ordem das declarações

é importante **declarar as IIFEs na ordem certa**

logo, a **ordem de inclusão dos arquivos** é essencial

exemplo

```
<!-- objeto app precisa ser declarado primeiro, -->
<script type="text/javascript" src="app.js" ></script>
<!-- pois app.config precisa de app -->
<script type="text/javascript" src="app-config.js" ></script>
```

observações

há quem prefira colocar cada módulo como um global

nesse caso, deve-se ter **muito cuidado** com a **colisão de nomes**...

exemplo:

```
'use strict'; // app-config.js
this.config = ( function( isPushAvailable ) {
  return {
    secondsToPull: 30,
    isPushAvailable: isPushAvailable
  };
} )( 'PushManager' in this );
```

exemplo 4 (usado, mas não recomendado!)

```
'use strict'; // forma-pagamento.js
var formaPagamento = ( function() {

  // Importações via variáveis globais
  var jurosCompostos = financeiro.jurosCompostos;
  var numeroParcelasPadrao = config.numeroParcelasPadrao;

  function dividirEmParcelas( valor, opcoes ) {
    /* ... calcula usando jurosCompostos */
  }

  function compensarValorParcelas( parcelas ) {
    /* ... */
  }

  // Objeto com somente as funções que se deseja expor
  return {
    dividirEmParcelas: dividirEmParcelas
  };
} )();
```

exemplo 4.1 (recomendado)

```
'use strict'; // forma-pagamento.js
( function( app, financeiro, config ) {

  // Importações via parâmetros locais
  var jurosCompostos = financeiro.jurosCompostos;
  var numeroParcelasPadrao = config.numeroParcelasPadrao;

  function dividirEmParcelas( valor, opcoes ) {
    /* ... calcula usando jurosCompostos */
  }

  function compensarValorParcelas( parcelas ) {
    /* ... */
  }

  // Insere objeto com somente as funções que se deseja expor
  app.formaPagamento = {
    dividirEmParcelas: dividirEmParcelas
  };
} )( this.app, this.app.financeiro, this.app.config );
```

formatos de módulo

formatos de módulo

Formato	Funciona em	Suporta carregamento assíncrono	Extensão de arquivo
Tag <script>	Navegadores	Sim	.js
CommonJS	Servidores	Não	.js ou .cjs
AMD	Navegadores	Sim	.js
UMD	Navegadores e Servidores	Sim	.js
ECMAScript	Navegadores e Servidores	Sim	.js ou .mjs

CommonJS e *AMD* foram criadas **antes** do ECMAScript 6

para tentar solucionar o problema da modularização via tag script

Tag <script>

exportação/importação realizada através de **variáveis globais**

risco de **colisão** de nomes

não há forma padrão carregar dependências

e dependências das dependências

tudo precisa ser declarado na ordem certa

Tag <script>

carregamento assíncrono muitas vezes não é utilizado

- por conta da ordem de carregamento/execução precisar ser específica

uma solução comum é **unir** os scripts em um **único arquivo**

- pode melhorar significativamente o desempenho
- requer programação cuidadosa

exemplo

- **`app.js`** (todos os módulos da aplicação)
- **`dependencias.js`** (todas as dependências externas)

observações

minificação de arquivos texto

- técnica para melhorar tempo de carregamento
- comentários, tabulações e espaços (não necessários) são removidos
- declarações têm identificadores (nomes) trocados por versões reduzidas
  - substituição ocorre também em todos os locais em que são usados
  - arquivo-mapa (.map) geralmente é gerado para possibilitar depuração
    - para código-fonte
- reduz consideravelmente o tamanho

técnicas similares também aplicáveis a outros tipos de arquivo

- aumento da compressão de imagens
- transmissão de arquivos em formato compactado (gzip ou outro)

CommonJS

originalmente criado para aplicações servidor e desktop

carregamento **síncrono**

- o importador aguarda o módulo importado ser carregado e executado

adotado no NodeJS

- ganhou popularidade
- usado também pelo Node Package Manager (NPM), por exemplo

foi **adaptado** para rodar no **lado cliente**

- Browserify, Webpack, *etc.*

CommonJS – exemplo


```
'use strict'; // forma-pagamento.js

var financeiro = require( './financeiro.js' );
var config = require( './config.js' );

var jurosCompostos = financeiro.jurosCompostos;
var numeroParcelasPadrao = config.numeroParcelasPadrao;

function dividirEmParcelas( valor, opcoes ) {
  /* ... calcula usando jurosCompostos */
}

function compensarValorParcelas( parcelas ) {
  /* ... */
}

// Exportações
module.exports = {
 dividirEmParcelas: dividirEmParcelas
};
```

Asynchronous Module Definition (AMD)

criado para ser mais fácil de usar em **navegadores** que o CommonJS

módulos são carregados de forma **assíncrona**

- navegador não espera enquanto baixa o módulo
- navegador é notificado quando o módulo fica disponível

sintaxe é um pouco mais complicada (que CommonJS)

- nada demais, porém

a implementação mais popular é a RequireJS

- há soluções interessantes que usam AMD quando ES6+ não está disponível
  - SystemJS é uma delas

AMD – exemplo


```
// forma-pagamento.js
define( [ './financeiro.js', './config.js' ],
function( financeiro, config ) {

  var jurosCompostos = financeiro.jurosCompostos;
  var numeroParcelasPadrao = config.numeroParcelasPadrao;

  function dividirEmParcelas( valor, opcoes ) {
    /* ... calcula usando jurosCompostos */
  }

  function compensarValorParcelas( parcelas ) {
    /* ... */
  }

  // Exportações
  return {
    dividirEmParcelas: dividirEmParcelas
  };

} );
```

CommonJS versus AMD

módulos CommonJS geralmente precisam ser...

1. compilados (unificados) antes da distribuição;

   já que a inclusão é síncrona

OU

2. ter um código customizado gerado e avaliado dinamicamente

   o que pode não ser viável na web, seja por segurança ou desempenho

módulos AMD podem ser executados imediatamente

similaridades entre CommonJS e AMD

declaram apenas um módulo por arquivo

um módulo é um pedaço de código que é executado

por *default*, todas as declarações de um módulo são locais

é preciso indicar explicitamente quais declarações se quer exportar

é preciso indicar explicitamente quais declarações se quer importar

definindo o caminho do arquivo ou sua URL

Universal Module Definition (UMD)

use CommonJS, AMD e uso escopo global

somente usa o escopo global se os outros não estiverem disponíveis

ECMAScript

ECMAScript modules (ESM) foram introduzidos na versão 6

suporta dependências com ciclos, como CommonJS

suporta carregamento assíncrono, como AMD

mais simples que os outros formatos

ESM – exemplo

```
// forma-pagamento.js
import { jurosCompostos } from './financeiro.js';
import { numeroParcelasPadrao } from './config.js';

export function dividirEmParcelas( valor, opcoes ) {
  /* ... calcula usando jurosCompostos */
}

function compensarValorParcelas( parcelas ) {
  /* ... */
}
```

ESM

exportação nomeada, importação nomeada

```
// matematica.js

export function multiplicar() {
  if ( arguments.length < 1 ) {
    return 0;
  }
  let acum = 1;
  for ( let arg of arguments ) {
    acum *= arg;
  }
  return acum;
}

export function aoQuadrado( x ) {
  return multiplicar( x, x );
}

export const PI = 3.14159;
```

```
// minha-conta.js

import { multiplicar, PI } from
'./matematica.js';

const raio = Number(
  document.getElementById( 'raio' ).value
);

const area = multiplicar( 2, PI, raio );

document.getElementById( 'area' ).innerHTML =
  `A área da circunferência é ${area}.`;
```

renomeando importação

```
// matematica.js

export function multiplicar() {
  if ( arguments.length < 1 ) {
    return 0;
  }
  let acum = 1;
  for ( let arg of arguments ) {
    acum *= arg;
  }
  return acum;
}

export function aoQuadrado( x ) {
  return multiplicar( x, x );
}

export const PI = 3.14159;
```

```
// minha-conta.js

import { multiplicar as mult, PI } from
'./matematica.js';

const raio = Number(
  document.getElementById( 'raio' ).value
);

const area = mult( 2, PI, raio );

document.getElementById( 'area' ).innerHTML =
  `A área da circunferência é ${area}.`;
```

importação como "namespace"

```
// matematica.js

export function multiplicar() {
  if ( arguments.length < 1 ) {
    return 0;
  }
  let acum = 1;
  for ( let arg of arguments ) {
    acum *= arg;
  }
  return acum;
}

export function aoQuadrado( x ) {
  return multiplicar( x, x );
}

export const PI = 3.14159;
```

```
// minha-conta.js

import * as mat from './matematica.js';

const raio = Number(
  document.getElementById( 'raio' ).value
);

const area = mat.multiplicar( 2, mat.PI, raio
);

document.getElementById( 'area' ).innerHTML =
  `A área da circunferência é ${area}.`;
```

exportação *default* (não nomeada)

um módulo pode ter **no máximo uma** exportação default

não é recomendado misturar com exportações nomeadas

a ideia é que **o módulo seja o que é exportado**

exportação *default* (não nomeada) – exemplo 1

```
// somar.js

export default function() {
  if ( arguments.length < 1 ) {
    return 0;
  }
  let acum = 0;
  for ( let arg of arguments ) {
    acum += arg;
  }
  return acum;
}
```

```
// minha-conta.js

import minhaSoma from './somar.js';

console.log( minhaSoma( 10, 20, 30 ) );
```

exportação *default* (não nomeada) – exemplo 2

```
// app/config.js

export default {
  debugMode: false,
  pullInterval: 60
}
```

```
// app/version.js

export default '1.3.0';
```

```
// main.js

import defaultConfig from './app/config.js';
import versao from './app/version.js';

console.log(
  `MyApp v${versao}\n`,
  'A configuração padrão é', defaultConfig
);
```

exportação *default* (não nomeada) – exemplo 3

```
// usuario.js

export default class Usuario {
  constructor( nome, nivelAcesso ) {
    this.nome = nome;
    this.nivelAcesso = nivelAcesso;
  }
}
```

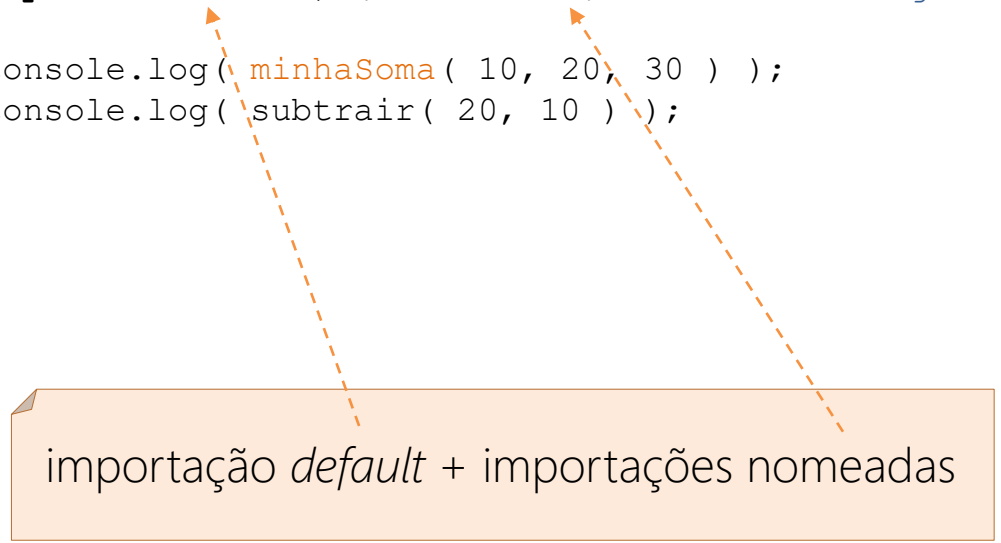
```
// foo.js

import Usuario from './usuario.js';

const u = new Usuario( 'Bob', 'Admin' );

console.log(
  `${u.nome} tem acesso de ${u.nivelAcesso}`
);
```

```
// bar.js

import MeuUsuario from './usuario.js';

const u = new MeuUsuario( 'Bob', 'Admin' );

console.log(
  `${u.nome} tem acesso de ${u.nivelAcesso}`
);
```

exportação *default* (não nomeada) – exemplo 4

```
// somar.js

export default function() {
  if ( arguments.length < 1 ) {
    return 0;
  }
  let acum = 0;
  for ( let arg of arguments ) {
    acum += arg;
  }
  return acum;
}

export function subtrair( a, b ) {
  return a - b;
}
```

```
// minha-conta.js

import minhaSoma, { subtrair } from './somar.js';

console.log( minhaSoma( 10, 20, 30 ) );
console.log( subtrair( 20, 10 ) );
```



sobre modificação de variáveis

```
// contador.js

export let contador = 10;

export function incrementarContador(){
  contador++;
}
```

```
// exemplo.js

import { contador, incrementarContador } from
'contador.js';

console.log( contador ); // 10
incrementarContador();
console.log( contador ); // 11

contador = 20; // Erro - não pode ser alterado
```

referências

RAUSCHMAYER, Axel. **JavaScript for impatient programmers**. 518p., livro digital, sem editora. Disponível em: https://exploringjs.com/impatient-js/

fim

2021.08.16 – Acrescenta exemplo 4 sobre exportação default. Faz pequenos ajustes visuais.
2019.08.12 – versão inicial.