

Tehnologii Web

programare Web



JavaScript la nivel de server – Node.js
(aspecte esențiale)

“Cine a văzut vreodată o bijuterie frumos cizelată
de bijutier cu ajutorul ciocanului?”

Jan Amos Comenius

JavaScript

un limbaj de programare pentru Web

Inventat de Brendan Eich (1995)
denumit inițial **Mocha**, apoi **LiveScript**

Oferit în premieră de *browser*-ul Netscape Navigator

Adaptat de Microsoft: **JScript** (1996)

Standardizat ca **ECMAScript**: ECMA-262 (1997)

www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

ECMAScript

versiunea standardizată actuală: 5.1 (iunie 2011)

www.ecma-international.org/publications/standards/Ecma-262.htm

versiunea în lucru: 6.0 – ES6 (în curând)

git.io/es6features

referința de bază: <https://developer.mozilla.org/JavaScript>

Limbaaj de tip *script* (interpretat)

destinat să manipuleze, să automatizeze
și să integreze funcționalitățile
oferite de un anumit sistem

Limbaaj de tip *script* (interpretat)
nu necesită intrări/ieșiri în mod implicit

Adoptă diverse paradigme de programare
imperativă
à la C

Adoptă diverse paradigme de programare

funcțională

λ calcul ► funcții anonime, închideri (*closures*),...

Adoptă diverse paradigme de programare

pseudo-obiectuală

via prototipuri

(obiectele moștenesc alte obiecte, nu clase)

Adoptă diverse paradigme de programare

dinamică

variabilele își pot schimba tipul
pe parcursul rulării programului

Cum putem executa programele JavaScript?

Mediu de execuție (*host-environment*)

navigator Web

permite rularea de aplicații Web la nivelul unei platforme
(un sistem de operare)

Mediu de execuție (*host-environment*)

navigator Web

permite rularea de aplicații Web la nivelul unei platforme
(un sistem de operare)

inclusiv pe dispozitive mobile: Android, iOS, Firefox OS,
Fire OS (Kindle Fire), Windows Phone,...

Mediu de execuție (*host-environment*)

navigator Web

„injectarea” de cod JavaScript
în documentele HTML via elementul **<script>**

Mediu de execuție (*host-environment*)

navigator Web

„injectarea” de cod JavaScript
în documentele HTML via elementul **<script>**

cod extern referit printr-un URL
vs. cod inclus direct în pagina Web

Mediu de execuție (*host-environment*)

independent de navigatorul Web

platforme de dezvoltare de aplicații distribuite: **Node.js**
servere de baze de date – *e.g.*, **Apache CouchDB**
componente ale sistemului de operare
aplicații de sine-stătătoare – *e.g.*, **Adobe Creative Suite**

caratteristici: **sintaxa**

Cuvinte rezervate:

**break else new var case finally return void catch
for switch while continue function this with default
if throw delete in try do instanceof typeof**

caracteristici: **sintaxa**

Alte cuvinte rezervate:

**abstract enum int short boolean export interface
static byte extends long super char final native
synchronized class float package throws const
goto private transient debugger implements
protected volatile double import public**

caracteristici: tipuri de date

Number

reprezentare în dublă precizie

stocare pe 64 biți – standardul IEEE 754

caracteristici: tipuri de date

String

secvențe de caractere Unicode

fiecare caracter ocupă 16 biți (UTF-16)

caracteristici: tipuri de date

Boolean

expresii ce se pot evalua ca fiind *true/false*

caracteristici: tipuri de date

Object

aproape totul e considerat ca fiind obiect,
inclusiv funcțiile

caracteristici: tipuri de date

Null

semnifică „nicio valoare”

caracteristici: **tipuri de date**

Undefined

are semnificația „nicio valoare asignată încă”

caracteristici: tipuri de date

Nu există valori întregi

caracteristici: **tipuri de date**

Nu există valori întregi

convertirea unui șir în număr: **parseInt ()**

parseInt ("123") \equiv 123

parseInt ("11", 2) \equiv 3



indică baza
de numerație

caracteristici: tipuri de date

„Valoarea” **NaN** (*“not a number”*)

parseInt ("Salut") \equiv NaN

isNaN (NaN + 3) \equiv true

caratteristici: **tipuri de date**

Valori speciale:

Infinity
-Infinity

caracteristici: **tipuri de date**

Un caracter reprezintă un șir de lungime 1

caracteristici: tipuri de date

Șirurile sunt obiecte

"Salut".length ≡ 5

caracteristici: **tipuri de date**

Metode utile
pentru procesarea șirurilor de caractere:

s.charAt(pos) **s.charCodeAt(pos)** **s.concat(s1, ..)**
s.indexOf(s1, start)
s.match(regex) **s.replace(search, replace)**
s.slice(start, end) **s.split(separator, limit)**
s.substring(start, end)
s.toLowerCase() **s.toUpperCase()**
etc.

caracteristici: tipuri de date

Valorile 0, "", NaN, null, undefined
sunt interpretate ca fiind *false*

!!234 \equiv true

caracteristici: variabile

Variabilele se declară cu **var**

```
var marime;  
var numeAnimal = "Tux";
```

variabilele declarate fără valori asignate,
se consideră undefined

caracteristici: **variabile**

Dacă nu se folosește **var**,
atunci variabila este considerată globală

de evitat așa ceva!

caratteristici: **variabile**

Valorele sunt „legate” tardiv la variabile
(*late binding*)

caracteristici: operatori

Pentru numere: $+$ $-$ $*$ $/$ $\%$

De asignare: $+=$ $-=$ $*=$ $/=$ $\%=$

Incrementare și decrementare: $++$ $--$

Concatenare de șiruri: "Java" $+$ "Script" \equiv "JavaScript"

caracteristici: operatori

Conversia tipurilor se face „din zbor” (dinamic)

"3" + 4 + 5 \equiv 345

3 + 4 + "5" \equiv 75

adăugând un șir vid la o expresie,
o convertim pe aceasta la *string*

caracteristici: operatori

Comparații: `<` `>` `<=` `>=` (numere și șiruri)

egalitatea valorilor se testează cu `==` și `!=`

`1 == true` \equiv `true`

caracteristici: operatori

Comparații: `<` `>` `<=` `>=` (numere și șiruri)

egalitatea valorilor se testează cu `==` și `!=`

`1 == true` \equiv `true`

a se folosi: `1 === true` \equiv `false`



inhibă conversia
tipurilor de date

caracteristici: operatori

Aflarea tipului unei expresii: operatorul **typeof**
typeof "Tux" \equiv string

operand	rezultat
undefined	'undefined'
null	'object'
de tip Boolean	'boolean'
de tip Number	'number'
de tip String	'string'
Function	'function'
orice alte valori	'object'

caracteristici: operatori

Operatorii logici **&&** și **||**

```
var nume = unNume || "Implicit";
```

caracteristici: operatori

Operatorul ternar de test ? :

```
var prezență = (studenți > 33) ? "Prea mulți" : "Cam puțini...";
```

caracteristici: control

Testare: **if ... else, switch**

pentru **switch**, sunt permise expresii la fiecare **case**
(testarea se realizează cu operatorul **===**)

```
switch (2 + 3) {                               /* sunt permise expresii */  
    case 4 + 1 : egalitate ();  
                break;  
    default    : absurd (); // nu se apelează niciodată  
}
```

caratteristici: control

Ciclare: **while**, **do ... while**, **for**

```
do {  
    var nume = preiaNume ();  
} while (nume != "");  
for (var contor = 0; contor < 33; contor++) {  
    // de 33 de ori...  
}
```

caracteristici: control

Excepții: **try ... catch ... finally**

```
try {  
    // Linii "periculoase" ce pot cauza excepții  
} catch (eroare) {  
    // Linii rulate la apariția unei/unor excepții  
} finally {  
    // Linii care se vor executa la final  
}
```

caracteristici: control

Excepții: **try ... catch ... finally**

```
try {  
    // Linii "periculoase" ce pot cauza excepții  
} catch (eroare) {  
    // Linii rulate la apariția unei/unor excepții  
} finally {  
    // Linii care se vor executa la final  
}
```

emiterea unei excepții: **throw**

```
throw new Error ("O eroare de-a noastră...");
```

caracteristici: control

Gestionarea erorilor

Error
EvalError
RangeError
ReferenceError
SyntaxError
TypeError
URIError

caracteristici: obiecte

Perechi nume—valoare

caracteristici: obiecte

Perechi nume—valoare

tabele de dispersie (*hash*) în C/C++
tablouri asociative în Perl, PHP sau Ruby
HashMaps în Java

“everything except primitive values is an object”

caracteristici: obiecte

Perechi nume—valoare

numele este desemnat de un șir de caractere
(*i.e.*, expresie de tip String)

valoarea poate fi de orice tip,
inclusiv null sau undefined

caracteristici: **obiecte**

Obiect \equiv colecție de *proprietăți*,
având mai multe *attribute*

caracteristici: obiecte

Obiect \equiv colecție de *proprietăți*,
având mai multe *atribute*

proprietățile pot conține alte obiecte,
valori primitive sau metode

caracteristici: **obiecte**

Obiecte predefinite:

Global	Object
Function	Array
String	RegExp
Boolean	Number
Math	Date

caracteristici: **obiecte**

Create prin intermediul operatorului **new**:

```
var ob = new Object();
```

```
var ob = { }; // echivalent cu linia anterioară
```



se preferă această sintaxă

caracteristici: obiecte

Accesarea proprietăților – operatorul .

```
ob.ume = "Tux";  
var nume = ob.ume;
```


caracteristici: obiecte

Accesarea proprietăților – operatorul .

```
ob.ume = "Tux";  
var ume = ob.ume;
```

echivalent cu:

```
ob["ume"] = "Tux";  
var ume = ob["ume"];
```

caracteristici: obiecte

Declarare + asignare:

```
var pinguin = {  
  nume: "Tux",  
  proprietati: {  
    culoare: "verde",  
    marime: 17  
  }  
}
```

caracteristici: obiecte

Declarare + asignare:

```
var pinguin = {  
  nume: "Tux",  
  proprietati: {  
    culoare: "verde",  
    marime: 17  
  }  
}
```

accesare:

```
pinguin.proprietati.marime ≡ 17  
pinguin["proprietati"]["culoare"] ≡ verde
```

```
var facultyContactInfo = {  
  // numele proprietăților sunt încadrate de ghilimele  
  "official-phone" : '+40232201090',  
  city : 'Iasi', // dacă numele e identificator valid, ghilimelele pot fi omise  
  'street' : 'Berthelot Street',  
  'number' : 16, // pot fi folosite orice tipuri de date primitive  
  "class" : "new", // cuvintele rezervate se plasează între ghilimele  
  coord : { // obiectele pot conține alte obiecte (nested objects)  
    'geo' : { 'x': 47.176591, 'y': 27.575930 }  
  },  
  age : Math.floor("21.7") // pot fi invocate metode de calcul a valorilor  
};  
  
console.log (facultyContactInfo.coord["geo"].y); // obținem 27.57593
```

caracteristici: obiecte

Iterarea proprietăților – considerate chei:

```
var pinguin = { 'nume': 'Tux', 'marime': 17 };  
for (var proprietate in pinguin) {  
    afiseaza (proprietate + ' = ' + pinguin[proprietate]);  
}
```

caracteristici: obiecte

Eliminarea proprietăților se realizează cu **delete**

```
var pinguin = { 'nume': 'Tux', 'marime': 17 };  
pinguin.nume = undefined; // nu șterge proprietatea  
delete pinguin.marime;    // eliminare efectivă  
for (var prop in pinguin) {  
    console.log (prop + "=" + pinguin[prop]);  
}  
// va apărea doar "nume=undefined"
```

caracteristici: tablouri

Tipuri speciale de obiecte

proprietățile (cheile) sunt numere,
nu șiruri de caractere

caracteristici: tablouri

Se poate utiliza sintaxa privitoare la obiecte:

```
var animale = new Array ();
```

```
animale[0] = "penguin";
```

```
animale[1] = "omida";
```

```
animale[2] = "pterodactil";
```

```
animale.length ≡ 3
```


caracteristici: tablouri

Se poate utiliza sintaxa privitoare la obiecte:

```
var animale = new Array ();
```

```
animale[0] = "penguin";
```

```
animale[1] = "omida";
```

```
animale[2] = "pterodactil";
```

```
animale.length ≡ 3
```

notație alternativă – preferată:

```
var animale = ["penguin", "omida", "pterodactil"];
```

caracteristici: tablouri

Elementele pot aparține
unor tipuri de date eterogene

```
var animale = [33, "vierme", false, "gaga"];
```

caracteristici: tablouri

Tablourile pot avea „găuri” (*sparse arrays*):

```
var animale = ["penguin", "omida", "pterodactil"];  
animale[33] = "om";
```

```
animale.length ≡ 34
```

```
typeof animale[13] ≡ undefined
```

caracteristici: tablouri

Tablourile pot avea „găuri” (*sparse arrays*):

```
var animale = ["penguin", "omida", "pterodactil"];  
animale[33] = "om";
```

```
animale.length ≡ 34
```

```
typeof animale[13] ≡ undefined
```

pentru a adăuga elemente putem recurge la:

```
animale[animale.length] = altAnimal;
```

caracteristici: tablouri – exemplu

```
var vector = [ ];  
vector[0] = "zero";  
vector[new Date().getTime()] = "now";  
vector[3.14] = "pi";  
  
for (var elem in vector) {  
    console.log ("vector[" + elem + "] = " + vector[elem] +  
        ", typeof( " + elem + ") == " + typeof (elem));  
}
```



The screenshot shows the JS Bin web application interface. At the top, there is a navigation bar with 'JS Bin' logo, 'Add library', 'Share', and tabs for 'HTML', 'CSS', 'JavaScript', 'Console', and 'Output'. On the right of the navigation bar are links for 'Log in', 'Register', and 'Help'. The main area is split into two panels. The left panel, titled 'JavaScript', contains the following code:

```
var vector = [ ];
vector[0] = "zero";
vector[new Date().getTime()] = "now";
vector[3.14] = "pi";

for (var elem in vector) {
  console.log ("vector[" + elem + "] = " +
    vector[elem] +
    ", typeof( " + elem + ") == " +
    typeof (elem));
}
```

The right panel, titled 'Console', shows the output of the code execution. It contains three lines of log messages and a blue prompt character '>'. A 'Run' button is located in the top right corner of the console panel.

```
"vector[0] = zero, typeof( 0) == string"
"vector[1350298529713] = now, typeof( 1350298529713) == string"
"vector[3.14] = pi, typeof( 3.14) == string"
>
```

rezultatul obținut în urma rulării programului JavaScript
via aplicația **JS Bin**
<http://jsbin.com/>

caracteristici: tablouri

Metode utile:

a.toString() **a.concat(item, ..)** **a.join(sep)**
a.pop() **a.push(item, ..)** **a.reverse()**
a.shift() **a.unshift([item]..)**
a.sort(cmpfn) **a.splice(start, delcount, [item]..)**
etc.

caracteristici: funcții

Definite via **function**

```
function transformaPixelInPuncte (px) {  
    var puncte = px * 300;  
    return puncte;  
}
```


caracteristici: funcții

Dacă nu este întors nimic în mod explicit,
valoarea de retur se consideră undefined

caracteristici: funcții

Parametrii de intrare pot lipsi,
fiind considerați undefined

caracteristici: funcții

Pot fi transmise mai multe argumente,
cele în surplus fiind ignorate

`transformaPixelInPuncte` (10, 7) \equiv 3000

caracteristici: funcții

Funcțiile sunt tot obiecte

astfel, pot fi specificate funcții anonime



caracteristici: funcții

Funcțiile sunt tot obiecte

astfel, pot fi specificate funcții anonime



în acest sens, JavaScript este un limbaj funcțional

caracteristici: funcții

```
var media = function () { // calculul mediei a N numere
    var suma = 0;
    for (var iter = 0,
        lung = arguments.length;
        iter < lung; iter++) {
        suma += arguments[iter];
    }
    return suma / arguments.length;
};
```

caracteristici: funcții

```
var media = function () { // calculul mediei a N numere
    var suma = 0;
    for (var iter = 0,
        lung = arguments.length;
        iter < lung; iter++) {
        suma += arguments[iter];
    }
    return suma / arguments.length;
};
```

variabilele declarate
în funcție nu vor fi
accesibile din exterior,
fiind „închise”

▼
funcție *closure*

 JS Bin

Add library

Share

HTML

CSS

JavaScript

Console

Output

Log in

Register

Help

JavaScript ▾

```
var media = function () {  
  // calculul mediei a N numere  
  var suma = 0;  
  for (var iter = 0, lung = arguments.length;  
       iter < lung; iter++) {  
    suma += arguments[iter];  
  }  
  return suma / arguments.length;  
};  
  
console.log (  
  media (9, 10, 7, 8, 7)  
);  
console.log (  
  media (3, 9, 9)  
);
```

Console

Run

8.2

7

>

console.log (media (9, 10, 7, 8, 7)) ≡ 8.2

 JS Bin Add library Share HTML CSS JavaScript Console Output Log in Register Help

JavaScript ▾

```
var media = function () {  
  // calculul mediei a N numere  
  var suma = 0;  
  for (var iter = 0, lung = arguments.length;  
    iter < lung; iter++) {  
    suma += arguments[iter];  
  }  
  return suma / arguments.length;  
};  
  
console.log (  
  media (9, 10, 7, 8, 7)  
);  
console.log (  
  media (3, 9, 9)  
);
```

Console

Run

8.2

7

>

precizați ce efect vor avea liniile de cod următoare:

console.log (typeof (media));

console.log (media());

<p>JavaScript ▾</p> <pre>var media = function () { // calculul mediei a N numere var suma = 0; for (var iter = 0, lung = arguments.length; iter < lung; iter++) { suma += arguments[iter]; } return suma / arguments.length; }; console.log (typeof (media)); console.log (media());</pre>	<p>Console Run</p> <pre>"function" NaN ></pre>
--	--

variabila **media** este de tip **function**
apelul **media()** întoarce valoarea **NaN**

caracteristici: de la funcții la clase

Exemplificare:
dorim să procesăm – via funcții –
caracteristici ale unor animale

caracteristici: de la funcții la clase

```
function creeazaAnimal (nume, marime) {  
    return { nume: nume, marime: marime }  
}  
function oferaNume (animal) {  
    return animal.nume;  
}  
function oferaMarime (animal) {  
    return animal.marime;  
}
```

caracteristici: de la funcții la clase

```
function creeazaAnimal (nume, marime) {  
    return { nume: nume, marime: marime };  
}  
function oferaNume (animal) {  
    return animal.nume;  
}  
function oferaMarime (animal) {  
    return animal.marime;  
}
```

```
var tux = creeazaAnimal ("Tux", 17);  
        oferaMarime (tux) ≡ 17
```

caracteristici: de la funcții la clase

O „clasă” referitoare la animale:

```
function creeazaAnimal (nume, marime) {  
  return {  
    nume: nume,           // date-membre  
    marime: marime,  
    oferaNume: function () { // funcție de furnizare a numelui  
      return animal.nume;  
    },  
    oferaMarime: function () { // o altă „metodă”  
      return animal.marime;  
    }  
  };  
}
```

caracteristici: de la funcții la clase

Dorim să apelăm funcțiile definite:


```
var tux = creeazaAnimal ("Tux", 17);
```

```
tux.oferaMarime ()
```



ReferenceError: animal is not defined

caracteristici: de la funcții la clase



clase – utilizarea
constructorilor

```
function Animal (nume, marime) {  
    this.nume = nume;           // date-membre  
    this.marime = marime;  
    this.oferaNume = function () { // metodă  
        return this.nume;  
    };  
    this.oferaMarime = function () { // metodă  
        return this.marime;  
    };  
}
```

creational pattern

caracteristici: funcții & obiecte

Operatorul **new** creează un nou obiect vid și apelează funcția specificată cu **this** setat pe acest obiect

aceste funcții se numesc *constructori*,
trebuie apelate via **new**
și, prin convenție, au numele scris cu literă mare

caracteristici: de la funcții la clase

Față de alte limbaje de programare,
obiectul curent – referit cu `this` –
este setat ca fiind obiectul global

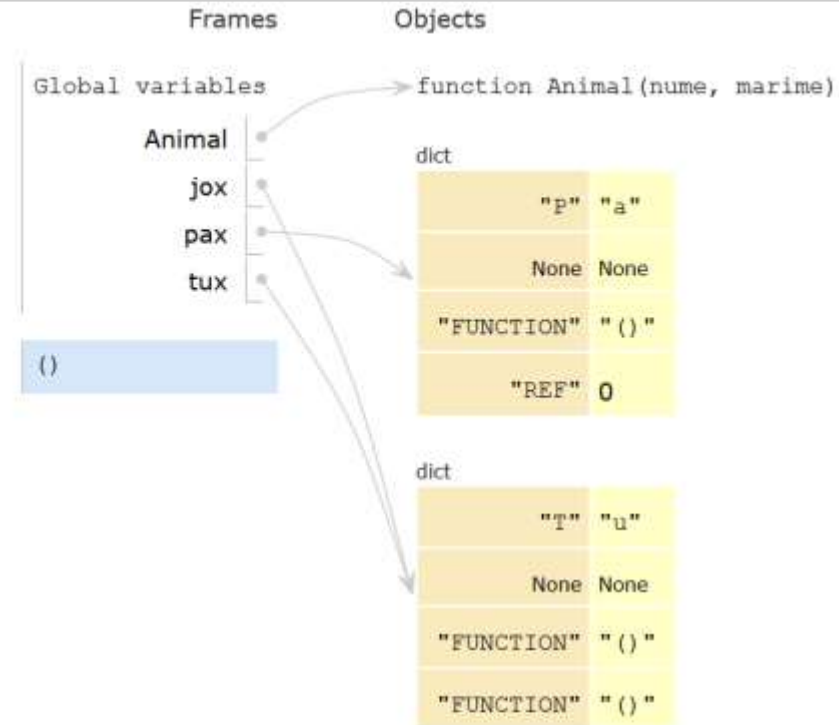
de exemplu, în *browser*, reprezintă fereastra curentă
în care este redat documentul: `this` \equiv `window`

```
1 function Animal (nume, marime) {
2     this.nume = nume; // date-membre
3     this.marime = marime;
4     this.oferaNume = function () { // metodă
5         return this.nume;
6     };
7     this.oferaMarime = function () { // metodă
8         return this.marime;
9     };
10 }
11 var tux = new Animal ("Tux", 17); // instanțiere
12 console.log (tux.oferaNume());
13 var jox = tux;
14 console.log (jox.oferaNume());
15 var pax = new Animal ("Pax", 15); // alt obiect
16 pax.marime = 21;
17 console.log (pax.oferaMarime());
```

[Edit code](#)

Program output:

```
Tux
Tux
```



```
// instanțierea unui obiect
var tux = new Animal ("Tux", 17);
console.log (tux.oferaNume ());
var jox = tux;
console.log (jox.oferaNume ());
// alt obiect
var pax = new Animal ("Pax", 15);
pax.marime = 21;
console.log (pax.oferaMarime ());
```

trasarea pas-cu-pas a execuției programului, cu inspectarea valorilor variabilelor, via www.pythontutor.com/visualize.html#py=js

caracteristici: funcții & obiecte

Metodele pot fi declarate și în exteriorul constructorului

```
function oferaNumeAnimal () {  
    return this.numa;  
}  
function Animal (numa, marime) {  
    this.numa = numa;  
    this.marime = marime;  
    this.oferaNume = oferaNumeAnimal; // referă funcția de mai sus  
}
```

caracteristici: proprietăți

Orice obiect deține trei tipuri de proprietăți:

named data property

o proprietate având asignată o valoare

named accessor property

de tip *setter/getter* pentru a stabili/accesa o valoare

internal property

folosită exclusiv de procesorul ECMAScript (JavaScript)

caracteristici: proprietăți

Fiecare proprietate are asociate attributele:

[[Value]] – desemnează valoarea curentă a proprietății

[[Writable]] – indică dacă o proprietate
poate să-și modifice valoarea

[[Get]] și **[[Set]]** – funcții opționale pentru a oferi/stabili
valoarea unei proprietăți de tip *accessor*

[[Enumerable]] – specifică dacă numele proprietății
va fi disponibil într-o buclă **for-in**

[[Configurable]] – indică dacă proprietatea
poate fi ștearsă ori redefinită

[[Prototype]]	definește ierarhiilor de obiecte
[[Get]] [[Put]] [[CanPut]]	pentru accesarea valorilor
[[HasProperty]] [[DefineOwnProperty]] [[GetProperty]] [[GetOwnProperty]] [[Delete]]	manipularea proprietăților
[[Extensible]]	indică obiectele ce pot fi extinse
[[Construct]] [[Call]]	asociate obiectelor executabile (funcții)
[[Code]] [[Scope]]	desemnează codul & contextul unei obiect de tip funcție

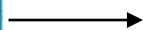
proprietăți interne importante (folosite de procesorul
ECMAScript, dar inaccesibile la nivel de program)

caracteristici: proprietăți

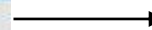
```
// crearea unei proprietăți simple stocând date  
// (writable, enumerable, configurabile)  
obiect.numeProprietate = 33;
```

```
// crearea via API-ul intern a unei proprietăți stocând date  
Object.defineProperty (obiect, "numeProprietate", {  
  value: 33, writable: true, enumerable: true, configurable: true }  
)
```


this
Object.prototype



Object.prototype
constructor: Object()
toSource: toSource()
toString: toString()
toLocaleString: toLocaleString()
valueOf: valueOf()
watch: watch()
unwatch: unwatch()
hasOwnProperty: hasOwnProperty()
isPrototypeOf: isPrototypeOf()
propertyIsEnumerable: propertyIsEnumerable()
__defineGetter__: __defineGetter__()
__defineSetter__: __defineSetter__()
__lookupGetter__: __lookupGetter__()
__lookupSetter__: __lookupSetter__()
null



Object()
prototype: Object.prototype
getPrototypeOf: getPrototypeOf()
getOwnPropertyDescriptor: getOwnPropertyDescriptor()
keys: keys()
is: is()
defineProperty: defineProperty()
defineProperties: defineProperties()
create: create()
getOwnPropertyNames: getOwnPropertyNames()
isExtensible: isExtensible()
preventExtensions: preventExtensions()
freeze: freeze()
isFrozen: isFrozen()
seal: seal()
isSealed: isSealed()
name: "Object"
length: 1
Function.prototype

proprietățile interne ale obiectelor definite

<http://www.objectplayground.com/>

caracteristici: prototipuri

Deoarece orice obiect deține în mod implicit
proprietatea **prototype**,
structura unei clase poate fi extinsă ulterior

caracteristici: prototipuri

Deoarece orice obiect deține în mod implicit
proprietatea **prototype**,
structura unei clase poate fi extinsă ulterior

un prototip e o proprietate oferind o legătură ascunsă
către obiectul de care aparține

caracteristici: prototipuri

Dacă se încearcă accesarea unui element inexistent
în cadrul unui obiect dat,
se va verifica lanțul de prototipuri (*prototype chain*)

```
function Animal (nume, marime) { // definiție inițială  
    this.nume = nume;  
    this.marime = marime;  
}
```

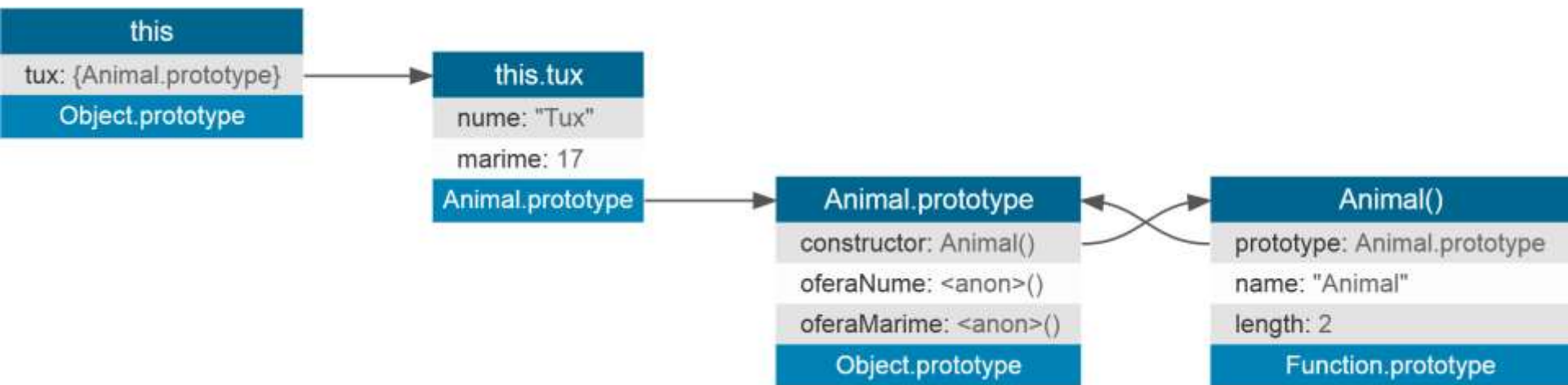
// pe baza protipurilor, definim noi metode

```
Animal.prototype.oferaNume = function () {  
    return this.nume;  
};
```

```
Animal.prototype.oferaMarime = function () {  
    return this.marime;  
};
```

// instanțiem un obiect de tip Animal

```
var tux = new Animal ("Tux", 17);
```



caracteristici: prototipuri

Pentru a cunoaște tipul unui obiect
(pe baza constructorului și a ierarhiei de prototipuri)
se folosește operatorul **instanceof**

caratteristiche: prototipuri

```
var marimi = [17, 20, 7, 14];
```

marimi instanceof Array	≡	true
marimi instanceof Object	≡	true
marimi instanceof String	≡	false

```
var tux = new Animal ("Tux", 17);
```

tux instanceof Object	≡	true
tux instanceof Array	≡	false

caracteristici: extinderea claselor

Adăugarea unei metode se realizează via **prototype**

```
Animal.prototype.oferaNumeMare = function () {  
    return this.num.toUpperCase ();  
};
```

```
tux.oferaNumeMare () ≡ "TUX"
```

caracteristici: extinderea claselor

Pot fi extinse și obiectele predefinite:

```
// adăugăm o metodă obiectului String
String.prototype.inverseaza = function () {
    var inv = "";
    for (var iter = this.length - 1; iter >= 0; iter--) { // inversăm șirul...
        inv += this[iter];
    }
    return inv;
};
"Web".inverseaza () ≡ "beW"
```

caracteristici: extinderea claselor

Cel mai general prototype este cel al lui **Object**

una dintre metodele disponibile – predefinite – este **toString()** care poate fi suprascrisă (*over-ride*)

caracteristici: extinderea claselor

```
var tux = new Animal ("Tux", 17);
```

tux.toString () ≡ [object Object]

```
Animal.prototype.toString = function () {  
    return '<animal>' + this.oferaNume () + '</animal>';  
};
```

tux.toString () ≡ "<animal>Tux</animal>"

Atenție la pericole! – *e.g.*, comportamentul diferit al construcției for (var *proprietate* in *obiect*)

caracteristici: extinderea claselor

Şablon general:

```
ClasaDeBaza = function () { /* ... */ };  
SubClasa = function () { /* ... */ };  
SubClasa.prototype = new ClasaDeBaza ();
```

caracteristici: funcții de nivel înalt

Deoarece o funcție reprezintă un obiect, poate fi:

- stocată ca valoare (asociată unei variabile)

- pasată ca argument al unei alte funcții

- întoarsă de o funcție – fiind argument pentru return

caracteristici: încapsulare

JavaScript oferă un singur spațiu de nume,
la nivel global

- ▶ conflicte privind denumirea funcțiilor/variabilelor
specificate de programe diferite,
concepute de mai multi dezvoltatori

caracteristici: încapsulare

Nu trebuie afectat spațiul de nume global,
păstrându-se codul-sursă la nivel privat

caracteristici: încapsulare

Codul poate fi complet încapsulat
via funcții anonime
care „păstrează” construcțiile la nivel privat

caracteristici: *closures*

Declararea imbricată – ca expresii de tip funcție – a funcțiilor anonime are denumirea *closures*



închideri

<https://developer.mozilla.org/en/JavaScript/Guide/Closures>

caracteristici: *closures*

```
// specificarea unei expresii de tip funcție  
( function () {  
    // variabilele & funcțiile vor fi vizibile doar aici  
    // variabilele globale pot fi accesate  
} ( ) );
```

```
var cod = (function () {  
    var priv = 0; // variabilă privată  
    function start (x) {  
        // ... poate accesa 'priv'  
        // și funcția 'faCeva'  
    }  
    function faAia (param) {  
        // ... invizibilă din afară  
    }  
})
```

```
function faCeva (x, y) {  
    // ...  
}  
return {  
    // sunt publice doar  
    // funcțiile 'start' și 'faCeva'  
    'start': start,  
    'faCeva': faCeva  
}  
})();  
  
cod.start (arg); // apelăm 'start'
```

```
var cod = (function () {  
    var priv = 0; // variabilă privată  
    function start (x) {  
        // ... poate accesa 'priv'  
        // și funcția 'faCeva'  
    }  
    function faAia (param) {  
        // ... invizibilă din afară  
    }  
})
```

```
function faCeva (x, y) {  
    // ...  
}  
return {  
    // sunt publice doar  
    // funcțiile 'start' și 'faCeva'  
    'start': start,  
    'faCeva': faCeva  
}  
})();  
  
cod.start (arg); // apelăm 'start'
```

- via *closures*, simulăm metodele private
- modularizarea codului (*module pattern*)

```

var makeCounter = function () {
  var contorPrivat = 0;           // un contor de valori (inițial, zero)
  function changeBy (val) {      // funcție privată
    contorPrivat += val;          // ce modifică valoarea contorului
  }
  return {                        // funcții publice (expuse)
    increment: function() {
      changeBy (1);
    },
    decrement: function() {
      changeBy (-1);
    },
    value: function() {
      return contorPrivat;
    }
  };
};

```

```

console.log (contor1.value ()); /* 0 */
contor1.increment ();
contor1.increment ();
console.log (contor1.value ()); /* 2 */
contor1.decrement ();
console.log (contor1.value ()); /* 1 */
console.log (contor2.value ()); /* 0 */

```

JavaScript ▾

Console

```
var makeCounter = function () {
    var contorPrivat = 0;    // un contor de valori (initial, zero)
    function changeBy (val) { // functie privata
        contorPrivat += val;    // ce modifica valoarea contorului
    }
    return {                // functii publice (expuse)
        increment: function() {
            changeBy (1);
        },
        decrement: function() {
            changeBy (-1);
        },
        value: function() {
            return contorPrivat;
        }
    };
};

// cream 2 contoare
var contor1 = makeCounter ();
var contor2 = makeCounter ();

console.log (contor1.value ()); /* 0 */
contor1.increment ();
contor1.increment ();
console.log (contor1.value ()); /* 2 */
contor1.decrement ();
console.log (contor1.value ()); /* 1 */
console.log (contor2.value ()); /* 0 */
```

0

2

1

0



de reținut!

Totul în JavaScript este obiect – chiar și funcțiile

de reținut!

Orice obiect este întotdeauna mutabil
(poate fi alterat oricând)

proprietățile și metodele sunt disponibile
oriunde în program (*public scope*)

de reținut!

Nu există vizibilitate la nivel de bloc de cod
(*block scope*),
ci doar la nivel global ori la nivel de funcție

de reținut!

Funcțiile ascund orice e definit în interiorul lor

de reținut!

Operatorul “.” este echivalent
cu de-referențierea:

`ob.prop === ob["prop"]`

de reținut!

Operatorul **new** creează obiecte aparținând „clasei” specificate de funcția constructor

de reținut!

Accesorul **this** este relativ la contextul execuției,
nu al declarării



Atenție la dependența
de mediul de execuție!

de reținut!

Proprietatea **prototype** are valori modificabile

json

JavaScript Object Notation

<http://json.org/>

json

JavaScript Object Notation

format compact pentru interschimb de date
între aplicații

biblioteci de procesare și alte resurse de interes:
<http://jsonauts.github.io/>

json

JavaScript Object Notation

datele pot fi specificate
în termeni de obiecte și literali

json

JavaScript Object Notation

```
{  
  'nume': 'Tux',  
  'stoc': 33,  
  'model': [ 'candid', 'furios', 'vesel' ]  
}
```



datele pot fi evaluate
direct în JavaScript

instrumente

procesare strictă a limbajului: **"use strict";**

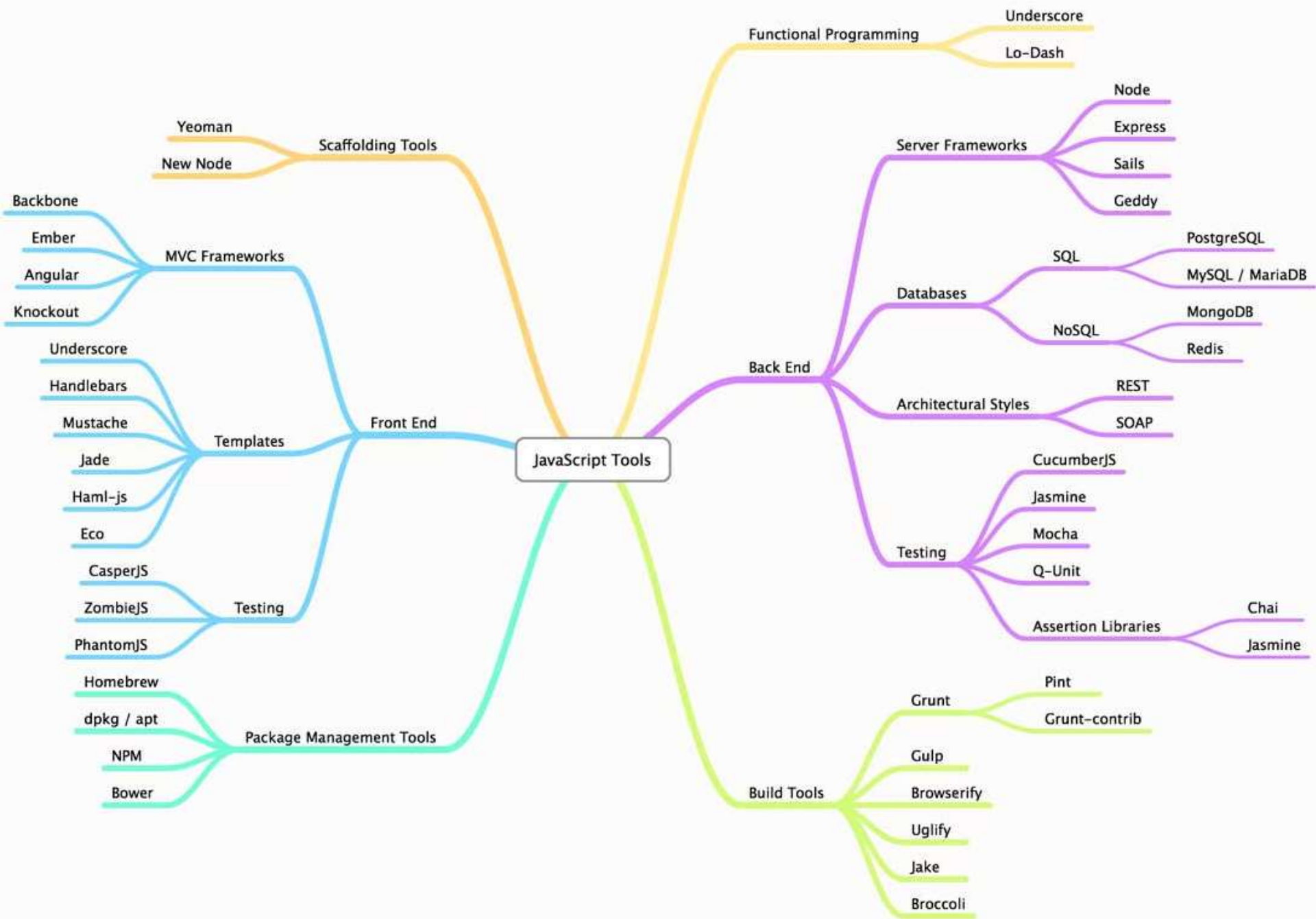
verificare statică a codului: **JSLint, JSHint, ESLint**

suport pentru *unit testing*: **Jasmine, Qunit, Sinon.js, Tyrtle**

documentarea codului: **JSDoc Toolkit**

optimizare de cod: **Closure Compiler**

de studiat și S. Buraga, „Ingineria dezvoltării aplicațiilor JavaScript”
www.slideshare.net/busaco/cliw-20142015-12-ingineria-dezvoltrii-aplicaiilor-javascript



Care este viitorul limbajului JavaScript?

EcmaScript 6 (ES6)

specificație în stadiu de ciornă (Rev 36 – 17 martie 2015)

wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

<https://people.mozilla.org/~jorendorff/es6-draft.html>

instrumente ES6: <https://github.com/addyosmani/es6-tools>

EcmaScript 6 (ES6)

definirea de clase – perspectiva paradigmei obiectuale
parametri cu valori implicite și parametri multipli
iteratori și generatori

noi tipuri de date – *e.g., map, set, proxy*
modularizarea codului: module + importuri

...

de studiat și *Essential JavaScript Links*

<https://gist.github.com/ericelliott/d576f72441fc1b27dace>

N-am putea dezvolta în JavaScript
aplicații Web la nivel de server?

node.js: caratterizzare

*“Node.js is a platform built on Chrome’s JavaScript runtime for **easily building fast, scalable network applications**.*

*Node.js uses an **event-driven, non-blocking I/O** model that makes it lightweight and efficient, perfect for **data-intensive real-time applications** that run across distributed devices.”*

Ryan Dahl concepe **Node.js** (2009) – <http://nodejs.org/>

Node.js rulează pe mașini respectând standardul POSIX
+ pe calculatoare Windows (2011)

Node.js este adoptat de industrie (2012)
e.g., Cloud9 IDE, eBay, LinkedIn, Storify, Yahoo!

Node.js Foundation (februarie 2015)

Joyent, IBM, Microsoft, PayPal, Fidelity, SAP, The Linux Foundation

Node.js 0.12 – versiunea actuală (martie 2015)

io.js – varianta Node.js concepută în ES6 (în lucru)

node.js

Permite dezvoltarea de **aplicații Web**
la nivel de server în limbajul JavaScript

recurge la **V8** – procesor (interpretor) JavaScript
creat de Google și disponibil liber

<https://code.google.com/p/v8/>

node.js: caracterizare

Oferă suport pentru cele mai importante
protocoale Web și Internet

HTTP (*HyperText Transfer Protocol*)

DNS (*Domain Name System*)

TLS (*Transport Layer Security*)

funcționalități de nivel scăzut (*socket-uri TCP*)

node.js: caracterizare

Operațiile de intrare/ieșire sunt **asincrone**

fiecare cerere (operație) adresată aplicației
– *e.g.*, acces la disc, la rețea, la alt proces – poate avea atașată o funcție de tratare a unui eveniment specific

evented I/O

**cod JS executat de
client (*browser Web*)**

așteaptă și tratează
evenimente de
interacțiune
(onclick, onmouseover,
onkeypressed,...)

programul trebuie să fie
responsiv atunci când
așteaptă încărcarea
datelor de pe rețea
(*e.g.*, JSON, XML, imagini,
video) via Ajax/Comet
ori *socket*-uri Web

**procesare
bazată pe
evenimente**
*evented/
event-based*

asincronism
(*e.g.*, operatii
neblocante)

**cod JS rulat pe partea
de server (node.js)**

așteaptă și tratează
cereri (evenimente)
provenite de la client(i)

programul trebuie să fie
responsiv atunci când
așteaptă încărcarea
datelor locale/externe
(preluate din baze de
date, fișiere,
servicii Web, API-uri,...)

node.js: caracterizare

O aplicație node.js rulează într-**un singur proces**

codul JavaScript nu este executat paralel,
dar tratează în mod asincron diverse evenimente I/O

single-process event loop

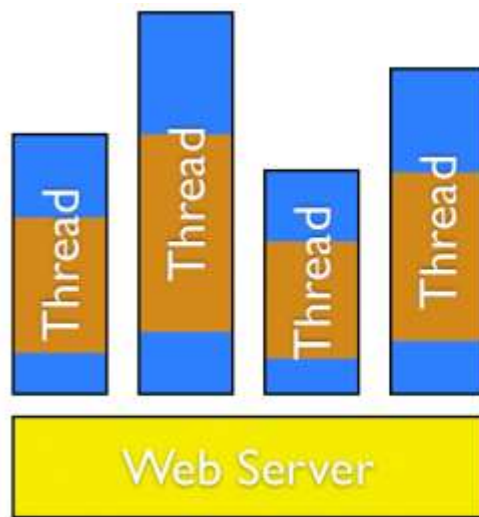
node.js: caracterizare

O aplicație node.js rulează într-**un singur proces**

deosebire esențială față de serverele de aplicații Web tradiționale ce recurg la servere *multi-process/threaded*

node.js: caracterizare

O aplicație node.js rulează într-**un singur proces**

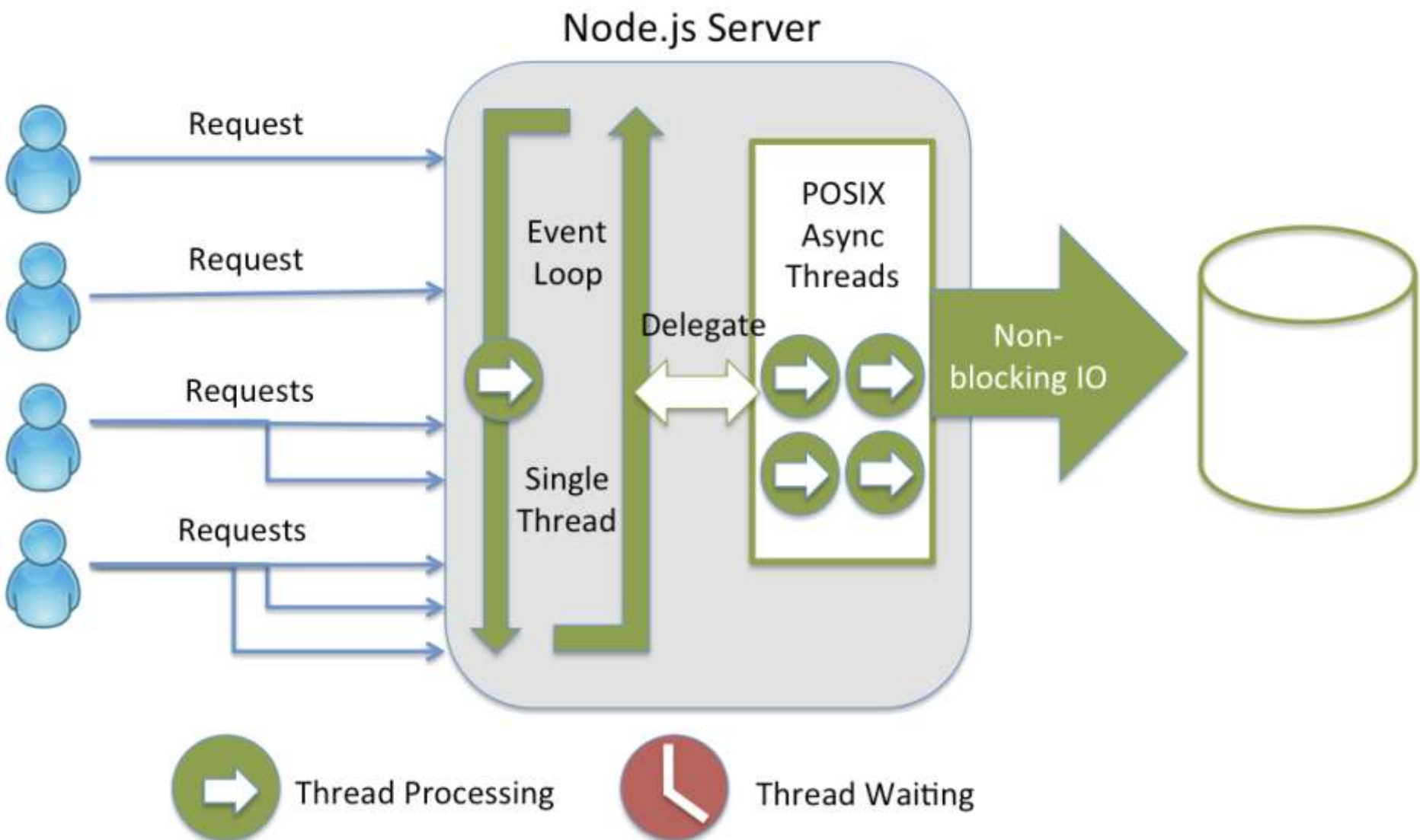


server tipic



server node.js

adaptare după Ben Sheldon (2012): <http://www.island94.org/?p=3066>



<http://strongloop.com/strongblog/node-js-is-faster-than-java/>

node.js: caracterizare

Mediul node.js e disponibil gratuit – *open source* – pentru platformele UNIX/Linux, Windows, Mac OS X

<http://nodejs.org/download/>



find packages



sign up or log in



npm is the package manager for **javascript**.



132,966

total packages



33,559,621

downloads in the last day



203,826,015

downloads in the last week



1,022,396,159

downloads in the last month

funcționalități suplimentare oferite
de module administrate via **npm**

<https://npmjs.org/>



browserify

browser-side require() the node way

9.0.3 published 24 days ago by substack

express

express

Fast, unopinionated, minimalist web fr...

4.12.2 published 14 days ago by dougwilson



pm2

Production process manager for Node....

0.12.7 published 15 days ago by jshkurti



grunt-cli

The grunt command line interface.



npm

a package manager for JavaScript



karma

Spectacular Test Runner for JavaScript.

node.js: caracterizare

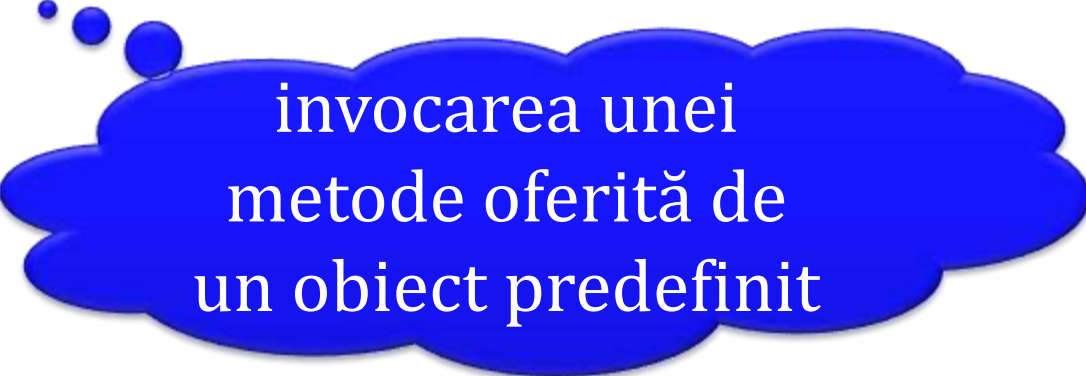
Oferă un mediu de execuție în linia de comandă, pe baza unor biblioteci C++ și a procesorului V8

`node program.js`

node.js: exemplu

Un prim program care emite mesaje de salut

```
// salutare.js: un program (de sine-stătător) care emite un salut  
console.log ('Salutari banale din Node.js');
```



invocarea unei
metode oferită de
un obiect predefinit

node.js: exemplu

Un prim program care emite mesaje de salut

```
// salutari.js: un program (de sine-stătător) care emite un salut  
console.log('Salutari banale din Node.js');
```

```
> node salutari.js  
Salutari banale din Node.js
```



```
// Un program JavaScript care salută toți posibili săi clienți Web  
var http = require ('http');      // folosim 'http', un modul Node predefinit
```

```
http.createServer (                // creăm un server Web  
// funcție anonimă ce tratează o cerere și trimite un răspuns  
function (cerere, raspuns) {  
    // afișăm la consola serverului mesaje de diagnostic  
    console.log ('Am primit o cerere...');  
    // stabilim valori pentru diverse câmpuri din antetul mesajului HTTP  
    raspuns.writeHead (200, { 'Content-Type': 'text/html' });  
    // emitem răspunsul propriu-zis conform tipului MIME (cod HTML)  
    raspuns.end ('<html><body><h1>Salutari...</h1></body></html>');  
}  
// serverul ascultă cereri la portul 8080 al mașinii locale  
) .listen (8080, "127.0.0.1");
```

```
console.log ('Serverul creat asteapta cereri la http://127.0.0.1:8080/');
```

programul JavaScript creat funcționează ca un server Web pentru fiecare cerere emisă de un posibil client (*browser*, aplicație *desktop* etc.) conform modelului client/server

pe partea de server – așteptare de cereri

> node **salutari-web.js**

Serverul creat asteapta cereri la http://127.0.0.1:8080/

Am primit o cerere...

Am primit o cerere...

programul JavaScript creat funcționează ca un server Web pentru fiecare cerere emisă de un posibil client (*browser*, aplicație *desktop* etc.) conform modelului client/server

pe partea de server – așteptare de cereri

> node **salutari-web.js**

Serverul creat asteapta cereri la http://127.0.0.1:8080/

Am primit o cerere...

Am primit o cerere...

la client – recepționarea răspunsului conform cererii GET emise de un program *desktop* și de un navigator Web

> node **client-salutari.js**

Am primit raspuns de la server -- cod HTTP: 200

Continut receptionat: <html><body>

<h1>Salutari din Node.js</h1></body></html>



```
// Un program JS care implementează un client pentru serviciul de salut
var http = require ('http');

http.get ('http://127.0.0.1:8080/',           // emite o cerere HTTP
function (raspuns) {
    console.log ('Am primit raspuns de la server -- cod HTTP: '
        + raspuns.statusCode);               // statusCode: 200, 404,...
})
// tratăm diverse evenimente
.on ('error',                                // eroare
function (e) { console.log ('Eroare: ' + e.message); })
.on ('response',                             // receptare răspuns de la server
function (raspuns) {                         // există date de procesat
    raspuns.on ('data', function (date) {
        console.log ('Continut receptionat: ' + date);
    });
}
);
```

node.js: module

Funcția **require ()** specifică utilizarea unui modul Node.js

module predefinite (*built-in*):

privitoare la tehnologii Web – **http, https, url, querystring**

referitoare la fișiere – **fs, path**

vizând rețeaua – **net, dns, dgram, tls,...**

resurse privind sistemul de operare – **os, child_process**

alte aspecte de interes – **buffer, console, util, crypto**

node.js: module – http

Dezvoltarea de aplicații Web via modulul **http**

funcționalități HTTP de bază

crearea unui server Web: **createServer()**

realizarea de cereri HTTP: **request()** **get()**

node.js: module – http

Dezvoltarea de aplicații Web via modulul **http**

servire de cereri HTTP – clasa **http.Server**

metode uzuale:

listen() setTimeout() close()

evenimente ce pot fi tratate:

request connect close clientError etc.

node.js: module – http

Dezvoltarea de aplicații Web via modulul **http**
răspuns emis de server – clasa **http.ServerResponse**

metode uzuale:

writeHead() **getHeader()** **removeHeader()** **write()** **end()** etc.

evenimente: **close** **clientError**

proprietăți folosite: **statusCode** **headersSent**

node.js: module – http

Dezvoltarea de aplicații Web via modulul **http**
cerere emisă de client – clasa **http.ClientRequest**

metode uzuale:

write() abort() end() setTimeout() setSocketKeepAlive()

evenimente ce pot fi tratate:

response connect continue socket etc.

node.js: module – http

Dezvoltarea de aplicații Web via modulul **http**

mesaj vehiculat – clasa **http.IncomingMessage**

metode: **setEncoding() setTimeout() pause() resume()**

evenimente ce pot fi tratate: **data end close**

proprietăți de interes:

httpVersion headers method url statusCode socket

node.js: module – url

Procesarea adreselor Web via modulul **url**

metode oferite:

parse() format() resolve()

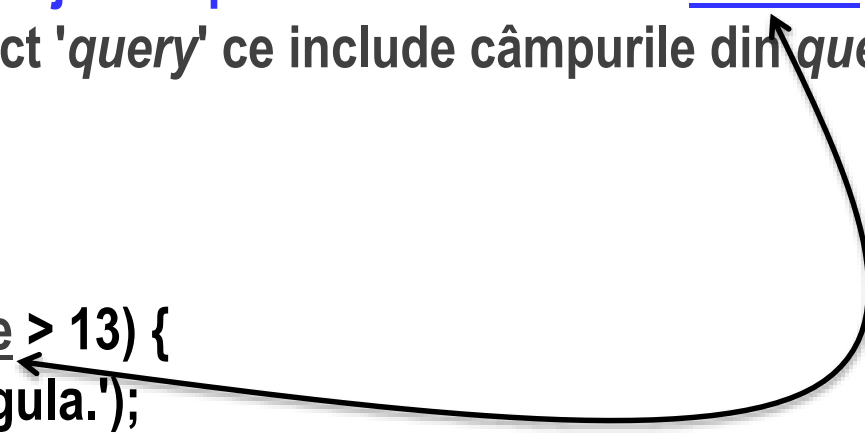
<http://nodejs.org/api/url.html>

```
var url = require ('url');

var adresaWeb = url.parse (
  'http://undeva.info:8080/oferta/jucarii/produs/?nume=Tux&marime=17#descriere',
  true    // generează un obiect 'query' ce include câmpurile din querystring
);

console.log (adresaWeb);

if (adresaWeb['query'].marime > 13) {
  console.log ('Jucaria e in regula. ');
} else {
  console.log ('Jucaria e stricata. ');
}
```



> node url.js

```
{ protocol: 'http:',  
  slashes: true,  
  auth: null,  
  host: 'undeva.info:8080',  
  port: '8080',  
  hostname: 'undeva.info',  
  hash: '#descriere',  
  search: '?nume=Tux&marime=17',  
  query: { nume: 'Tux', marime: '17' },  
  pathname: '/oferta/jucarii/produs/',  
  path: '/oferta/jucarii/produs/?nume=Tux&marime=17',  
  href: 'http://undeva.info:8080/oferta/jucarii/produs/?nume=...'  
}
```

Jucaria e in regula.

node.js: module – net

Crearea de aplicații Internet – modulul **net**

partea de server:

createServer()

+

clasa **net.Server**

metode: **listen() close() address() getConnections()**

evenimente: **listening connection close error**

node.js: module – net

Crearea de aplicații Internet – modulul **net**

partea de client:

connect()
createConnection()

node.js: module – net

Crearea de aplicații Internet – modulul **net**

acces la *socket*-uri – clasa **net.Socket**

metode: **connect() write() setEncoding() destroy() end()** etc.

evenimente: **connect data end timeout drain error close**

proprietăți utile: **localAddress localPort**

remoteAddress remotePort bytesRead bytesWritten bufferSize

node.js: module – fs

Acces la sistemul de fişiere via modulul **fs**

metode folosite uzual – comportament asincron:

open() read() write() close()

truncate() stat() chmod() rename() exists()

isFile() isDirectory() isSocket()

mkdir() rmdir() readdir()

node.js: module – fs

Acces la sistemul de fișiere via modulul **fs**

studiu de caz (Victor Porof, 2013):
generare de liste de melodii via iTunes și Last.fm

<https://github.com/victorporof/plgen>

(ilustrează și maniera de creare a modulelor proprii)

node.js: fluxuri de date

Accesul la date poate fi realizat
prin intermediul fluxurilor (*streams*)

abstractizează accesul la date stocate parțial
(*partially buffered data*)

se emit evenimente ce pot fi tratate de codul aplicației

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi citite (*readable streams*)

e.g., create de **fs.createReadStream()** **http.ServerRequest**
http.ClientResponse **net.Socket** **child.stdout** **process.stdin**

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi citite (*readable streams*)
e.g., create de **fs.createReadStream()** **http.ServerRequest**
http.ClientResponse **net.Socket** **child.stdout** **process.stdin**

emit evenimentele **data end error**

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi citite (*readable streams*)
e.g., create de **fs.createReadStream()** **http.ServerRequest**
http.ClientResponse **net.Socket** **child.stdout** **process.stdin**

au asociate metodele **pause()** **resume()** **destroy()**

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi scrise (*writeable streams*)
e.g., create de **fs.createWriteStream()** **http.ServerResponse**
http.ClientRequest **net.Socket** **child.stdin** **process.stdout**

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi scrise (*writeable streams*)
e.g., create de **fs.createWriteStream()** **http.ServerResponse**
http.ClientRequest **net.Socket** **child.stdin** **process.stdout**

emit evenimentele **drain error**

node.js: fluxuri de date

Accesul la date poate fi realizat prin intermediul fluxurilor (***streams***)

fluxuri ce pot fi scrise (*writeable streams*)
e.g., create de **fs.createWriteStream()** **http.ServerResponse**
http.ClientRequest **net.Socket** **child.stdin** **process.stdout**

oferă metodele **write()** **end()** **destroy()**

```
// Program ce preia ceea ce tastează utilizatorul la intrarea standard
// și scrie într-un fișier – conform M. Takada (2012)
var fs = require ('fs');

var fisier = fs.createWriteStream ('./spion.txt');

// la apariția datelor, le scriem în fișier
process.stdin.on ('data', function (date) { fisier.write (date); } );

// tratăm evenimentul de terminare a fluxului
process.stdin.on ('end', function() { fisier.end (); } );

// "reactivăm" intrarea standard; implicit, e în starea 'paused'
process.stdin.resume ();
```

obiectul **process** e global – detalii la <http://nodejs.org/api/process.html>

node.js: rulare temporizată

Se poate planifica execuția codului JavaScript

recurgerea la funcțiile globale

setTimeout ()
clearTimeout ()
setInterval ()
clearInterval()

```
// creăm un server Web care trimite fiecărui client secvența valorilor unui contor
var server = http.createServer().listen(8080, '127.0.0.1');

// stabilim un comportament la apariția evenimentului 'request' (cerere de la un client)
server.on('request', function(cerere, raspuns) {
  console.log('Cerere de la clientul ' + cerere.headers['user-agent']);
  raspuns.writeHead(200, { 'Content-Type': 'text/html' });

  var contor = 0;
  var interval = setInterval( // generăm valori ale contorului conform intervalului de timp
    function() {
      raspuns.write('<p>Contorul are valoarea ' + contor + '</p>');
      console.log('Contorul are valoarea ' + contor);
      contor++;
      if (contor >= 7) {
        clearInterval(interval); // ștergem intervalul
        raspuns.end();           // închidem fluxul de răspuns
        console.log('Am trimis raspuns clientului ' + cerere.headers['user-agent']);
      }
    }, 1000); // cod rulat la interval de 1000 milisec.

});
```

Cerere de la clientul ... Gecko/20100101 Firefox/36.0

Contorul are valoarea 0

Contorul are valoarea 1

Contorul are valoarea 2

Contorul are valoarea 3

Cerere de la clientul ... Windows NT 6.3; WOW64; Trident/7.0...

Contorul are valoarea 4

Contorul are valoarea 0

Contorul are valoarea 5

Contorul are valoarea 1

Contorul are valoarea 6

Am trimis raspuns clientului ... Gecko/20100101 Firefox/36.0

Contorul are valoarea 2

Contorul are valoarea 3

Contorul are valoarea 4

Contorul are valoarea 5

Contorul are valoarea 6

Am trimis raspuns clientului ... Windows NT 6.3; WOW64; Trident/7.0...



**codul este rulat
asincron**

Cerere de la clientul ... Gecko/20100101 Firefox/36.0

Contorul are valoarea 0

Contorul are valoarea 1

Contorul are valoarea 2

Contorul are valoarea 3

Cerere de la clientul ... Windows NT 6.3; WOW64; Trident/7.0...

Contorul are valoarea 4

Contorul are valoarea 0

Contorul are valoarea 5

Contorul are valoarea 1

Contorul are valoarea 6

Am trimis raspuns clientului ... Gecko/20100101 Firefox/36.0

Contorul are valoarea 2

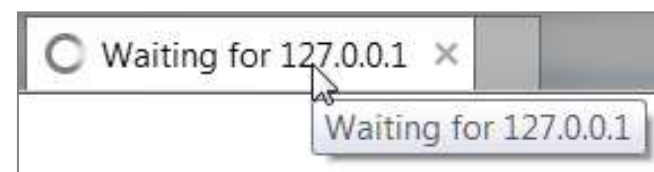
Contorul are valoarea 3

Contorul are valoarea 4

Contorul are valoarea 5

Contorul are valoarea 6

Am trimis raspuns clientului ... Windows NT 6.3; WOW64; Trident/7.0...



*browser-ul va aștepta ca întreaga secvență de valori
de ce? să fie trimisă de către server*

node.js: evenimente

Emiterea (lansarea) și tratarea (prinderea) evenimentelor specificate de programator se realizează via **event.EventEmitter**

clasă utilizată intern de multe biblioteci de bază

<http://nodejs.org/api/events.html>

node.js: module externe

Funcția **require ()** specifică utilizarea unui modul Node.js

module disponibile *on-line* (instalate via utilitarul **npm**)

instalare globală a unui modul: **npm install modul -g**

listarea modulelor existente: **npm list**

căutarea unui modul: **npm search modul**

eliminarea unui modul: **npm uninstall modul**

actualizarea unui modul: **npm update modul**

node.js: **module** – node-sqlite-purejs

Exemplificare:

utilizarea modulului **node-sqlite-purejs**
pentru operații cu baze de date relaționale SQLite

nu depinde de alte module

detalii la <http://npmjs.org/package/node-sqlite-purejs>

```
var sql = require ('node-sqlite-purejs');

var nume = [ 'Tuxy', 'Pingou', 'Ux', 'Ping', 'Ingu', // nume fictive de pinguini
            'Nes', 'Xut', 'Nipu', 'Guin', 'Esc' ];

// operații SQL uzuale: creare tabelă, inserare date, interogare
sql.open ('./pinguini.sqlite', {}, function (eroare, bd) {
  bd.exec ('CREATE TABLE tucsi (id integer, nume text, varsta integer)');
  for (var id = 0; id < 10; id++) { // inserăm date (vârsta e generată aleatoriu)
    bd.exec ('INSERT INTO tucsi VALUES (' + id + ', "'
      + nume[id] + '", ' + Math.round (Math.random () * 60) + ');');
  };

  // preluăm pinguinii adulți
  bd.exec ("SELECT * FROM tucsi WHERE varsta >= 18 ORDER BY nume;",
    function (eroare, rezultate) { console.log (rezultate); }
  );
}
);
```

node.js: module – studiu de caz

Dorim să realizăm un mini-robot care procesează
conținutul diverselor resurse disponibile pe Web

rol de client pentru un server Web

prelucrează codul HTML ► *Web scrapping*
(metode consacrate: DOM sau SAX)



în cursurile
viitoare

```
var http    = require ('http');
var qs      = require ('querystring');
var xpath   = require ('xpath');
var dom     = require ('xmldom').DOMParser;

// de exemplu, dorim să obținem reprezentarea corespunzătoare resursei
// de la http://www.php.net/manual-lookup.php?pattern=cookie&scope=quickref
var param = {                                // stabilim parametrii cererii HTTP
  host: 'www.php.net', port: 80, method: 'GET',
  path: '/manual-lookup.php' + '?' + qs.stringify ( { pattern: 'cookie', scope: 'quickref' } )
};

var cerere = http.get (param, function (raspuns) {
  var rezultat = "";                        // răspunsul poate sosi în fragmente de date
  raspuns.on ('data', function (date) {    // tratăm evenimentul privitor la apariția datelor
    rezultat += date;
  });
  raspuns.on ('end', function() {          // tratăm evenimentul de finalizare a transferului
    console.log (procesareHTML (rezultat));
  });
});
```

conectare la un server Web

```
// realizează procesarea dorită (Web scrapping)
function procesareHTML (document) {
    var adrese = [];
    // instanțiem un procesor DOM
    var doc = new dom().parseFromString (document);
    // selectăm via o expresie XPath elementele <a> incluse în <li>
    var noduri = xpath.select ("//li/a", doc);
    // obținem valoarea atributului 'href' (URL-ul)
    for (var i = 0; i < noduri.length; i++) {
        adrese.push (noduri[i].getAttribute ('href')); // plasăm în tablou
    }
    return (adrese);
}
```

procesare document cu DOM și XPath

NODE.JS MODULES

POPULAR **INTERESTING** NEW

TAGS

- | Rank | Module Name | Percentage | Description |
|------|------------------|------------|---|
| #1 | npm | 98.4% | A package manager for node |
| #2 | express | 98.2% | Sinatra inspired web development framework |
| #3 | mocha | 97.3% | simple, flexible, fun test framework |
| #4 | jade | 97.2% | Jade template engine |
| #5 | connect | 96.9% | High performance middleware framework |
| #6 | socket.io | 95.3% | Real-time apps made cross-browser & easy with a WebSocket-like... |
| #7 | mongodb | 95.3% | A node.js driver for MongoDB |
| #8 | stylus | 95.0% | Robust, expressive, and feature-rich CSS superset |

jade: Jade template engine

Find tag...

all javascript client api simple
browser server framework web
module parser http nodejs
implementation test css json file
express data cli tool engine
testing coffeescript object
database html based middleware
wrapper command connect
template easy language files
asynchronous async build code
fast stream ender line mongodb
utility generator compiler rest
system applications jquery
templates dom support bdd
lightweight objects tools bindings

lista modulelor Node.js – <https://nodejsmodules.org/>






node.js: *framework-uri web*

Uzual, încurajează dezvoltarea de aplicații Web
în care interacțiunea cu utilizatorul
se realizează într-o singură pagină

realtime single-page Web apps

Full-stack frameworks

That's where Node.js really shines. The full-stack MVC frameworks are bundled with scaffolding, template engines, websocket and persistence libraries to allow you build real-time scalable web apps.

- ✓ Derby 
- ✓ SocketStream 
- ✓ MEAN.io 
- ✓ Meteor 
- ✓ TWEE.IO
- ✓ Mojito 
- ✓ SANE 

Alte facilități notabile:

MVC (*Model-View-Controller*) și variantele
transfer de date în timp-real – *e.g.*, cu **Socket.IO**
servicii Web – paradigma REST
suport pentru baze de date NoSQL
template-uri de prezentare a conținutului

<http://nodeframework.com/>

node.js: *framework-uri* web – exemple

Derby
Express
Flatiron
Geddy
Locomotive
Meteor
Mojito
SocketStream
TowerJS

node.js: utilizări pragmatice

Deservirea unui volum mare de conexiuni concurente cu necesar minim de resurse (procesor, memorie) într-un singur proces


node.js: utilizări pragmatice

Procesarea în timp-real
a datelor JSON oferite de API-uri (multiple)

inclusiv, crearea de aplicații
oferind fluxuri de date (*streaming data*)
de experimentat <http://nodestreams.com/>

node.js: utilizări pragmatice

Dezvoltarea rapidă de servicii Web
sau API-uri conform paradigmei REST
(*REpresentational State Transfer*)



în cursurile
viitoare

framework-uri pentru dezvoltarea de API-uri:
actionHero.js, facet, Frisby, restify, restmvc, percolator,...

rezumat



dezvoltarea de aplicații Web la nivel de server
de la JavaScript la Node.js
caracteristici, module, exemple

```

1  <!-- Modelarea unui lumi 3D via XML -->
2  <!-- o sfera oranj, stralucitoare -->
3  <Transform>
4    <Shape>
5      <Appearance>
6        <Material diffuseColor="1 0.2 0"
7          specularColor="1 0.8 1" shininess="0.9"/>
8      </Appearance>
9      <Sphere radius="2"/>
10   </Shape>
11 </Transform>
12 <!-- un con turtit -->
13 <Transform rotation="0 1 0 1" translation="-1 0.7 -1">
14   <Shape>
15     <Appearance>
16       <Material diffuseColor="0.8 0.4 0"/>
17     </Appearance>
18     <Cone bottomRadius="3.5" height="1.5"/>
19   </Shape>
20 </Transform>
21 <!-- un text -->
22 <Transform translation="0 4 0">
23   <Shape>
24     <Text solid="false" string="Salut din X3D">
25       <FontStyle justify="MIDDLE"/>
26     </Text>
27     <Appearance>
28       <Material diffuseColor="0.8 1 0.3"/>
29     </Appearance>
30   </Shape>
31 </Transform>

```

episodul viitor:

un model de date pentru Web: familia XML