



JavaScript klient Web API

kurz Webové technológie
Eduard Kuric

- JS úvod, základné konštrukcie
- OO vs. funkcionálne programovanie
 - imperatívny vs. deklaratívny štýl
- Koncepty - multiparadigmový, prototypovací, dynamický jazyk
- Objekty, dedenie (prototype chaining)
- Vloženie JS do HTML dokumentu, režimy vykonania
- Web API – manipulácia s dokumentmi
- Udalosti, delegovanie udalosti

JavaScript – 3. pilier (HTML,CSS,JS)

- **je notoricky známy tým, že je najviac nepochopený programovací jazyk na svete**
- **často vnímaný ako hračka**
 - za klamlivou jednoduchosťou sa skrýva silný jazyk s množstvom zaujímavých vlastností
- v súčasnosti najmä na tvorbu moderných webových stránok/aplikácií
 - plne interaktívny, dynamicky menený obsah
- posledné roky významný posun
 - hlbšie znalosti jazyka + technológií
 - dnes nutnosť každého dobrého web vývojára

História stručne

- vznikol v 1995 - Brendan Eich
 - v čase keď bol inžinierom v Netscape
- prvýkrát bol uverejnený v Netscape Navigator 2, v 1996
- pôvodný názov ***LiveScript***
 - premenovali ho, lebo chceli využiť popularitu Javy (aj keď majú málo spoločného)
 - čo sa neskôr ukázalo ako nie moc šťastné rozhodnutie
- o niekoľko mesiacov MS vydal *JScript* v IE3
 - +- kompatibilný s JS
- Netscape požiadal o šandardizáciu JavaScriptu Európsku šandardizačnú organizáciu - Ecma Int.
 - v 1997 vzniklo prvé vydanie štandardu ECMAScript 1

História stručne - ECMAScript

- ECMAScript je štandard pre skriptovacie jazyky
- JavaScript je implementáciou ECMAScriptu
- akronym ES - ECMAScript

História stručne /3

- v **1999** významný posun JS – ECMAScript 3
- ECMAScript 4 nebol vydaný, kvôli rozdielnym názorom na zložitosť jazyka s navrhovanými zmenami
- viaceré časti z ES4 boli neskôr základom pre ES5 (**2009**) a ES6 (2015)
- od ES6 ročný cyklus uvoľňovania novej verzie
- aktuálne ES 2019
- všetky moderné prehliadače podporujú ES5

I/O koncept

- nariadenie od väčšiny jazykov nemá štandardný input/output koncept
- je navrhnutý ako skriptovací jazyk
 - funguje v „hostiteľskom prostredí“
 - je na prostredí, aby poskytlo mechanizmy na komunikáciu s okolím
 - prostredie - spravidla webový prehliadač, “interpreterov” je ale viacero (Photoshop, Node.js, NoSQL DB)

Interpretovaný?

- interpretovaný
 - kód vykonáva interpreter (postupne sa interpretuje)
- Chrome V8
 - Chrome, Node.js : JIT (just-in-time) kompilácia
 - za behu optimalizácia + kompilácia, neprodukuje ale bytecode

Multiparadigmový

- multiparadigmový dynamický programovací jazyk
 - OO prototypovací jazyk
 - funkcionální jazyk
- syntax podobná s Java, C

Imperatívny vs. deklartívny štýl

- **imperatívny štýl programovania**
 - presná postupnosť príkazov (algoritmus),
ako danú úlohu vyriešiť
 - 2 podtypy jazykov – procedurálne (C), OO (Java)
- **deklaratívny štýl programovania**
 - špecifikuje cieľ - **čo sa má spraviť** a nie ako sa to má spraviť
 - algoritmizácia je ponechaná na interpreterovi,
napr. jazyk SQL

Imperatívny vs. deklartívny štýl

- Porovnajme
 - [imperative.js](#)
 - [declarative.js](#)

OO programovanie

- imperatívny štýl (paradigma) programovania
- v OO svete – objekty
 - obsahujú informácie (atribúty) a operácie (metódy), ktoré sa vzťahujú na rovnaký koncept
 - atribúty reflektujú stav objektu
 - metódy umožňujú meniť/ovplyvňovať stav
- **zjednodušene – premenné sú objekty**

(Rýdzo) funkcionálne programovanie

- deklaratívny štýl (paradigma) programovania
- kód je kombináciou funkcií
 - hodnoty definovaných premenných sú nemenné
 - funkcia nie je schopná meniť vonkajší „svet“, jej výstup ovplyvňujú iba argumenty
 - logika je oddelená od údajov
 - napr. jazyk Haskell - čisté FP
- **zjednodušene – premenné sú hodnoty, najmä však funkcie**

JavaScript je hybrid

- keď
 - v OO môžeme premenné považovať za objekty
 - vo FP môžeme premenné považovať za funkcie
- **Nemožeme funkcie považovať za objekty?**
- **v JavaScripte môžeme!**

Čo je JavaScript?

- **prototypovací OO programovací jazyk**
 - v ktorom nemusia byť triedy (prototypy) definované explicitne
 - môžu byť „dodefinované/doplnené“ dynamicky, pridávaním atribútov a metód za behu
 - alebo (menej často) pridaním do prázdneho objektu
- **prototyp v JS**
 - môžeme zdefinovať pomocou funkcie – tzv. constructor function
 - môžeme ju chápať aj ako JavaScriptovú podobu klasickej triedy v OO
 - v JS sú funkcie prvotriedne objekty (first-class objects), lebo môžu obsahovať atribúty a metódy

Triedu môžeme definovať takto...

```
function Apple(type) {  
  this.type = type;  
  this.color = "red";  
  
  this.getInfo = function() {  
    return this.color + ' ' + this.type + ' apple';  
  };  
}  
  
// vytvorime instanciu/objekt daného prototypu  
var a = new Apple;  
  
> "red undefined apple"
```


Atribúty, metody...dynamicky

```
function Apple(type) {  
    this.type = type;  
    this.color = "red";  
}
```

```
Apple.prototype.getInfo = function() {  
    return this.color + ' ' + this.type + ' apple';  
};
```

```
var a = new Apple;
```

Môžeme rovno vytvoriť objekt...

```
var apple = {  
  type: "macintosh",  
  color: "red",  
  getInfo: function () {  
    return this.color + ' ' + this.type + ' apple';  
  }  
}
```

Pridať do prázdneho objektu...

```
var person = new Object();  
  
person.name = 'Peter';  
person['age'] = 33;  
person.greeting = function() {  
    alert('Ahoj! Volam sa ' + this.name + '.');  
};
```

Prototype chaining



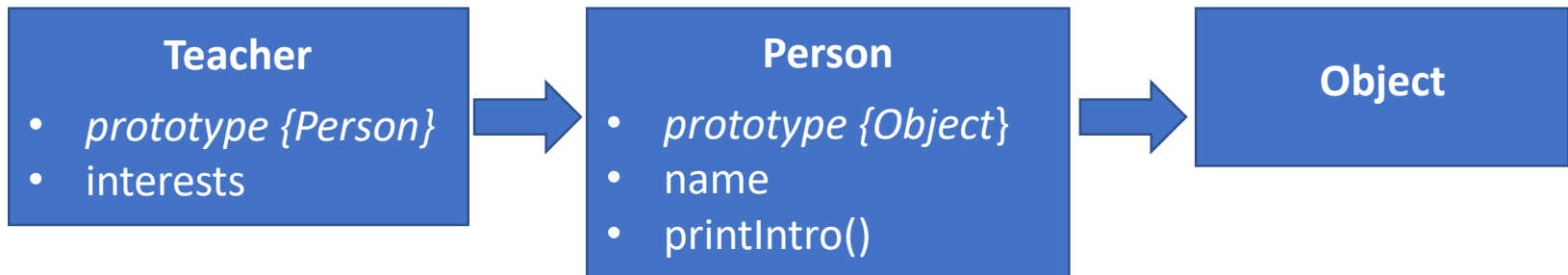
```
function Person(name) {  
    this.name = name;  
    this.job = 'Osoba';  
};
```

```
Person.prototype.printIntro = function() {  
    alert('Ahoj! Volam sa ' + this.name + '.');  
};
```

```
var person1 = new Person('Peter');  
person1.__proto__; // tiez Object.getPrototypeOf(person1)  
person1.__proto__.__proto__;
```

```
Person.prototype;
```

Prototype chaining /2



```
function Teacher(name, interests) {  
    Person.call(this, name);  
    this.interests = interests;  
}
```

```
var teacher1 = new Teacher('Peter', 'math, programming');
```

```
Teacher.prototype = Object.create(Person.prototype);  
var teacher1 = new Teacher('Peter', 'math, programming');
```

Prototype chaining /3

- slúži na vytváranie nových objektov založených na existujúcich objektoch
- keď vytvárame nový objekt (prototyp) z/založený na inom objekte (prototype)

- `var teacher = Object.create(person) ;`
- `Teacher.prototype =
Object.create(Person.prototype) ;`

všetky atribúty a metódy sú automaticky „pridané“
do nového objektu - cez atribút `prototype`

- (`__proto__`) klasický JavaScript objekt, ktorý odkazuje na atribúty a metódy „nadradeného“ prototypu

Takto zrealizujeme dedenie...

```
function Person(first, last, age, gender) {  
    this.name = {  
        first,  
        last  
    };  
    this.age = age;  
    this.gender = gender;  
};  
  
Person.prototype.greeting = function() {  
    alert('Ahoj! Volam sa ' + this.name.first + '.');  
};
```

... takto zrealizujeme dedenie

```
function Teacher(first, last, age, gender, interests) {  
    Person.call(this, first, last, age, gender);  
  
    this.interests = interests;  
}
```

- vytvorme objekt typu Teacher
 - `var teacher1 = new Teacher;`
 - atribúty a metódy nie sú automaticky prepojené pri použití `call()`

- použitím `Object.create()` musíme prepojiť prototypy

```
Teacher.prototype = Object.create(Person.prototype);
```

```
var teacher1 = new Teacher('Peter', 'math,  
programming');
```


... takto zrealizujeme dedenie /2

- prepojením prototypov sa nastaví konštruktor

```
Teacher.prototype.constructor na Person
```

```
// vypisme si
```

```
teacher1.constructor.name
```

- prototypu Teacher nastavíme správny konštruktor

```
Teacher.prototype.constructor = Teacher;
```

...create aj na inštanciu

```
var person = {  
  isHuman: false,  
  printIntroduction: function () {  
    console.log('Moje meno ${this.name}. Som clovek?  
                ${this.isHuman}');  
  }  
};  
  
var me = Object.create(person) ;  
  
// name je atribut me, nie person  
me.name = "Peter";  
// zdedeny atribut prepiseme  
me.isHuman = true;  
// "Moje meno Peter. Som clovek? true"  
me.printIntroduction();
```

Čo je JavaScript?

- prototypovací OO programovací jazyk
- **obsahuje prvky funkcionálneho programovania**
 - má prvotriedne funkcie (first-class functions)
 - premenná môže byť funkcia
 - funkcia môže byť odovzdaná ako argument inej funkcii
 - funkcia môže vrátiť funkciu
 - anonymné funkcie (Lambda výrazy)
 - funkcie vyššieho radu (higher-order functions)

Funkcie vyššieho radu

```
function noisy(f) {  
  return (...args) => {  
    console.log("calling with", args);  
    let result = f(...args); // Math.min([3,2,1])  
    console.log("called with", args, ", returned", result);  
    return result;  
  };  
}  
  
noisy(Math.min)(3, 2, 1);  
// calling with [3, 2, 1]  
// called with [3, 2, 1]  
// result: 1
```

Arrow funkcje

```
const power = (base, exponent) => {  
  let result = 1;  
  for (let count = 0; count < exponent; count++) {  
    result *= base;  
  }  
  return result;  
};
```

// VS. function expression

```
const power = function(base, exponent) {  
  ...  
}
```

// pozn. arrow iba callable, decl. a expr. constructable
(new) a callable

Premenné

- nie je potrebné udávať typ premennej
 - dynamické priradenie typov
- 6 jednoduchých typov
 - Boolean true and false.
 - null (nie Null, ani NULL, JS je case sensitive)
 - null je špeciálny objekt, lebo typeof null vrati object
 - undefined (ak hodnota nie je definovaná)
 - premenná je deklarovaná, hodnota nie je priradená
 - number (celé číslo, s pohyblivou čiarkou)
 - string
 - [Symbol \(od ES6\)](#)
- Object

Premenné /2

- `var`
 - ak je deklarovaná vo funkcii, tak platí iba v nej
 - inak je platnosť globálna
- `let`
 - platnosť iba na úrovni bloku, v ktorom je deklarovaná
- `const`
 - nemôže byť reдекlarovaná
 - hodnota sa nemôže meniť opätovným priradením
 - platnosť iba na úrovni bloku, v ktorom je deklarovaná

Premenné - hoisting

```
// môžeme odkazovať na premennú, aj keď je  
// deklarovaná neskôr, nedostaneme exception,  
// „iba“ undefined
```

```
console.log(x === undefined); // true  
var x = 3;
```

```
var x;  
console.log(x === undefined); // true  
x = 3;
```

```
// pri let to neplatí, dostaneme error  
console.log(x); // ReferenceError  
let x = 3;
```


Funkcie - hoisting

```
/* deklarácia funkcie OK */  
foo(); // "bar"  
function foo() {  
    console.log('bar');  
}
```

```
/* function expression NIE */  
baz(); // TypeError: baz is not a function  
var baz = function() {  
    console.log('bar2');  
};
```

Konfliktné názvy

```
// error  
function f() {};  
const f = 5;
```

```
// ERROR  
function f() {  
    const g = 5;  
    var g;  
}
```

```
// OK, atribúty objektov nie sú konštanty  
const MY_OBJECT = {'key': 'value'};  
MY_OBJECT.key = 'otherValue';
```

Úvodzovky

- pri reťazcoch môžeme použiť jednoduché aj dvojité úvodzovky

```
var s1 = 'Jednoduche uvodzovky';
```

```
var s2 = "Dvojite uvodzovky";
```

```
var s3 = 'What on earth?"; // CHYBA
```

```
var s4 = "I'm feeling blue."; // OK
```

Spájanie reťazcov

- operátor +

```
var res = one + '- status OK -' + two;
```

Typové konverzie

```
typeof "John"           // "string"
typeof 3.14              // "number"
typeof NaN               // "number"
typeof false             // "boolean"
typeof [1,2,3,4]         // "object"
typeof {name:'John', age:34} // "object"
typeof new Date()        // "object"
typeof function () {}    // "function"
typeof myCar              // "undefined" *
typeof null              // "object"
```

Typové konverzie /2

- **Number -> String**
 - `String(100 + 23); (123).toString();`
- **Boolean -> String**
 - `String(false); String(true); false.toString();`
- **Date -> String**
 - `String(Date()); Date().toString();`
- **String -> Number**
 - `Number("3.14");`
- **Boolean -> Number**
 - `Number(false);`
- **Date -> Number**
 - `Number(new Date()); // pocet ms od epochy - 1.1.1970`
- **String -> Boolean**
 - `X Boolean('true'); // true`
 - `X Boolean('false'); // true`
 - `X Boolean('0'); // true`
 - `OK var isTrueSet = (myValue == 'true');`

Metódy na prácu s reťazcami

```
var browserType = 'chrome';

browserType.length;           // 6
browserType[0];               // 'c'
browserType.indexOf('ome');   // 3 – od 0
browserType.slice(0,3);       // 'chr',
browserType.slice(2);
browserType.toUpperCase();     // 'CHROME'
// tiež toLowerCase();
browserType.replace('ch','');  // 'rome'
```

Vloženie JavaScriptu do stránky

- interný JavaScript

- element `<script></script>` pred ukončovaciou značku `</body>` elementu

- externý JavaScript

- cesta k zdrojovému súboru v atribúte `src`
`<script src="script.js"></script>`

- inline – pokiaľ možno nepoužívať, lepšie zaregistrovať listener

```
<button onclick="createParagraph()">  
    Click me!  
</button>
```


Vykonanie JavaScriptu

- je spravidla spracovávaný v poradí, v akom naň prehliadač pri načítavaní stránky natrafí
 - zhora-nadol
 - musíme dbať na poradie, ak sú medzi skriptami závislosti
- načítanie a vykonanie JS blokuje ďalšie spracovanie HTML dokumentu
 - preto sa vkladá JS väčšinou na konci dokumentu
- atribúty `defer` a `async`

Vykonanie JS – atribút `defer`

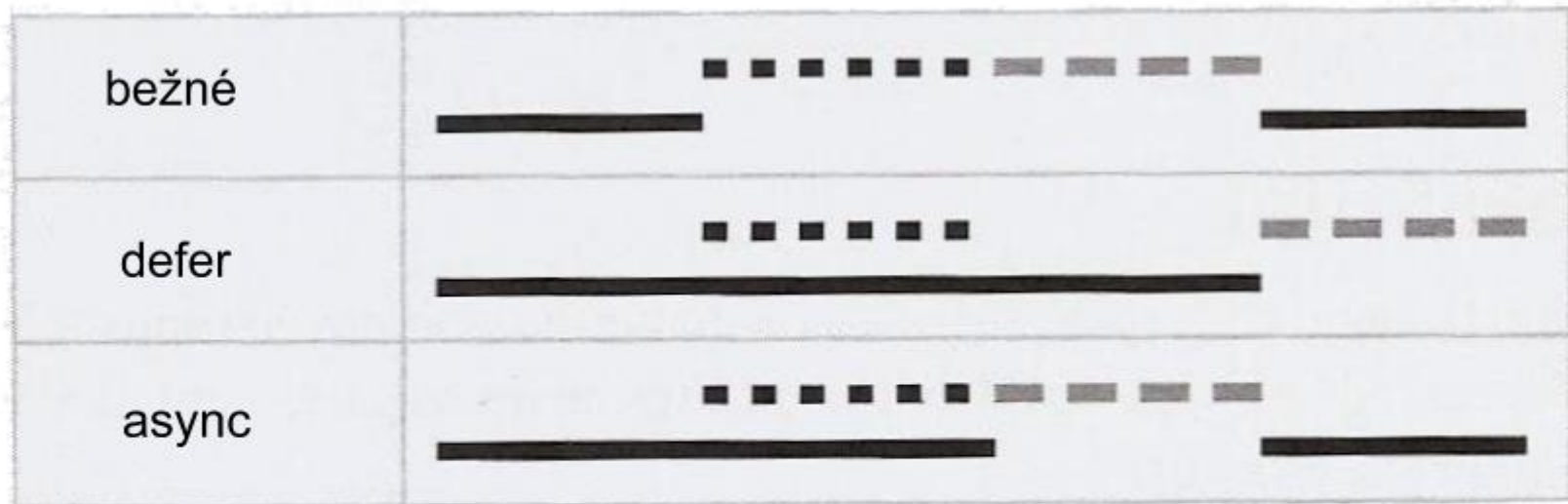
- indikuje prehliadaču, aby paralelne pársoval dokument a načítaval skript(y), jeho/ich vykonanie odložil do momentu, kedy dokončí pársovanie zvyšku HTML dokumentu




```
<script src="script.js" defer></script>
```

Vykonanie JS – atribút `async`

- indikuje prehliadaču, aby **paralelne pársoval dokument a načítal skript**, na ktorý natrafí, pričom ho vykonal hneď, ako je to možné (po načítaní)
 - pársovanie dokumentu je počas vykonávania prerušené
- je to kompromis, neblokuje spracovanie dokumentu pokiaľ načítava skript
- nevýhodou je, že nie je zaručené poradie vykonania skriptov
 - niektorý zo skriptov môže byť načítaný skôr, pričom v dokumente sa vyskytuje neskôr

Vykonanie JS



 parsovanie HTML
 načítanie skriptu
 vykonanie skriptu

JavaScript WEB API

API prehliadačov

- Manipulácia s dokumentmi (DOM)
- Získavanie údajov zo servera (AJAX, Fetch API)
- Vykreslenie a tvorbu grafiky (Canvas, WebGL)
- Audio a Video (HTMLMediaElement, Web Audio API, WebRTC)
- Prístup k zariadeniam (Geolocation, Notifications, Vibration API)
- Ukladanie údajov na strane klienta (Web Storage, IndexedDB)

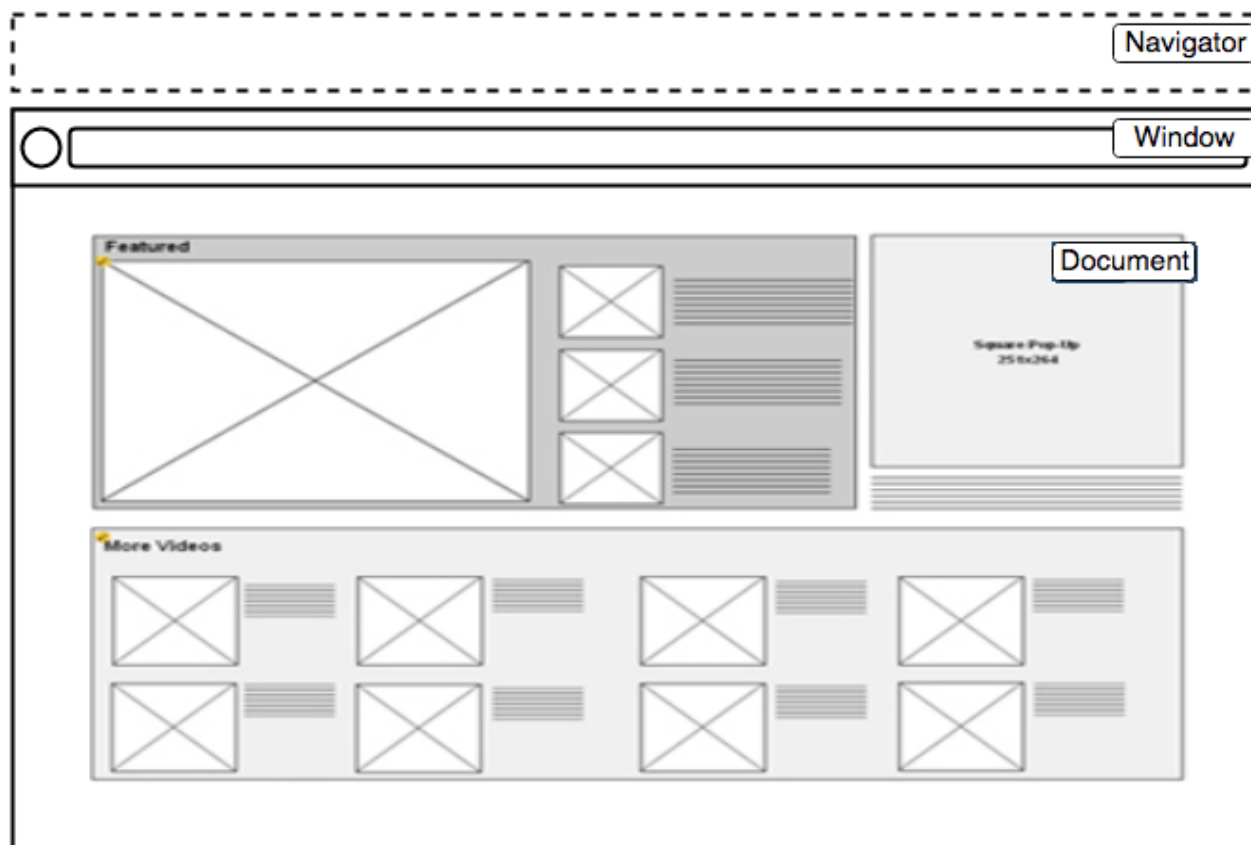
API 3. strán

- [Google Maps API](#)
- [YouTube API](#)
- [Facebook API](#)

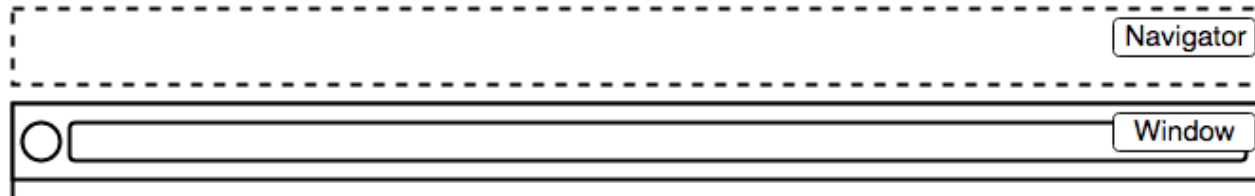
- Rôzne JS knižnice (jQuery, SystemJS, ...)
- Rámce (frameworks), Vue.JS, AngularJS, React, ...

Manipulácia s dokumentami

Časti webového prehliadača

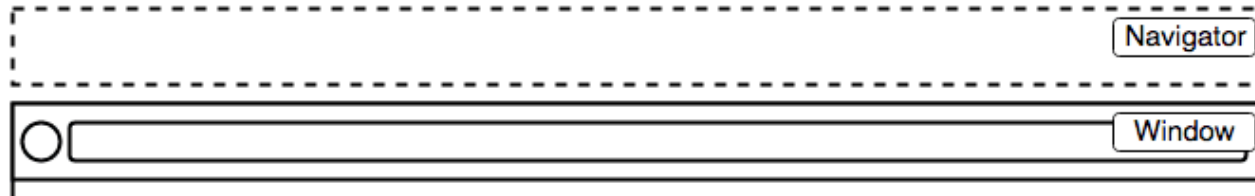


Časti webového prehliadača



- `navigator`
 - `.geolocation` - informácie o lokalite používateľa /používateľ je notifikovaný pri žiadosti o sprístupnenie informácie/
 - `.userAgent` - informácie o prehliadači
 - `.cookieEnabled` - indikácia, či sú cookies povolené
 - `.language` - preferovaný jazyk používateľa (BCP-47 formát)

Časti webového prehliadača



- `window` – karta prehliadača
 - `.screen` – napr. rozlíšenie
 - `.screenX` – hor. vzdialenosť ľavého horného rohu okna od ľavej strany obrazovky
 - `.innerWidth` – vnútorná šírka viewportu okna (ak je vykreslený, vrátane scrollbaru)
 - `.scrollX` – počet pixelov, o ktorý je dokument aktuálne posunutý od ľavého rohu viewportu
 - `.localStorage` – úložisko dát, nemá exp. čas
 - `.sessionStorage` – úložisko dát, má exp. čas

DOM

- JS poskytuje API na manipuláciu s DOM (Document Object Model) danej stránky

```
// moderný prístup, umožňuje používať css selektory  
document.querySelector('a');  
document.querySelectorAll(),
```

```
// pôvodné metódy, kompatib. so staršími prehliadačmi  
document.getElementById()  
document.getElementsByTagName()
```

Vytvorenie elementu

```
var mySection = document.querySelector('section');  
var paragraph = document.createElement('p');
```

```
<section></section>
```

Vytvorenie elementu

```
var mySection = document.querySelector('section');  
var paragraph = document.createElement('p');  
paragraph.textContent = 'Dnes je to o JS.';  
mySection.appendChild(paragraph);
```

```
<section>  
    <p>Dnes je to o JS.</p>  
</section>
```

Vytvorenie elementu

```
var mySection = document.querySelector('section');  
var paragraph = document.createElement('p');  
paragraph.textContent = 'Dnes je to o JS.';  
mySection.appendChild(paragraph);
```

```
var text = document.createTextNode(' - hurrrá');  
var linkParagraph = document.querySelector('p');  
linkParagraph.appendChild(text);
```

```
<section>  
    <p>Dnes je to o JS. - hurrrá</p>  
</section>
```

Odobratie elementu

```
var mySection.removeChild(linkParagraph);
```

```
<section>  
  <p>Dnes je to o JS. – hurrá</p>  
</section>
```

- nie je metóda, ktorá umožňuje odobrať samého seba,
dá sa to takto:

```
linkParagraph.parentNode.removeChild(linkParagraph);
```


Štýly

- `HTMLElement.style`
- `var para = document.querySelector('p');`

```
para.style.color = 'white';
```

```
// vsimnime si, v CSS pomlcka,
```

```
// v JS lower camel case
```

```
para.style.backgroundColor = 'black';
```

```
para.style.padding = '10px';
```

```
para.style.width = '250px';
```

```
para.style.textAlign = 'center';
```

Štýly /2

- lepší spôsob je zdefinovať si štýly a meniť ich dynamicky nastavením triedy

```
.highlight {  
  color: white;  
  background-color: black;  
  padding: 10px;  
  width: 250px;  
  text-align: center;  
}
```

```
para.setAttribute('class', 'highlight');
```

Udalosti

```
var btn = document.querySelector('button');

btn.onclick = function() {
    var rndCol = 'rgb(' + random(255) + ',' +
        random(255) + ',' + random(255) + ')';
    document.body.style.backgroundColor = rndCol;
}

// btn.onclick = bgChange;
```

Udalosti /2

- `btn.onfocus`
- `btn.onblur`
- `window.onkeypress`
- `window.onkeydown`
- `window.onresize`
- `window.onkeyup`
- `btn.onmouseover`
- `btn.onmouseout`

Udalosti *EventListener

```
btn.addEventListener('click', function() {  
    var rndCol = 'rgb(' + random(255) + ',' + random(255)  
        + ',' + random(255) + ')';  
    document.body.style.backgroundColor = rndCol;  
});
```

```
btn.addEventListener('click', bgChange);  
btn.removeEventListener('click', bgChange);
```

- pozn. horsia kompatibilita v starsich prehliadacoch (IE9)

Event object

- poskytuje ďalšie informácie o udalosti

```
var divs = document.querySelectorAll('div');

for (var i = 0; i < divs.length; i++) {
    divs[i].onclick = function(e) {
        e.target.style.backgroundColor = bgChange();
    }
}
```

Odstavenie pôvodného správania

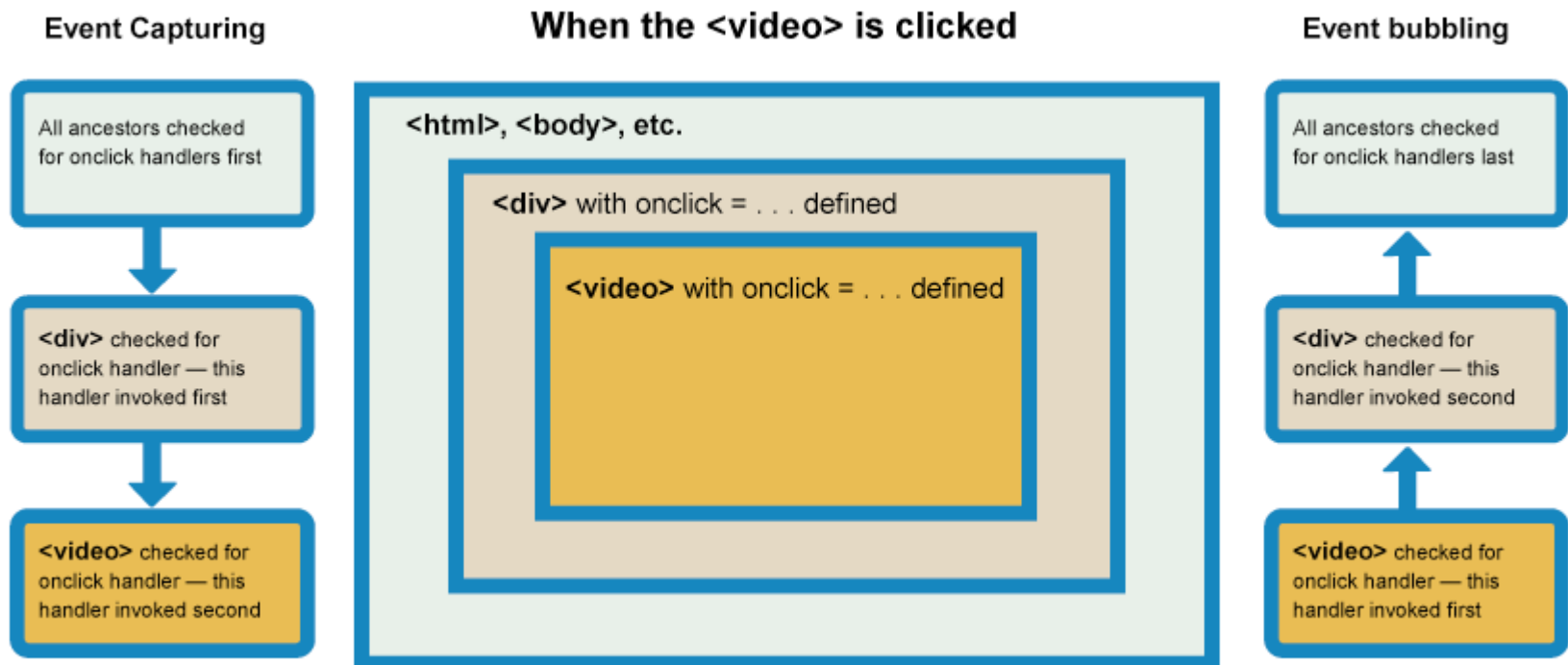
- niekedy sa stretneme so situáciou, keď chceme zastaviť, aby udalosť vykonala svoju predvolenú aktivitu

```
form.onSubmit = function(e) {  
    if (fname.value === '' || lname.value === '') {  
        e.preventDefault();  
        para.textContent = 'Musíte vyplniť celé meno!';  
    }  
}
```

Zachytenie a prebublanie udalosti

- keď je udalosť vyvolaná na určitom elemente, ktorý má predkov, prehliadače môžu vykonať registráciu udalosti v režime:
 - zachytenie (angl. capturing)
 - prebublanie (angl. bubbling)

Zachytenie a prebublanie udalosti



Zachytenie a prebublanie udalosti

- v súčasnosti moderné prehliadače registrujú udalosti v režime – **prebublanie**

- prebublávanie pozastavíme cez `stopPropagation`

```
video.onclick = function(e) {  
    e.stopPropagation();  
    video.play();  
};
```

- ak by sme chceli vynútiť režim – zachytenie, nastavíme pri registrácii udalosti
 - `addEventListener()`, tretí voliteľný parameter na `true`

Delegovanie udalosti

- chceme vykonať kód, ktorý sa má vykonať na viacerých potomkoch
- namiesto, aby sme registrovali udalosť pre každého potomka, zaregistrujeme udalosť na najvyššom možnom rodičovi
 - prebublávaním, prejdeme cez potomkov, cez `e.target` vieme prísť k aktuálnemu potomkovi

```
<ul id="parent-list">  
  <li id="post-1">Item 1</li>  
</ul>
```

```
document.getElementById("parent-list").addEventListener("click", function(e) {  
  if(e.target && e.target.nodeName == "LI") {  
    }  
});
```

Zdroje

- [Eloquent JavaScript](#)
- [Functional Programming in JavaScript](#)