DESENVOLVIMENTO DE SOFTWARE PARA WEB

Fundamentos de JavaScript

Prof. Bruno Góis Mateus

(brunomateus@ufc.br)

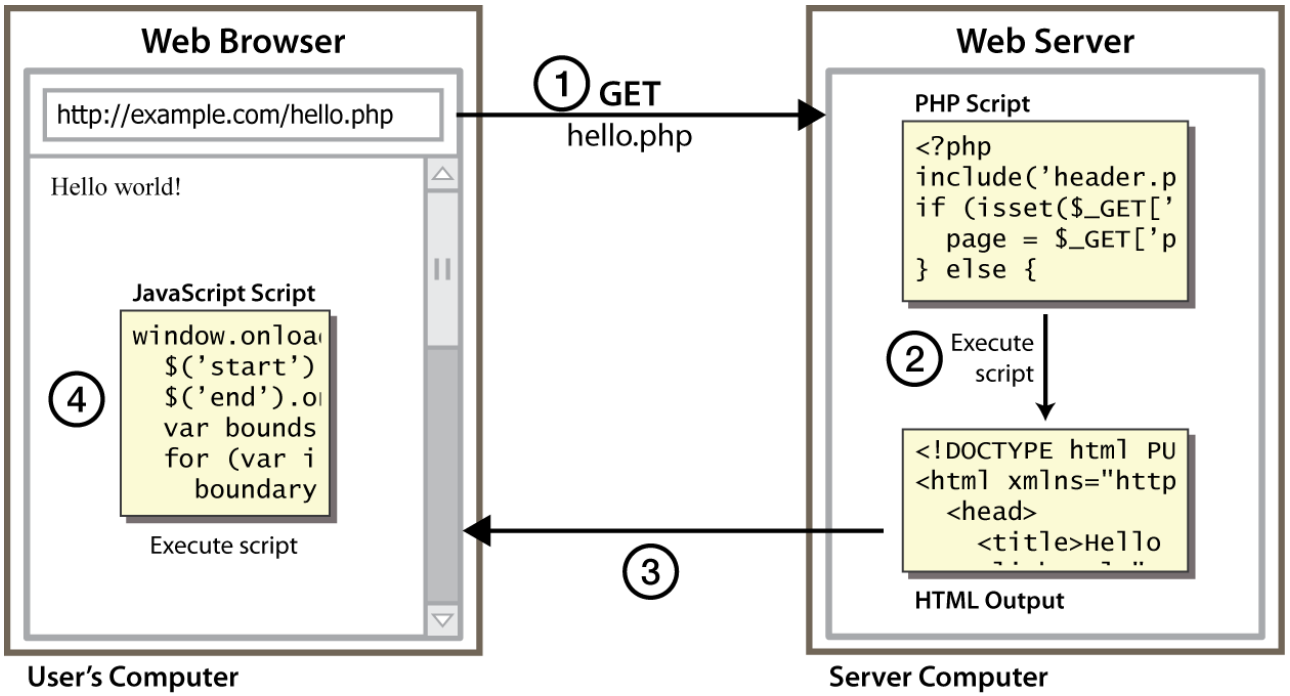
Índice

- Introdução
- JavaScript
- Tipos de dados
- Comentário
- Instruções
- Vetores
- Exceções
- Objetos
- Funções

INTRODUÇÃO

Introdução

- Scripts no lado do cliente
  - São executados no navegador após a página ser enviada de volta pelo servidor
  - Frequentemente esse código manipula a página ou as ações do usuário



Por que usar programação no lado do cliente?

- Benefícios dos scripts no lado do cliente:
  - Usabilidade:
    - Pode modificar uma página sem aguardar por dados do servidor (mais rápido)
  - Eficiência:
    - Pode realizar mudanças pequenas e rápidas sem esperar pelo servidor
  - Orientada a eventos:
    - Pode responder a ações do usuário, como cliques e teclas pressionadas
- Benefícios de linguagens do lado do servidor:
  - Segurança:
    - Tem acesso a dados privados. O cliente não pode ver o código fonte
  - Compatibilidade:
    - Não depende das implementações dos navegadores
  - Poder:
    - Pode escrever arquivos, abrir conexões com servidores, conectar com banco de dados

Histórico

- Criada por Brendan Eich em 1995 para a Netscape
  - Originalmente chamado LiveScript
  - Alteração do nome por decisão de *marketing* apoiada na popularidade da linguagem Java tem sido uma fonte de mal entendidos
- Adaptado pela Microsoft como JScript para IE 3 em 1996
- Padronizado como ECMAScript em 1997
- Variação incluída no Flash como ActionScript
- Atualizada para ECMAScript 3ª Edição em 1998

O que é JavaScript

- Uma linguagem de programação (linguagem e script)) leve
- Usada para fazer páginas web interativas
  - Inserir texto dinamicamente no código HTML
  - Reagir a eventos (ex: carregamento da página, cliques do usuário)
  - Pegar informações sobre o computador do usuário(ex: navegador)
  - Realizar cálculos no computador do usuário (Ex: validação de formulário)
- Um **padrão web**
  - Não é suportado por todos os navegadores
- Não tem nenhuma relação com Java, a não ser pelo nome e algumas sintaxes similares

JavaScript vs Java

- **Interpretada**, não compilada
- Regras e sintaxe menos rígidas
  - Poucos tipos de dados
  - Variáveis não precisam ser declaradas
  - Os erros são frequentemente silenciados
- A construção mais importante são as funções, ao invés de classes
- Estão contidas na página web, integrada com seu HTML e CSS

JavaScript vs PHP

- Similaridades
  - Ambas são interpretadas
  - Ambas tem regras e sintaxe "relaxadas"
  - Ambas são case-sensitive
  - Ambas possuem mecanismos de avaliar expressões regulares embutidos
- Diferenças:
  - JavaScript é mais orientada a objetos
  - JavaScript foca mais na interface e interações com o documento, PHP em saídas HTML, arquivos e formulários
  - JavaScript roda no navegador* do cliente, PHP é executada no servidor web

TIPOS DE DADOS

Tipos de dados

```
var name = expression;
var age = 32;
var weight = 127.4;
var clientName = "Connie Client";
```

- Variáveis são declaras através da palavra chave var
- O tipo da variável não é especificado
- Os possíveis tipos de dados são:
  - Numbers, Boolean, String, Null, Undefined
  - Object
    - Array,
    - Function
  - As função typeof retorna o tipo de dados de objeto

String

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); //"Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant'; // can use "" or ''
```

- Principais métodos
  - charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring toLowerCase, toUpperCase
- lenght é uma propriedade, não um método com em Java
- Concatenação: 1 + 1 é 2, mas "1" + 1 é "11"

String – Principais métodos

Método / Propriedade	Descrição
lenght	Propriedade que contém o tamanho da string
concat( )	Concatena um ou mais strigs
indexOf( )	Retorna a primeira ocorrência de um caractere na string
lastIndexOf( )	Retorna a última ocorrência de um caractere na string
match( )	Verifica a ocorrência de uma expressão regular na string
replace( )	Substitui alguns caracteres na string
slice( )	Extrai em uma nova string, parte da string original
split( )	Quebra a string em um array de strings
toLowerCase( )	Mostra a string em letras minúsculas
toUpperCase( )	Mostra a string em letras maiúsculas

Mais sobre Strings

- Convertendo String em números

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah!"; // "10 bananas, ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN
```

- Acessando caracteres

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0);// does work in IE
var lastLetter = s.charAt(s.length - 1);
```

O tipo número

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
```

- O inteiros, números reais são do mesmo tipo
- Operadores: +, -, *, /, \%, ++, --, = ,+= -=, *=, /=, \%=
  - Mesma precedência do Java
  - Muitos operadores realização conversão automática de tipos: "2" * 3 é 6

O objeto Math

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

- Métodos:
  - abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tam
- Propriedades:
  - E, PI

Valores especiais

```
var ned = null;
var benson = 9;
var n2 = parseFloat("booyah"); // NaN
var caroline;
// at this point in the code, ned is null, benson's 9
//caroline is undefined
1 / 0
Infinity
```

- undefined
  - Objeto não foi declarado, logo não existe
- null
  - Existe, mas foi atribuído com vazio
- NaN
  - Not a number, valor não válido para u número
- Infinity
  - Exibido quando um número excede o limite do JavaScript: 1797693134862315E+308

O tipo booleano

```
var iLike190M = true;
var ieIsGood = "IE6" > 0;
if ("web dev is great") {
if (0) { /* false */ }
// false
/* true */ }
```

- Qualquer valor pode ser usado como \alert{Boolean}
  - Valores para falso:
    - 0, 0.0, NaN, "", null, e undefined
- Valores para verdadeiro
  - Todos o resto
- Convertendo um valor para boolean explicitamente

```
var boolValue = Boolean(outroValor);
```

COMENTÁRIO

Comentário

```
// comentário de uma linha
/* comentário de
    Múltiplas
    linhas     */
```

- Idêntico ao do Java

INSTRUÇÕES BÁSICAS

Estrutura de controle – if/else

```
if (condition) {
statements;
} else if (condition) {
statements;
} else {
statements;
}
```

- Idêntico ao java
- Praticamente qualquer coisa pode ser usada como condição

Estrutura de controle – switch/case

```
switch (expression) {
  case value1:
    [break;]
  case value2:
    [break;]
  case valueN:
     [break;]
  default:
     [break;]
}
```

- Pode ser usado para comparar string

Operadores relacionais

Operadores	Descrição
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
==	Igualdade
!=	Diferente
===	Igualdade sem coerção
!==	Igualdade com coerção

Operadores relacionais

- A maioria dos operadores convertem os tipos automaticamente
  - 5 < “7” true
  - 42 == 42.0 true
  - ”5.0”== 5 true
- **===** e **! ==** não realizam a conversão de tipo
  - ”5.0”=== 5 false

Operadores lógicos

Operadores	Descrição
&&	E
||	Ou
!	Negação

Avaliação Curto Circuito Lógico

- && e || só executam o segundo operando, dependendo do resultado do primeiro

- Útil para checagem de objetos antes de acessar seus atributos
  - var name = o && o.getName();
- Atribuição de valor default
  - var name = otherName || "default”;

Operador ternário

```
var allowed = (age > 18) ? "yes" : "no";
```

Estruturas de Repetição - while

```
while (condition) {
    code block to be executed
}

while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

- A condição é testada antes de iniciar a execução do bloco

Estruturas de Repetição – do/while

```
do {
    code block to be executed
}
while (condition);

do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

- A condição é testada após a execução do bloco
  - O laço é executado pelo menos uma vez

Estruturas de Repetição – for

```
for (instrução 1; instrução 2; instrução 3) {
    code block to be executed
}

for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

- Instrução 1
  - Executada antes de iniciar o bloco
- Instrução 2
  - Executadas antes de cada iteração do laço
- Instrução 3
  - Executadas após a iteração do laço

Break e continue

```
for (i = 0; i < 10; i++) {
    if (i === 3) { break }
    text += "The number is " + i + "<br>";
}

for (i = 0; i < 10; i++) {
    if (i === 3) continue;
    text += "The number is " + i + "<br>";
}
```

VETORES

Vetores

```
var name = new Array();
var name = []; //empty array
var name = [value, value, ..., value]; //pre filled array
name[index] = value; //stored element

var ducks = ["Huey", "Dewey", "Louie"];

var stooges = []; // stooges.lenght e 0
stooges[0] = "Larry"; // stooges.lenght e 1
stooges[1] = "Moe"; // stooges.lenght e 2
stooges[4] = "Curly"; // stooges.lenght e 5
stooges[4] = "Shemp"; //stooges.lenght e 5
```

- Existem duas maneira de inicializar um vetor
- O tamanho do vetor aumenta de acordo com a a necessidade

Vetores

```
var a = ["Stef", "Jason"]; //Stef, Jason
a.push("Brian"); //Stef, Jason, Brian
a.unshift("Kelly"); //Kelly, Stef, Jason, Brian
a.pop(); //Kelly, Stef, Jason
a.shift(); //Stef, Jason
a.sort(); // Jason, Stef
```

- Principais métodos:
  - concat, pop, push, reverse, shift, unshift, slice, sort, splice, toString
- Vetores funcionam como estrutura de dados
  - Pilhas
    - push e pop, adicionam e removem respectivamente
  - Fila
    - unshift e shift, adicionam e e removem respectivamente

Quebrando e juntado String

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```

- split
  - Quebra a sring em partes utilizando um delimitador
  - Pode ser utilizado com expressões regulares
- join

  Transforma um vetor em uma string, utilizando um delimitador entre os elementos

EXCEÇÕES

Try/Catch

```
try {
  //  Block of code to try
}
catch(err) {
  //  Block of code to handle errors
}
finally {
  // Block of code to be executed regardless of the try / catch
result
}
```

OBJETOS

Objetos

- Simples pares nome-valor, como
  - Dicionários em Python
  - Hashes em Perl e Ruby
  - Hash tables em C e C++
  - HashMaps em Java
  - Arrays associativos em PHP
- Muito comuns, estrutura de dados versátil
- Nome é uma string e o valor pode ser qualquer coisa

Criação de Objetos

```
var objeto = new Object();
// ou
var objeto = {};
```

Acesso a Atributos

```
objeto.name = "Simon"
var name = objeto.name;
//ou
objeto["name"] = "Simon";
var name = objeto["name"];
```

Sintaxe Literal de Objetos

```
var obj = {
    name: "Carrot",
    "for": "Max",
    details: {
        color: "orange",
        size: 12
    }
}

obj.details.color // orange
obj["details"]["size"] // 12
```

Percorrendo um objeto

```
var obj = { 'name': 'Simon', 'age': 25 };
for (var attr in obj) {
    print (attr + ' = ' + obj[attr]);
}
```

FUNÇÕES

Funções

```
function name() {
statement ;
statement ;
...
statement ;
}

function myFunction() {
alert("Hello!");
alert("How are you?");
}
```

Funções e Expressões

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
var z = x(4, 3, 5);
```

O objeto argumento

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
    var i, sum = 0;
    for (i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}
```

Funções anônimas

```
var avg = function() {
    var sum = 0;
    for (var i = 0, j = arguments.length; i < j; i++) {
        sum += arguments[i];
    }
    return sum / arguments.length;
}
```

Auto-Invocação

```
(function () {
    var x = "Hello!!";      // I will invoke myself
})();
```

Funções objetos

```
function makePerson(first, last) {
    return {
        first: first,
       last: last
    }
}
function personFullName(person) {
    return person.first + ' ' + person.last;
}
function personFullNameReversed(person) {
    return person.last + ', ' + person.first
}

s = makePerson("Simon", "Willison");
personFullName(s)
personFullNameReversed(s)
```

Métodos

```
function makePerson(first, last) {
    return {
        first: first,
       last: last,
        fullName: function() {
            return this.first + ' ' + this.last;
        },
        fullNameReversed: function() {
            return this.last + ', ' + this.first;
        }
    }
}

s = makePerson("Simon", "Willison")
s.fullName()
s.fullNameReversed()
```

Construtor

```
function Person(name){
    this.name = name
    this.sayHi = function(){
        return 'Hi, I am ' + this.name
    }
}
```

O que vem por aí

- Fundamentos de JavaScript