JAVASCRIPT

DICAS INCRÍVEIS

CAIO RIBEIRO PEREIRA

JavaScript Dicas Incríveis

Uma coleção de dicas e truques úteis para JavaScript

Caio Ribeiro Pereira

Esse livro está à venda em http://leanpub.com/javascript-dicas-incriveis

Essa versão foi publicada em 2019-03-29

Dedico esse livro para os meus pais, obrigado por tudo e principalmente por estar ao meu lado em todos os momentos.

Um agradecimento especial para Aline Brandariz Santos, por ser o sorriso em minha vida, por estar ao meu lado em todos os momentos, por me fazer feliz, por me apoiar e me incentivar na vida.

Agradeço à sra. Charlotte Bento de Carvalho, pelo apoio e incentivo nos meus estudos desde a escola até a minha formatura na faculdade.

Um agradecimento ao meu primo Cláudio Souza. Foi graças a ele que entrei nesse mundo da tecnologia. Ele foi a primeira pessoa a me apresentar o computador, e me aconselhou anos depois a entrar em uma faculdade de TI.

Um agradecimento ao Bruno Alvares da Costa, Leandro Alvares da Costa e Leonardo Pinto, esses caras me apresentaram um mundo novo da área de desenvolvimento de software. Foram eles que me influenciaram a escrever um blog, a palestrar em eventos, a participar de comunidades e fóruns, e principalmente a nunca cair na zona de conforto, a aprender sempre. Foi uma honra trabalhar junto com eles.

Por último, obrigado a você, prezado leitor, por adquirir este livro. Espero que aprenda muito e que você tenha uma ótima leitura!

Conteúdo

Prefácio

JavaScript dominando o mundo

Atualmente JavaScript esta em todo lugar, ele esta fortemente nos browsers, nos servidores (via Node.js e outras JS engines), no mobile, IoT, SmartTVs e muito mais. Se você é desenvolvedor web, já teve que brincar um pouco de JavaScript em algum momento de sua carreira e cada vez mais JavaScript ganha espaço no mundo. Essa linguagem desde 2015 andou recebendo diversos upgrades, como o surgimento do ES6, ES7, ES8, ES9 e ES10, que são implementações de novos recursos para essa linguagem, e isso trouxe novos poderes e principalmente novos meios de resolverem problemas de forma mais simplificada.

Sobre o público desse livro

Esse livro se destina para todos os desenvolvedores que já trabalharam ou trabalham com JavaScript e possuem conhecimentos básicos ou intermediários, e buscam aprender mais sobre essa linguagem, através de um compilado de dicas e truques práticos.

Como ler esse livro

Esse livro diferente de um livro introdutório, apresentará diversos problemas e soluções práticas utilizando JavaScript, não há necessidade de ler os capítulos desse livro em ordem sequencial, afinal esse livro esta em formato “cookbook”, com diversas receitas de soluções práticas, para você aplicar em seus projetos ou usar este livro como referência para consultas.

Capítulo 1: String Tips

Criando slug strings usando regex

Se você precisar criar uma versão "slug" de uma string, exemplo, transformar a frase: `"JavaScript é muito legal"` em `"javascript-é-muito-legal"`, você pode facilmente cria a seguinte função:

```
function slugify(texto) {
  return texto.toLowerCase().replace(/\s/g, '-').trim();
}

slugify('Escrevendo JavaScript Melhor');
// "escrevendo-javascript-melhor"
```

Ou caso queira injetar essa função no objeto String, tornando-a uma função nativa para strings, faça o seguinte:

```
String.prototype.slugify = function() {
  return this.toLowerCase().replace(/\s/g, '-').trim();
}

'Escrevendo JavaScript Melhor'.slugify();
// "escrevendo-javascript-melhor"
```

Interpolando dados em uma string

Desde o surgimento da implementação ES6 (aka ES2015), também já compatível com diversos browsers, já é possível aplicar interpolação de dados em uma string de forma mais elegante, provavelmente você já deve ter visto muito um código parecido com esse:

```
const nome = 'John Connor';
const mensagem = 'Eu venho do futuro!';
let template = '<p>';
template += '<h3>'+ nome + '</h3>';
template += '<span>'+ mensagem + '</span>';
template += '</p>';
```

Porém existe uma forma mais clean, graças a feature *Template Strings* do ES6, você consegue interpolar dados em uma string, sem aplicar concatenação complexa em uma strings, removendo o uso desnecessário do operador **+** em uma string, veja a seguir como fazer isso:

```
const nome = 'John Connor';
const mensagem = 'Eu venho do futuro!';
const template = `
 <p>
    <h3>${nome}</h3>
    <span>${mensagem}</span>
 </p>`;
```

Repetindo strings em uma linha

Quando se trata de repetir strings, é muito comum encontrar códigos semelhantes a esses:

```
let conteudo = '';
const mensagem = 'Olá! ';
const repetir = 3;
for (let i = 0; i < repetir; i++) {
  conteudo += mensagem;
}
console.log(conteudo);
// Olá! Olá! Olá!
```

Ou algo mais enxuto como esse:

```
const repetir = Array(3);
const mensagem = 'Olá! ';
const conteudo = repetir.map(() => mensagem).join('');
console.log(conteudo);
// Olá! Olá! Olá!
```

Mas graças ao ES6, surgiu uma simples função para exercer essa tarefa simples de repetir strings com base no valor de iterações necessárias que for informada em seu primeiro parâmetro, permitindo em uma única linha executar essa tarefa de forma semântica, legível e performática, para isso basta usar a função `String.prototype.repeat()`:

```
console.log('Olá! '.repeat(3));
// Olá! Olá! Olá!
```

Diferenças entre substring() e substr()

Em primeiro lugar, você sabe a diferença entre as funções: `substring()` e `substr()`?

Basicamente ambas funções utilizam a mesma entrada de argumentos, o primeiro argumento é o **indíce de ponto de partida**, porém a diferença esta na **semântica do segundo argumento** de ambas funções, veja.

```
'JavaScript'.substr(4, 6); // "Script"
'JavaScript'.substring(4, 6); // "Sc"
```

- Em `substr(index, quantidade)` o segundo parâmetro retorna a **quantidade de caracteres** a partir do **index** informado.
- Em `substring(inicio, fim)` o segundo parâmetro retorna a **posição final da string**, ou seja, essa função retorna uma nova string com base nas **posições de início e fim**.

Capitalizando strings

Nativamente você pode transformar uma string totalmente em maiúscula (`String.prototype.toUpperCase()`) ou minúscula (`String.prototype.toLowerCase()`), porém não existe uma função para capitalização de uma string, que basicamente tornar a primeira letra em maiúscula e as demais em minúscula de uma palavra, mas é possível criar em poucas linhas de código a lógica dessa função:

```
function capitalize(s) {
  return `${s.charAt(0).toUpperCase()}${s.substr(1).toLowerCase()}`;
}

console.log(capitalize('JAVASCRIPT')); // "Javascript"
```

Caso queira tornar essa função nativa para o objeto String, faça o seguinte:

```
String.prototype.capitalize = function() {
  return `${this.charAt(0).toUpperCase()}${this.substr(1).toLowerCase()}`;
}

console.log('JAVASCRIPT'.capitalize()); // "Javascript"
```

Editando querystrings no browser

Ao invés de perder tempo e aumentar complexidade ao editar strings de uma url manualmente com intuito de incluir novos parâmetros de querystrings, veja esse exemplo ruim:

```
const idioma = 'pt-br';
const secao = 'livros';
let url = 'https://crpwebdev.github.io';
url += '?idioma='+ idioma;
url += '&secao='+ secao;
console.log(url); // "https://crpwebdev.github.io?idioma=pt-br&secao=livros"
```

Você pode utilizar as funções nativas do objeto URL, para aplicar querystrings de uma forma mais segura, esse recurso existe somente no browser via API do HTML5:

```
const idioma = 'pt-br';
const secao = 'livros';
const url = new URL('https://crpwebdev.github.io');
url.searchParams.set('idioma', idioma);
url.searchParams.set('secao', secao);
console.log(url); // "https://crpwebdev.github.io?idioma=pt-br&secao=livros"
```

Três maneiras de converter string para array

Atualmente existem três maneiras de aplicar *"String Splitting"* em uma única linha de código, esse conceito basicamente é sobre transformar uma **string** em um **array de chars**, as vantagens dessa técnica visa habilitar funções nativas de array para o uso em strings:

Usando o clássico método `String.prototype.split()` com argumento **string vazia**:

```
const titulo = 'Book';
// É necessário string vazia em argumento
console.log(titulo.split('')); // ['B', 'o', 'o', 'k']
// split sem argumento vai retornar um array com uma única string
console.log(titulo.split()); // ['Book']
```

Usando `Array.from()` com a string a ser manipulada em argumento:

```
const titulo = 'Book';
console.log(Array.from(titulo)); // ['B', 'o', 'o', 'k']
```

Combinando array com spread operador:

```
const titulo = 'Book';
console.log([...titulo]); // ['B', 'o', 'o', 'k']
```

Aplicando replace all

A função `String.prototype.replace()` é muito útil quando se precisa substituir determinados trechos de uma string, o que torna ela muito mais interessante, é que ela permite a **utilização de expressão regular** na manipulação da string. E graças a isso, é possível fazer com que essa função substitua numa única execução várias ocorrências, ao invés de executar múltiplas vezes a mesma função, para isso basta utilizar o `/g` no final do argumento da expressão que pretende usar:

```
const title = 'JavaJavaScript';
console.log(title.replace(/Java/, '')); // "JavaScript"
console.log(title.replace(/Java/g, '')); // "Script"
```

Pesquisando palavras nos atributos de um objeto

Essa dica é sensacional e bastante útil, com ela você pode percorrer e pesquisar se existe uma ocorrência de uma palavra dentro dos atributos de um objeto, e tudo isso pode ser feito em uma única linha, usando a combinação de `Object.values().toString().includes('string de consulta')`, entenda os detalhes:

```
const pessoa = {
  nome: 'John Connor',
  twitter: '@john'
};

Object.values(pessoa)
  .toString()
  .includes('Connor');
```

Basicamente aconteceu o seguinte comportamento: `Object.values()` converteu para array todos os valores dos atributos de um objeto, em seguida foi aplicado a conversão para String desse Array, através da função `Array.prototype.toString()`, e por fim, desde a implementação do ES6, surgiu a função `String.prototype.includes()` que consulta se existe uma ocorrência de uma palavra dentro da string e retorna `true` se tal ocorrência for existente e `false` caso não exista tal ocorrência.

Extraindo conteúdo de string tags html

Se você estiver num cenário que necessite extrair somente conteúdo de uma string que contenha tags html, você pode limpar essa string, removendo essas tags usando função replace com os seguintes argumentos:

```
const conteudo = '<h1>JavaScript</h1> <h2>é o melhor!</h2>';
const texto = conteudo.replace(/<[a-zA-Z/][^>]*>/g, '');
console.log(texto); // "JavaScript é o melhor!"
```

Capítulo 2: Number Tips

Convertendo número para moeda nativamente

Desde a implementação do ES6, já é possível converter number para "string currency" totalmente de forma nativa, usando apenas a função `Number.prototype.toLocaleString()`, veja:

```
(10.9).toLocaleString(); // "10,90"
(1002.5).toLocaleString('pt-BR'); // "1.002,50"
(5.55).toLocaleString('pt-BR', {
  // Ajustando casas decimais
  minimumFractionDigits: 2,
  maximumFractionDigits: 2
});
```

O melhor disso, é que dessa forma você evita fazer as gambiarras clássicas em usar `Math.abs()` ou `Number.prototype.toFixed()` para realizar esse tipo de conversão, e um outro detalhe legal, essa função esta segura contra bugs de ponto flutuante, que ainda existem no JavaScript, por exemplo:

```
// Resultado com bug de ponto flutuante
0.1 + 0.2 // 0.30000000000000004
// Resultado sem bug de ponto flutuante
(0.1 + 0.2).toLocaleString(); // "0.3"
```

Para mais detalhes sobre formato de locales e outros parâmetros aplicáveis nessa função, você pode consultar nessa documentação da Mozilla: Developers Mozilla: Number.prototype.toLocaleString()[1]

Convertendo string para numbers usando operador +

Essa magia é muito legal e também simples de se fazer. Mesmo que o código não fique legível igual seria usando `Number('100')`, é sempre bom aprender hacks novos, nesse caso, você pode usar o sinal `+`, para converter strings númericas em number, caso a string não tenha números, será retornado um `NaN` *(Not a Number)*. Veja esse exemplo:

[1] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Number/toLocaleString

```
const stringNumerica = '100';
console.log(+stringNumerica); // 100
```

Convertendo date para numbers usando operador +

Assim como o operador + converte uma `String` para `Number`, ele também converte um objeto `Date` para seu respectivo valor de data em milisegundos, muito útil para captura de timestamps:

```
console.log(+new Date()); // 1410480611962
```

Observação: caso você queira usar uma implementação mais legível no lugar do operador +, você pode usar `Date.now()` para obter o valor milisegundos do momento exato que executar essa função ou caso tenha que converter qualquer `Date` existente em milisegundos, utilize sua função `Date.prototype.getTime()`, exemplo: `new Date().getTime()`.

Arredondamento de números usando operador ~~

O uso do operador ~~ para arredondamento de números, em minha opinião, confesso que ficou muito estranho, porém é uma alternativa do uso da função `Math.floor()`, entenda como usá-lo:

```
console.log(Math.floor(10.99)); // 10
console.log(~~10.99); // 10
```

Verificando se número é par ou ímpar

Para descobrir se um número é par ou ímpar, basta utilizar o operador `%` responsável por obter o resto de uma divisão entre dois números, neste caso, se o resto da divisão de um número qualquer por 2 for igual a zero, significa que é par, se for igual a 1 significa que é impar:

```
// Se o resto da divisão for igual zero, então é número par
console.log(12 % 2 === 0); // true
// Se o resto da divisão for igual um, então é número impar
console.log(13 % 2 === 1); // true
```

Calculando idade like a boss

Depois de aprender o uso do operador + para converter `String` ou `Date` para `Number`, e o uso do operador ~~ para arredondamento de números, que tal criar uma função minimalisticamente pequena para calcular a idade com base na data de nascimento informada? É possível fazer um código não muito legível, porém extremamente performático, veja:

```
function calcularIdade(dataNascimento) {
  return ~~((Date.now() - dataNascimento.getTime()) / 31557600000);
}

console.log(calcularIdade(new Date(1990, 5, 4))); // 28
```

Observação: Que número `31557600000` mágico é esse? É basicamente a duração de um ano em milisegundos. Caso queira ver o quão performático é essa função, veja esse link que possui um benchmark dessa função: jsperf.com/birthday-calculation[2]

[2] https://jsperf.com/birthday-calculation