Linguagem de Programação: JavaScript

Departamento de Sistemas de Informação
Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil

Jadson Tavares Santos
João Vitor Sodré de Sousa
Vitor Oliveira Santos

1. Introdução

JavaScript é uma linguagem de programação interpretada, dinâmica e de alto nível. Ela foi originalmente criada para ser usada em páginas web para criar interações dinâmicas com o usuário, como animações, formulários interativos e muito mais. Hoje em dia, ela é amplamente utilizada não apenas para desenvolvimento de aplicações web, mas também em aplicativos de desktop, servidores e muito mais.

JavaScript é conhecido por ser uma linguagem fácil de aprender e altamente versátil, o que a torna uma escolha popular para desenvolvedores de todos os níveis de habilidade. Além disso, existe uma grande comunidade de desenvolvedores e uma ampla gama de bibliotecas e frameworks disponíveis, o que significa que há uma solução para quase todos os problemas que você possa encontrar ao desenvolver sua aplicação.

2. Lexemas do JavaScript

Os lexemas são os elementos básicos da linguagem de programação, também conhecidos como "tokens". Os lexemas incluem:

1. Palavras-chave: como "if", "else", "function", "var", etc.
2. Identificadores: nomes de variáveis, funções, etc.

3. Literais: números, strings, objetos, etc.
4. Operadores: como "+", "-", "===", etc.
5. Delimitadores: como "{", "}", ";", etc.

Estes lexemas são combinados para formar expressões e declarações que são executadas pelo interpretador JavaScript. Ao aprender os lexemas de uma linguagem de programação, você está aprendendo as "peças de construção" que você pode usar para criar suas aplicações.

2.1 Comentários

Existem duas maneiras de adicionar comentários ao seu código:

1. Comentários de linha única: começam com "//" e vão até o final da linha. Por exemplo:

```
// Este é um comentário de linha única
```

2. Comentários de múltiplas linhas: começam com "/*" e terminam com "*/". Por exemplo:

```
/*
 Este é um comentário
de múltiplas linhas.
 */
```

Comentários são úteis para adicionar explicações claras e descrições do que o código está fazendo. Eles também podem ser úteis para desabilitar rapidamente partes do código sem excluí-las completamente.

2.2 Palavras reservadas

As palavras-chave são termos especiais na linguagem que têm uma função específica e não podem ser usados como identificadores (nomes de variáveis, funções, etc.). Aqui estão algumas das palavras-chave do JavaScript:

- ***break***: É usado para encerrar um loop ou declaração switch e transferir o controle para a instrução imediatamente após a instrução encerrada.

- ***case***: É usado em uma instrução switch para especificar a ação a ser tomada quando um valor particular corresponde ao valor sendo avaliado na instrução switch.
- ***catch***: É usado para lidar com exceções (erros) que são lançadas em um bloco try.
- ***class***: É usado para definir uma classe em JavaScript, que é um modelo para criar objetos com propriedades e métodos específicos.
- ***const***: É usado para declarar uma variável que não pode ser reatribuída depois de ter sido atribuída um valor.
- ***continue***: É usado para pular a iteração atual de um loop e seguir para a próxima iteração.
- ***debugger***: É usado para pausar a execução de um programa JavaScript em um ponto específico no código, para facilitar a depuração.
- ***default***: É usado em uma instrução switch para especificar a ação a ser tomada quando nenhum dos valores do case corresponde ao valor sendo avaliado na instrução switch.
- ***delete***: É usado para remover uma propriedade de um objeto.
- ***do***: É usado para criar um loop que executa um bloco de código até que uma condição especificada seja atendida.
- ***else***: É usado para especificar uma ação alternativa a ser tomada se a condição em uma instrução if for falsa.
- ***export***: É usado para exportar um módulo ou um objeto nomeado de um módulo.
- ***extends***: É usado para criar uma subclasse de uma classe existente em JavaScript.
- ***finally***: É usado em um bloco try/catch para especificar um bloco de código que será executado independentemente de uma exceção ser lançada ou não.
- ***for***: É usado para criar um loop que executa um bloco de código por um número específico de vezes.
- ***function***: É usado para definir uma função em JavaScript, que é um bloco de código reutilizável que executa uma tarefa específica.

- ***if***: É usado para especificar uma condição e executar um bloco de código se a condição for verdadeira.
- ***import***: É usado para importar um módulo ou um objeto nomeado de um módulo.
- ***in***: É usado para verificar se uma propriedade existe em um objeto.
- ***instanceof***: É usado para verificar se um objeto é uma instância de uma classe específica.
- ***new***: É usado para criar uma instância de uma classe em JavaScript.
- ***return***: É usado para retornar um valor de uma função.
- ***super***: É usado para chamar um método na classe pai de um objeto.
- ***switch***: É usado para avaliar um valor e executar um bloco de código dependendo da correspondência do valor com os casos na instrução switch.
- ***this***: É usado para se referir ao objeto atual em JavaScript.
- ***throw***: É usado para lançar uma exceção em JavaScript.
- ***try***: É usado para criar um bloco de código que pode gerar exceções (erros) e especificar um bloco de código que lidará com essas exceções.
- ***typeof***: É usado para obter o tipo de uma variável ou valor em JavaScript.
- ***var***: Era usado para declarar uma variável em JavaScript, mas agora é recomendado usar let e const.
- ***void***: É usado para especificar que uma expressão não deve retornar um valor.
- ***while***: É usado para criar um loop que executa um bloco de código enquanto uma condição específica é verdadeira.
- ***with***: Era usado para estabelecer um contexto de objeto para uma sequência de instruções, mas agora é desencorajado devido a problemas de segurança.
- ***yield***: É usado em funções geradoras para retornar um valor e pausar a execução da função até que a próxima chamada seja feita.
- ***let***: É usado para declarar uma variável com escopo de bloco. É recomendado usar let em vez de var.
- ***private***: É usado para definir membros privados em classes.
- ***protected***: É usado para definir membros protegidos em classes.
- ***public***: É usado para definir membros públicos em classes. É o modificador

padrão se nenhum modificador de acesso for especificado.

- ***static***: É usado para declarar membros estáticos em classes ou métodos estáticos em funções.
- ***await***: É usado em funções assíncronas para pausar a execução até que uma Promise seja resolvida.
- ***abstract***: É usado para definir classes abstratas ou métodos abstratos em classes abstratas. Uma classe abstrata não pode ser instanciada e deve ser estendida por uma classe filha.
- ***boolean***: É usado para declarar uma variável que pode ter um valor booleano, true ou false.
- ***byte, char, double, float, int, long, short***: São tipos de dados numéricos em JavaScript.
- ***final***: É usado para declarar uma variável que não pode ser reatribuída após a sua inicialização.
- ***true***, ***false***: São valores booleanos que representam verdadeiro e falso, respectivamente.
- ***null***: É usado para representar a ausência de um valor.

2.3 Literais reservados

• ***null***: define que uma variável não está apontando para nada.

• ***false***: valor lógico que indica falso.

• ***true***: valor lógico que indica verdadeiro.

2.4 Operadores e delimitadores

Existem vários tipos de operadores e delimitadores que são usados para realizar operações matemáticas, comparações, atribuições e outras ações. Aqui estão alguns dos principais operadores e delimitadores do JavaScript:

1. Operadores matemáticos: +, -, *, /, %, ++, --.
2. Operadores de atribuição: =, +=, -=, *=, /=, %=.
3. Operadores de comparação: ==, ===, !=, !==, >, <, >=, <=.
4. Operadores lógicos: &&, ||, !.
5. Operadores ternários: ? e :.
6. Delimitadores de bloco: { e }.
7. Delimitadores de parênteses: (e).
8. Delimitadores de colchetes: [e].
9. Delimitadores de ponto e vírgula: ;.
10. Delimitadores de string: aspas duplas “ ” e aspas simples ‘ ’.
11. Delimitadores de objetos: chaves { e }.

Cada um desses operadores e delimitadores desempenha uma função específica na linguagem JavaScript e é usado para construir expressões e declarações na linguagem. A precedência de operação é a ordem em que as operações matemáticas são avaliadas em uma expressão. Em JavaScript, existe uma ordem padrão de precedência para os operadores, que é a seguinte:

1. Expressões entre parênteses (e) têm a maior precedência e são avaliadas primeiro. 2. Operações matemáticas unárias, como ++ e --, têm a segunda maior precedência. 3. Operações matemáticas de multiplicação, divisão e resto *, /, % têm a terceira maior precedência.
4. Operações matemáticas de adição e subtração +, - têm a quarta maior precedência.
5. Operações de atribuição =, +=, -=, etc. têm a quinta maior precedência.
6. Operadores ternários ? e : têm a sexta maior precedência.
7. Operadores lógicos &&, || têm a menor precedência.

Se houver mais de uma operação com a mesma precedência, elas são avaliadas da esquerda para a direita. É sempre uma boa prática usar parênteses para garantir que as operações sejam avaliadas na ordem correta.

2.5 Literais String

Os literais de string são valores que representam cadeias de caracteres. Eles são

delimitados por aspas duplas “ ” ou aspas simples ‘ ’.

Por exemplo:

var string1 = "Olá, mundo!";

var string2 = 'Bem-vindo ao JavaScript';

Você pode usar aspas duplas dentro de uma string delimitada por aspas simples e vice-versa, basta colocar uma barra invertida \ antes da aspa que você deseja usar dentro da string:

var string3 = "Ele disse: \"Olá, mundo!\""; var

string4 = 'Ele disse: \'Olá, mundo!\';

Além disso, você pode usar uma sintaxe especial de string de múltiplas linhas, incluindo aspas triplas “““ ”””” ou apóstrofos triplos ‘‘‘’’’. Por exemplo:

var string5 = `Este é um exemplo de string de múltiplas linhas`;

Você também pode usar expressões dentro de uma string, incluindo variáveis, operações matemáticas e métodos, usando chaves ${}:

var nome = "João";

var idade = 30;

var string6 = `Meu nome é ${nome} e tenho ${idade} anos`;

2.6 Literais Numéricos

Os literais numéricos representam valores numéricos. Eles podem ser inteiros ou decimais, sem restrições de tamanho.

2.6.1 Literais Integer

Um literal inteiro é um número inteiro, sem casas decimais. Eles podem ser positivos, negativos ou zero. Alguns exemplos de literais inteiros em JavaScript são: *var positiveInteger = 42;*

var negativeInteger = -42;

var zero = 0;

2.6.1.1 Literal Decimal

Um literal decimal é um número com casas decimais. Eles podem ser positivos, negativos ou zero. Alguns exemplos de literais decimais em JavaScript são: *var positiveDecimal = 3.14;*

var negativeDecimal = -3.14;

var zero = 0.0;

Também é possível representar números decimais usando notação científica, usando um e seguido de um expoente:

var scientific = 1.23e-5;

2.6.1.2 Literal Octal

Um literal octal é um número inteiro representado na base octal (base 8). Para representar um número na base octal, você deve adicionar um prefixo 0o antes do número. Alguns exemplos de literais octais em JavaScript são:

var octal1 = 0o10;

var octal2 = 0o20;

var octal3 = 0o30;

2.6.1.3 Literal Hexadecimal

Um literal hexadecimal é um número inteiro representado na base hexadecimal (base 16). Para representar um número na base hexadecimal, você deve adicionar um prefixo 0x antes do número. Alguns exemplos de literais hexadecimais são:

var hex1 = 0xA; var hex2 = 0xB; var hex3 = 0xC;

2.6.4 Literais Booleanas

Existem dois valores lógicos: *true* (verdadeiro) e *false* (falso). Eles são chamados de literais booleanos. Esses valores são usados para representar verdadeiro ou falso em condições lógicas e estruturas de controle de fluxo. Alguns exemplos de literais booleanos são: *var boolean1 = true;*

var boolean2 = false;

2.6.5 Literais Char:

JavaScript não tem suporte para literais de caracteres (char) separados. No entanto, você pode representar caracteres como strings de tamanho 1. Por exemplo: *var char1 = "a";*

var char2 = "b";

Para obter o valor numérico correspondente ao caractere, você pode usar a função charCodeAt():

var charCode = "a".charCodeAt(0);

Observe que as strings são representadas como arrays de caracteres e você pode acessar cada caractere de uma string usando índices.

Referências:

MDN web docs. JavaScript. Referência à gramática léxica. Mozilla Developer Network. Data de acesso: 13 de fevereiro de 2023.

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Lexical_grammar.