História e conceitos

Lançada em setembro de 1995

Criado por Brendan Eich

ECMAScript

Padroniza as linguagens baseadas em script

TC39

Mantem e atualiza os padrões das bibliotecas ECMAScript

Fluxo de proposta

* Stage 0: strawman
* Stage 1: proposal
* Stage 2: draft
* Stage 3: candidate
* Stage 4: finished

ES2018

* Operadores rest/spread
* Iteração assíncrona
* Promise.prototype.fiinally()

ES.Next : futuras implementações

babeljs.io

Javascript

* É uma linguagem interpretada
* Linguagem de tipagem fraca e dinâmica
* Dinâmica: não precisa definir o tipo de uma variável

Typescript

* Superset do javascript
* Adiciona funcionalidade que o javascript não possui
* <https://www.typescriptlang.org/>
* Possui grande suporte no vscode

Flow

* Versão mais simplificada do typescript

Função de primeira classe e ordem maior

* Significa que a função pode ser atribuída a uma variável, estrutura de dados (array ou object) e passadas para argumentos, retornar outras funões

Closure

* Conhecido também como escopo léxico
* Capacidade de uma função lembrar do ambiente que ela foi criada
* 3 tipos de escopo: global, função e bloco

Currying

* Transforma uma função com n parâmetros, em apenas uma função que recebe um parâmetro, e para cada parâmetro se retorna uma função

Hoisting

* Levantar ou suspender algo
* Declarações tanto de variáveis quanto de funções são elevadas no escopo que ela se encontra
* Dois tipos: de variável e de função
  + Diferença: no de variável só eleva a criação da variável e não sua atribuição, enquanto que no de função ela é elevada ao topo como um todo

Imutabilidade

* Conceito de linguagem funcional, mas existe no javascript
* Uma variável nunca muda, se você precisa alterar ela, você cria uma nova
* Cria uma variável e transfere as informações para ela de forma alterada
* Squid/speed operator: para cada propriedade que recebe ele vai passar para o novo objeto

Tipos e variáveis

* Var:
  + Declaração: var nome;
* Let: se limita ao bloco que foi criado
  + Declaração: let nome;
* Const: cria uma constante, quando é atribuído a um tipo primitivo, não é mais permitido que troque de valor
  + Se criado um objeto é permitido que se altera suas propriedades, o que não se pode mudar é pra onde o ponteiro dele está apontando
  + Declaração: const nome

**TIPOS E VARIÁVEIS**

* string
* number
* boolean
* null
* undefined
* symbol
* Object
* Function (objeto)
* Array (objeto)

**STRING**

* .legth
* .split(‘letra’): corta pelo delimitador
* .replace(‘texto\_trocado’, ‘texto\_troca’)
* .slice(-1): ultima letra de uma string
* .slice(1): da segunda a última letra
* .slice(0,-1): string menos ultima letra
* .substr (0, 2): seleciona as duas primeiras letras

**NUMBER**

* typeof <nome var>: retorna tipo da variável
* .toString()
* .toFixed(numero): fixa quantidade de casas decimais
* parseFloat(): transforma string em float
* parseInt(): transforma string em int

**OBJECT**

* Object.keys()
* Object.values()
* Object.entries()
* Object.assign()
* Object.freeze()
* Object.seal()
* Delete

**SYMBOL**

* Não pode ter as propriedades sobrescritas
* Não enumerável
* getOwnPropertySymbols()
* ownKeys()
* Pode ser usado pra simular a criação de um enum

FUNCTION

Arrow function?

OPERADORES

Operador binário

* Operando1 operador operando2

Operador unário

* ++x
* x++(recebe depois o valor)
* --x
* x--

Aritméticos

* Exponenciação: \*\*
* Adição: + (pode transformar string em number)
* Módulo: %
* Subtração: - (negação)
* Agrupamento: ()

ATRIBUIÇÃO

* x = y
* x = x + y ⬄ x += y
* x = x – y ⬄ x -= y
* x = x \* y ⬄ x \*= y
* x = x / y ⬄ x /= y
* x = x % y ⬄ x %= y

COMPARAÇÃO

* Igualdade: ==
* Não igual: !=
* Estritamente igual: ===
* Estritamente não igual !==
* Ternário
  + Condição ? valor1 : valor

LÓGICOS

* and: &&
* or: ||
* not: !

SPREAD

...<variável>: introduz valores em um lugar

OTHERS

* Unários:
  + Delete
  + Typeof
* Binários
  + In: comparador
  + instanceof: verifica se o objeto é instancia de outro

ESTRUTURAS CONDICIONAIS

* if(true) {code here}
* else if: encadeamento de if’s
* switch(expressão)  
   case valor1: [break;]  
   |  
   default: [break];

ESTRUTURAS DE REPETIÇÃO

* for
  + Mesmo funcionamento do java
* while
  + Mesmo funcionamento do java
* do while
  + Executa ao menos uma vez, depois compara
* for ... in
  + for (let x in arr) -> percorre a array e mostra os índices
* for ... of
  + Mesmo que o for in, mas só percorre as propriedades numeradas
* continue: pula para a próxima iteração, ignora a interação que
* break: quebra o laço

ORIENTAÇÃO A OBJETOS NO JS

HERANÇA

* Baseada em protótipos
* Prototype
* \_\_proto\_\_
* Constructor
* Image: a guide to prototype-based class inheritance in js
* new x

1. Um novo objeto é criado, herdando seu x.prototype
2. A função construtora x é chamada com os argumentos especificados e com o `this` vinculando ao novo objeto criado
3. Caso a função construtora tenha um retorno explícito, será respeitado seu `return`. Senão, será retornado o objeto criado no passo 1.

CLASSE

* ES6
* Simplificação da herança de protótipos
* Palavra-chave class

ENCAPSULAMENTO

* Ocultar detalhes do funcionamento interno

STATIC

* Permite acessar métodos e atributos sem instanciar

DESIGN PATTERNS

Ou padrões de projetos são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software. Não se trata de um framework ou código pronto, mas é sempre uma definição de mais alto nível de como um problema comum pode ser solucionado

Livro: A pattern Language

* 1978
* Christopher Alexander, Sara Ishikawa e Murray Silverstein
* 253 tipos de problemas e desafios de projetos

Formato de um pattern

* Nome
* Exemplo
* Contexto
* Problema
* Solução

Using Pattern languages for object-oriented programs

* 1987
* Kent Beck e Ward Cunningham
* 5 padrões de projetos

Design Patterns: elements of reusable object-oriented softare

* 1994
* Gang of four (GoF)
* Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides

Tipos

* Criação
* Estruturais
* Comportamentais

Padrões de criação

São aqueles que abstraem e/ou adiam o processo de criação de objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados.

* Abstract Factory
* Builder
* Factory Method
* Prototype
* Singleton

Padrões estruturais

Se preocupam com a forma como classes e objetos são compostos para formas estruturas maiores.

* Adapter
* Bridge
* Composite
* Decorator
* Facade
* Bussness Delegate
* Flyweight
* Proxy

Padrões comportamentais

Se concentram nos algoritmos e atribuições de responsabilidades dentre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os objetos.

* Chain of Responsability
* Command
* Interpreter
* Iterator
* Mediator
* Observer
* State
* Strategy
* Template Method
* Visitor

Patterns mais utilizados

* Factory
* Singleton
  + Seu objetivo é criar uma instância de uma função construtora e retorná-la toda vez que for necessário
  + jQuery: <https://jquery.com>
* Decorator
  + Recebe uma outra função como parâmetro e estende o seu comportamento sem modifica-la explicitamente.
  + <https://github.com/tc39/proposals-decorators>
  + <https://www.typescriptlang.org/docs/handbook/decorators.html>
* Observer
  + Muito popular no js. A instância (subscriber) mantém uma coleção de objetos (observers) e notifica todos eles quando ocorrem mudanças no estado
  + Vue: <https://github.com/vuejs/vue/blob/dev/src/core/observer/index.js>
  + RxJs: <https://rxjs-dev.firebaseapp.com/guide/observable>
* Module
  + Possibilita organizarmos melhor nosso código, sem necessidade de expor variáveis globais.

MANIPULAÇÃO E ITERAÇÃO DE ARRAYS

Arrays não são limitados por tipos de parâmetros

CRIAÇÃO DE ARRAY

* Arr = [x, y, z]
* Array.of(x,y,z): cria uma nova instância de array a partir do número de parâmetros informados
* <nome\_var> = Array(n): cria uma array com n posições vazias
* <nome\_var> = Array(3, 2): saída -> 3, 2
* Array.from(): cria uma nova instância de array a partir de um parâmetro array-like ou iterable object

INSERIR E REMOVER ELEMENTOS

* .push(): pode adicionar um ou mais elementos no final do array e retorna o tamanho do novo array
* .pop(): remove o último elemento da array e retorna o elemento removido
* .unshift(): pode adicionar um ou mais elementos no início do array e retorna o tamanho do novo array
* .shift(): remove o primeiro elemento da array e retorna o elemento removido
* .concat(): concatena um ou mais arrays retornando um novo array
* .slice(): retorna um novo array “fatiando” o array de acordo com início e fim, usa a posição dos parâmetros, e se for negativo começa pelo fim
* .splice(): altera um array adicionando novos elementos enquanto remove antigos
  + .splice(n, x, z): n -> posição | x -> quantidade para remover | z -> elementos para adicionar

ITERAR ELEMENTOS

* .forEach(): iteração de cada item dentro de um array
  + Arr.forEach((value, index) => {
  + console.log(´${index}: ${value}` );
  + });
* .map(): retorna um novo array, de mesmo tamanho, iterando cada item de um array
* .flat(): retorna um novo array com todos os elementos de um sub-array concatenados de forma recursiva de acordo com a profundidade especificada (depth), padrão 1.
* .flatMap(): retorna um novo array, assim como a função map e executa um flat de profundidade 1.
* .keys(): retorna um Array Iterator que contém as chaves para cada elemento do array.
* .values(): retorna um Array Iterator que contém os valores para cada elemento do array.
* .entries(): retorna um Array Iterator que contém os pares chave/valor para cada elemento do array.
* .find(value => value > x)
* .findIndex() ~ find()
* .filter(): retorna todos os elementos que satisfazem a condição
* .indexOf(): retorna primeiro índice em que um elemento pode ser encontrado
* .lastIndexOf(): retorna último índice em que um elemento pode ser encontrado
* .includes(): retorna um booleano verificando se determinado elemento existe no array
* .some(): verifica se ao menos um item do array satisfaz determinada condição
* .every(): verifica se todos os itens satisfazem a condição
* .sort(): ordena elementos da array
* .reverse(): inverta a ordem do array
* .join(): junta todos os elementos de um array, separados por um delimitador e retorna uma string
* .reduce(): retorna um novo tipo de dado iterando cada posição de um array

BUSCAR ELEMENTOS

.find():