Základy JavaScriptu

17.02.2022

Marek Nagy

HTML

- statické webové dokumenty
- značky upravujú text
- elementy vytvárajú štruktúru – strom
- dokument môže byť
  - lokálny statický súbor
  - statický súbor z webového serveru
  - dynamicky vygenerovaný súbor na strane servera (napr. cez php)

CSS

- snaha oddeliť obsah od prezentácie
- CSS2
  - trocha dynamiky (:hover)
- CSS3
  - transition, animation, …
  - flexbox, grid, ...
  - selektory, funkcie, premenné, ...

JavaScript

- firma Netscape
- pridanie jednoduchých príkazov
  - prístup do stromu dokumentu
  - manipulácia s elementami (napr. zmena obrázkov)
- zavedenie udalostí dokumentu
  - onclick, onmouseover, onmouseout, …
- interpretovaný jazyk
  - zavedený štandard ECMAScript
  - každoročné revízie (ES2017, ..., ES2020)

JavaScript (JS)

- rozširovanie možností (štandardu)
  - objekty, triedy, iné udalosti, …, ES2017
  - paralelizmus
  - zdielaná pamäť, …
  - object rest, spread, …, ES2018

- predkompilovanie kusov kódu
  - JIT (Just-In-Time)
  - zrýchlenie

Vkladanie JS

- priamo do HTML dokumentu

Netreba uvádzať atribút type.

```
<script>

  console.log ('Ahoj, môj script!')

</script>
```

- odkazom na externý súbor

```
<script src='moj.js'></script>
```

Komentáre v JS

- ignorovanie pri vykonávaní

```
// Komentár


/*
Dlhší komentár
cez viac riadkov
*/
```

Príkazy

- medzi príkazmi
  - oddelovacia bodkočiarka
- rezervované mená
  - nemožno použiť na meno premennej
- prázdne znaky
  - medzery, nové riadky
  - sú ignorované

Názvy premenných

- rozlišovanie malých a veľkých písmen
- viacslovné
  - cammelCase
  - first_name
- môžu začínať aj _ alebo $

Základné typy premenných

- číslo (double - 64bits)

Najväčšie celé číslo, ktoré možno reprezentovať v double je $\pm 2^{53}-1$. Pre veľké celé čísla zaviedli typ BigInt.

```
a = 4; b = 3.14;
```

- boolean

```
a = false; g = true;
```

- reťazec (utf-16)

```
d = 'Čau';
// d[0] == 'Č';      a = "  "; a = "你好";
```

- pole

```
e = [1, 2, 10, 23, 1, 8];

w = ['auto', 'bicykel', 6];

// e[3] == 23
```

Vetvenie

```
if (x == 2) {
}


if (x == 6) {
}
else {
}


switch (doprava) {
  case 'auto':
    break;

  case 'kolobezka':
    break;

  default:
}
```

Testovacie operátory

- porovnanie hodnoty

  `==` `!=` `<` `>` `<=` `>=`

- porovnanie hodnoty aj typu

  `===` `!==`

- logické

  `&&` `||` `!` `??`

```
a = x || 10
a = x ?? 10       // Ak x nemá hodnotu priradí sa 10
a ??= 10
```

Aritmetické operátory

- aritmetické

  `+  -  *  /  %  ++  --`

- priradenie

  `=`

- aritmetické s priradením

  `+=  -=  *=  /=  %=`

Podmienený operátor

podmienka ? hodnota1 : hodnota2

```
a = i==1 ? 'dobre' : 'zle';
```

Operátory s reťazcami

- spojiť reťazce

  +

- spojiť a priradiť

  +=

```
s = 'Ahoj'; i = 3;

s += ' Ferko. Daj mi '+i+' hrušky';
```

Reťazcové šablóny

- back-ticks
- vloženie výrazov

  ${ výraz }

```
vek = 7;
a = 'Ahoj. Dnes máš '+(vek+1)+' rokov.';
a = `Ahoj. Dnes máš ${vek+1} rokov.`;
```

Cykly

```
for (i = 0; i < 10; i++) {

}



i = 0;
while (i < 10) {
  i++;
}



i = -1;
do {
  i++;
} while (i >= 10);
```

Prerušenie cyklu

```
s = [1,2,3,4,5,6];


for (i = 0; i < 10; i++) {

  if (s[i] == 3) break;

  s[i]++;
}



for (i = 0; i < 10; i++) {

  if (s[i] == 3) continue;

  s[i]++;
}
```

try – catch

- vznikajú chyby, ktoré sa šíria ako udalosť
- ak sú neodchytené, rieši ich prehliadač

```
try {

  // Kód, ktorý môže generovať chybu. Aj syntaktickú.

  fooor (i = 0;;);

}

catch (err) {
  console.log (err);
}

finally {
  // Kód sa vykoná bez ohľadu, či bola alebo nebola chyba
  // Vykoná sa na konci try aj na konci catch bloku
}
```

throw

- vygenerovanie / hodenie vlastnej chyby

```
try {

  for (i = 0; i < 10; i++)
    if (i%2 == 1) throw 'Nepár';

}

catch (txt) {
  console.log (txt);
}
```

Špeciálne hodnoty premenných

- undefined
  - nedefinovaná (nepriradená) hodnota

```
if (a === undefined) {}
```

- NaN
  - not a number
  - napr. výsledok delenia nulou

```
if ( isNaN(x) ) {}
```

- Infinity
  - nekonečno

Definovanie funkcie

- názov + zoznam argumentov
- návratová hodnota undefined
  - return vráti inú hodnotu

```
function log (txt) {
  console.log (txt);
}


function Scitaj (a, b) {
  return a+b;
}


log ('Ahoj');

c = Scitaj (3, 5);
```

Argumenty funkcie

- argumenty sa priraďujú zľava doprava

Argumentu možno pridať preddefinovanú hodnotu. Použije sa, ak volanie funkcie nemá toľko argumentov. Ináč má hodnotu undefined.

```
function Spracuj (meno, priezvisko='Hraško', vek=7) {

  if (meno === undefined) meno = 'Janko';

  return `${meno} ${priezvisko} bude mať ${++vek} rokov`;

}


a = Spracuj ('Marek', 'Nagy');
a = Spracuj ();
```

Dynamické preddefinované hodnoty

- hodnoty sa priraďujú v čase zavolania
  - hodnoty sú počítané a priraďované zľava doprava

```
function Moja (a=1, b=[a,3,4], c=4, d = Sucet(a,c), e) {

  return b;
}


Moja (10);   // [10,3,4]
```

Premenlivý počet argumentov

- zvyšok argumentov
- uloží sa do poľa

```
function Scitaj (a, ...zvysok) {

  // a == 1
  // zvysok == [2,3,4,5]

}


c = Scitaj (1, 2, 3, 4, 5);
```

Anonymné funkcie

- nemajú meno
- treba ich priradiť

```
moja = function (x) {return x+3};

c = moja (5);
```

Lokálne premenné (var)

- viažu sa ku funkcii

```
function Moja (a, b) {
  var sum;

  sum = a+b;

  return sum;
}
```

- deklarácie sa vytiahnu na začiatok funkcie
  - potom sa deklarovanie ignoruje (priradenie však nie)

```
function Pocitaj (r) {
  c = 3;
  var a = 3, b = 9;
  a *= r;
  if (r == 3) {
    var c = a;
    var a = 100;
  }
}
```

Blokové premenné (let)

- vytvorené a platné iba vrámci bloku {}
  - vnorené bloky vidia premennú tiež

```
function Sucet (a, b) {
  let x = 1, y = 2;

  if (x == 1) {
    let y = 0;
    x += y;
  }

  for (let y = 0; y < 10; y++) {x += y}
}
```

Blokové konštanty (const)

- podobne ako let
- platné iba vrámci bloku
  - prípadne vnorené bloky
- hodnotu nemožno zmeniť (vygeneruje sa chyba)

```
const PI = 3.14;
```

… syntax

...rest

- zlúči elementy do poľa

...spread

- expanduje elementy z poľa

```
function Sucet (a,b,c, ...rest) {return a+b+c}

Sucet (...[1,2,3]);   // 6


var p = [1,2,3]
var q = [0, ...p, 4]      // [0,1,2,3,4]
```

Deštrukturalizácia poľa

- priradenie podľa štruktúry

```
var x = [1,2,3];

let [a, b, c] = x;

// Bez let,var,const
let m, n, o;
[m, n, o] = x;

// Defaultné hodnoty
[m, n = 2, o = 4] = [1, 5];

// Ignorovanie hodnoty
[m, ,n] = x;

// Výmena hodnôt
[a, b] = [b, a];

// Zvyšok
let [p, ...rest] = x;
```

Šípkové funkcie

- sú anonymné
- jednoduchšia syntax

(args) => {telo}

Ak je len jeden príkaz v tele, netreba písať zátvorky. Automaticky sa pred ním predpokladá return príkaz.

(x) => x*3 // {return x*3}

```
x => x*3;

// function (x) {return x*3}
```

Ak je len práve jeden argument, zátvorky netreba písať.

Funkcie s reťazcami

- dĺžka

```
var s = ' Ahoj, Ferko ';
for (let i = 0; i < s.length; i++) log (s[i]);
```

- rozloženie podľa oddeľovača

```
b = s.split(',');          // b=[" Ahoj", " Ferko "]
```

- nájdenie pozície danej hodnoty

```
i = s.indexOf ('Fer');  // i=7
```

- orezanie prázdnych znakov z krajov

```
ns = s.trim ();            // ns="Ahoj, Ferko"
```

Funkcie s poľom

- veľkosť

```
var a = [3,4,5];
log (a.length);
```

- vkladanie/vyberanie z konca

```
a.push (3);   // [3,4,5,3]
b = a.pop (); // 3
```

- vkladanie/vyberanie zo začiatku

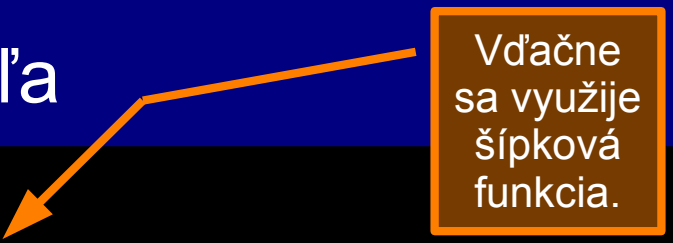
```
a.unshift (9); // [9,3,4,5]
b = a.shift(); // 9
```

- hľadanie prvku (-1 → nenašiel sa)

```
var i = a.indexOf (4); // 1
i = a.indexOf (100);   // -1
```

Funkcie s poľom

- spracuj každý prvok poľa




```
var a = [3,4,5];

a.forEach ( x => console.log (x*3) );
```

- utrieď

```
b = a.sort ( (a,b) => a-b ); // zmení pole
```

- filtruj prvky podľa kritéria

```
b = a.filter ( x => x >= 4); // vytvorí nové pole [4,5]
```

- nájdi prvok

```
c = a.find (x => x >= 5); // 5
j = a.findIndex (x => x >= 5); // 2
t = a.includes (4) // true
```

Funkcie s poľom

- zmena prvkov poľa (začiatok, počet) , indexuje sa od 0

```
var a = [3,4,5];
b = a.splice (2,1);   // b=[5], a=[3,4]; zmení pole a
```

- selekcia prvkov (začiatok, koniec)

```
b = a.slice (1,2);    // vytvorí nové pole b=[4]
```

- premapovanie prvkov

```
b = a.map ( x => 3*x); // vytvorí nové pole b=[9,12,15]
```

- vytvor reťazec

```
b = a.join (':');       // b='3:4:5'
```

- akumulácia hodnoty

```
var a = [3,4,5];
b = a.reduce ((aku,x) => aku+x, 10);   // b= 10+3+4+5 = 22
```

Funkcie s poľom

- zistí, či nejaký prvok spĺňa podmienku

```
let a = [3,4,5];
a.some (x => x > 4);     // true
```

- zistí, či všetky prvky spĺňajú podmienku

```
let a = [3,4,5];
a.every (x => x > 4);     // false
```

- priraďuje hodnoty

```
let a = [3,4,5];
a.fill (0);     // zmení pole a=[0,0,0]
```

Ďakujem za pozornosť