JAVASCRIT

Introduzione

JavaScript è stato ideato nel 1995, aggiunse alle pagine HTML, la possibilità di essere modificate in modo dinamico, in base all'interazione dell'utente con il browser (lato client).

In Javascript c'è bisogno di almeno tre strumenti:

- Editor
- Interprete o compilatore
- Debugger

Possibili tools:

- Notepad++
- Eclipse

Interprete o compilatore, JavaScript engine.

Engine dei comuni browser:

- V8, Google, utilizzato in Chrome e node.js
- Chakra, Microsoft, Explorer
- SpiderMonkey, Mozilla, Firefox

ECMAScript 6

ES6 è l'ultima versione rilasciata, non ancora supportata su tutti i browser e engine. Esiste lo strumento denominato **transpiler** in grado di tradurre un programma scritto in un determinato linguaggio in un programma equivalente scritto in un linguaggio diverso. Tra i più noti:

- TypeScript, Microsoft
- Traceur, Google
- Babel

Librerie e codice JS esterno

Esistono tre modi per inserire il codice JS in HTML:

- Inserire codice inline
- Scrivere blocchi di codice nella pagina
- Importare file con codice JS esterno

Codice inline

Consiste nell'inserire direttamente le istruzioni JavaScript nel codice di un elemento HTML, assegnandolo ad un attributo che rappresenta un evento.

Esempio:

<button typer="button" onclick="alter('Ciao!')>Cliccami</button>

L'attributo onclick rappresenta l'evento del clic sul pulsante del mouse, quindi in corrispondenza di questo evento verrà analizzato ed eseguito il codice JS assegnato. In questo caso specifico verrà visualizzato un box con la scritta Ciao!

Un altro approccio per l'inserimento di codice inline, utilizzabile però soltanto con i link:

 Cliccami

Invece di effettuare un ancora gli assegno un operazione.

Vantaggi: Immediato

Svantaggi: Scomodo per codici complessi, o per la definizione di variabili e funzioni.

Blocchi di codice, il tag <script>

Il tag <script> si utilizza per inserire blocchi di codice in una pagina HTML, esempio:

<script> alert('Ciao')</script>

Il paser HTML quando vede il tag <script> lo passa direttamente all'engine JavaScript.

Possiamo inserirlo sia nella sezione <head> sia nella sezione <body>

IMPORTANTE

Se il codice JS interagisce con un elemento HTML, occorre assicurarsi che tale elemento sia già stato analizzato dal parser HTML

Il tag <script> prende alcuni attributi come language e type, nati per specificare il linguaggio di scripting contenuto al suo interno, non più necessario.

JS ESTERNO

Questa tecnica permette di agganciare script e librerie in modo non intrusivo, come accade per i fogli di stile CSS.

Per inserire un file JS esterno ci serviamo del tag <script> in cui specificando l'attributo src, esempio:

<script src="codice.js"></script>

Può essere sia relativo che assoluto, in questo caso non è necessario che il file JS risieda sullo stesso server della pagina.

Commenti, punti, virgole e maiuscole

Il punto e virgola in JS

Ciascun istruzione (o blocco di istruzioni) è delimitata da **;** , esempio:

var x = 5;

x = x + 1;

Si può anche non aggiungerlo, il parser lo fa in automatico, ma è meglio farlo.

Case sensitive e spazi bianchi

JS è case sensitive, cioè fa distinzione tra maiuscole e minuscole nei nomi di istruzioni, variabili e costanti.

JS non rileva gli spazi bianchi, si possono usare per rendere più leggibile il codice

Commenti

Esistono due tipi di commenti nel codice:

- Commento per singola riga, // seguito dal commento
- Commento multi riga, /* commento */

Tipi di dati in JS

JS prevede cinque tipi di dato primitivi, numeri, stringhe, booleani, null e undefined, e un tipo di dato complesso, gli oggetti (array, espressioni regolari, funzioni, etc).

JS converte automaticamente un tipo primitivo nel corrispondente oggetto quando utilizziamo un suo metodo o una sua proprietà. Esempio:

var nome = “Domenico”.toUpperCase();

JS trasforma la variabile di tipo primitivo stringa in un oggetto di tipo String e invoca il metodo toUppercase();

Stringhe

Un stringa in JS è una sequenza di caratteri delimitata da doppi o singoli apici. Esempio:

“Hello World”

‘Ciao Mondo’

Non vi è differenza tra l’uno e l’altro.

Un tipo speciale di stringa è la stringa vuota (senza caratteri, “” o ‘’).

Per inserire caratteri speciali all’interno di una stringa si fa ricorso al carattere di escaping \. Esempio

“Ieri pioveva.\nOggi c’è il sole.”

Stampa a schermo

Ieri pioveva.

Oggi c’è il sole.

È possibile inserire caratteri Unicode in una stringa utilizzando la sequenza di escaping \u.

Numeri

JS ha un unico tipo di dato numerico, niente distinzione tra intero o decimale. Esempi

Var x = -10;

var zero = 0;

var y = 0.52;

è possibile rappresentare i valori numerici secondo la notazione scientifica:

var primo Numero = 12e3 //equivalente a 12 x 10^3 cioè 12000

Oltre alla classica notazione in base dieci, possiamo rappresentare i numeri in notazione ottale (inizia per 0) ed esadecimale (inizia per 0x), Esempio:

var x = 0123;

var y= 0x123;

Un altro valore numerico speciale è NaN, Not a Number, che indica un valore numerico non definito. Esempio

var x = x + 1;

Booleano, null undefined

Il tipo di dato null prevede il solo valore null, che rappresenta un valore che non rientra tra i tipi di dato del linguaggio. Esempio

Var x = null;

Il tipo di dato undefined, rappresenta un valore che non esiste

Il tipo di dato booleano prevede solo: true or false;

Variabili, costanti e dichiarazioni

In JS una variabile è identificata da un nome che deve rispettare alcune regole:

- Non deve coincidere con una delle parole chiave del linguaggio
- Non può iniziare per numero
- Non può contenere caratteri speciali, esempio: spazio, trattino etc. sono permessi pero solo: $ e _

Dichiarazione implicita della variabili

JS non prevede la dichiarazione obbligatoria delle variabili, cioè un'operazione esplicita di creazione. Il semplice utilizzo di un identificatore indica all'engine di creare implicitamente la veriabile, se non esiste già. Anche se non è obbligatorio è importante per i codici più complessi utilizzare la dichiarazione var.

Strict mode

È un operazione introdotta nella versione 5 dello standard ECMAScript che ci consente, tra le altre cose, di ricevere una segnalazione di errore quando non dichiariamo le variabili. Offre i vantaggi di abituarci ad uno stile di programmazione più strutturato, sia di prevenire eventuali futuri malfunzionamenti. Per abilitare lo strict mode sarà sufficiente inserire all'inizio del nostro codice, il comando:

"use strict"

Le costanti

Prima della versione 5 di ECMAScript non è presente il concetto di costante, cioè di un valore non modificabile. Si utilizza una convenzione di una variabile, Esempio

Var PIGRECO = 3.14;

a partire dalla versione 6 dello standard viene introdotta la possibilità di dichiarare costanti tramite la parola chiave cost.

Espressioni e operatori

Un'espressione è una combinazione di valori, variabili ed operatori che rappresentano un nuovo valore. Esempio:

x + 1

è evidente che un ruolo fondamentale nelle espressioni è assunto dagli operatori, dal momento che determinano il valore risultante dell'espressione.

JS prevede operatori unari, binari e ternari a seconda che possano combinare rispettivamente uno, due o tre valori.

Operatori aritmetici

- +, addizione
- -, sottrazione
- /, divisione
- *, moltiplicazione
- %, modulo o resto

Seguono le regole di precedenza matematiche.

Prevedono anche tre operatori unari

- -, negazione
- ++, incremento
- --, decremento

Operatori relazionali

- <, minore
- <=, minore o uguale
- >, maggiore
- >=, maggiore o uguale
- ==, uguale
- !=, diverso
- ===, strettamente uguale
- !==, strettamente diverso

Restituiscono un valore booleano.

Operatori logici

- &&, and
- ||, or
- !, not

Restituiscono un valore booleano.

Operatori bitwise ("bit a bit")

Trattano i valori numerici come sequenze di bit applicandovi le relative operazioni.

Sono:

- &, confronta a coppie i bit degli operandi e restituisce 1 se entrambi i bit sono 1, 0 altrimenti.
- | confronta a coppie i bit degli operandi e restituisce 1 se almeno uno dei bit è 1, 0 altrimenti.
- ^, xor, restituisce 1 se uno dei bit, ma non entrambi, è 1 , 0 altrimenti.
- ~, not, inverte il valore di ciascun bit .
- <<, left shift, sposta di n posizioni verso sinistra la rappresentazione binaria di un numero.
- >>,right shift, sposta di n posizioni verso destra la rappresentazione binaria di un numero.

Operatori ?, per assegnamenti condizionali

L'unico operatore ternario previsto da JS è l'operatore condizionale. Esso restituisce un valore in base ad una espressione booleana. La sua sintassi è:

condizione ? valore1 : valore2

Se condizione è vera viene restituito valore1, altrimenti valore2

Esempio

x%2 == 0 ? "pari" : "dispari"

Operatori di assegnamento compositi

Esistono altri operatori di assegnamento derivati dalla combinazione degli operatori aritmetici e degli operatori sui bit con il simbolo =.

Forma compatta	Scrittura equivalente
X += y	X = x + y
X -= y	X = x - y
X*=y	X = x * y
X /=y	X=x/y
X%=y	X= x% y

Operatori su stringhe

È un operatore di concatenazione. Esso consente di creare una nuova stringa come risultato della concatenazione di due stringhe ed è rappresentato dal simbolo del "più" (+);

"piano" + "fronte" // "pianoforte"

Oppure

Var strumento ="piano"

Strumento += "forte" //strumento = "pianoforte"

Conversioni tra tipi di variabili

Le variabili JS possono contenere valori di qualsiasi tipo e cambiare senza problemi il tipo di dato contenuto. Ogni tipo di dato primitivo può essere convertito in un altro tipo di dato primitivo:numeri, stringhe, booleani, undefined e null.

Iniziamo col vedere come si comporta JS quando deve convertire un valore in booleano. Le seguente tabella riepiloga le conversioni implicite previste.

Tipo	Valore booleano
Undefined	False
Null	False
Numero	False se 0 o NaN, true in tutte gli altri casi
Stringa	False se stringa vuota, true in tutti gli altri casi.

La conversione di un valore in numero segue le regole riassunte nella seguente tabella:

tipo	Valori numerici
Undefiend	Nan
Null	0
Booleano	1 se true 0 se false
Stringa	Intero, decimale, zero o NaN, in base alla specifica stringa

Conversione implicita di un valore in stringa, la tabella di conversione è la seguente:

Tipo	Valore stringa
Undefined	“undefined”
Null	“null”
Booleano	“true” o “false”
Numero	“NaN” o “Infinity” o “valorenumero”

Operatori polimorfi

Sono operatori che prevedono operandi di tipo diverso. L’engine JS si trova a dover fare una scelta ben precisa per stabilire verso quale tipo di dato convertire gli operandi. Le regole seguite variano in base agli specifici operatori. Nel caso dell’operatore + la regola stabile che: se almeno uno dei due operandi è una stringa, allora viene effettuata una concatenazione di stringhe, altrimenti viene eseguita una addizione.

Evitare le conversioni implicite

Le conversioni implicite di JS sono spesso fonte di bug.

parseInt e parseFloat

è opportuno convertire esplicitamente un valore di un tipo in un altro tipo, ricorrendo ad alcune funzioni predefinite.

parseInt() converte una stringa in un valore intero. Esempio

parseInt(“12”) //12

parseInt(“12abc”) //12

parseInt(“a12bc”) //NaN

parseInt(“12”,8) // 12 in ottale

parseFloat per convertire in un valore decimale.

Typeof, verifica il tipo delle variabili

In qualsiasi momento si può verificare il tipo della variabile con l’operatore typeof.

Definire Array in JS

Gli array consentono di associare più valori ad un unico nome di variabile. L’uso degli array evita di definire più variabili e semplifica lo svolgimento di operazioni cicliche su tutti i valori.

Var nomearray = [valore1, valore2, …, valoren]; // array di dimensione n+1, si parte da 0

per accedere al contenuto si usa questa sintassi

x = nomearray[n] //n valore cella da conoscere

Un array può essere vuoto nomearray[];

in JS gli array possono contenere anche diversi tipi di primitive.

Array Multidimensionali

O anche detti matrici

Var nomematrice = [[valore11, valore 12, valore 13],[valore21,valore22,valore23]];

per ottenere un elemento della matrice

var x = nomematrice = [n] [m];

If, istruzioni condizionali e blocchi di codice

Blocco di codice

Un blocco di codice è racchiuso tra parentesi graffe, serve per evidenziare che un gruppo di istruzioni deve essere eseguito interamente. Esempio

```
{
        X=x+2;
        Y=x+3;
}
```

Istruzioni condizionali: if

In JS esistono due istruzioni condizionali : if e switch

If esiste in tre forme:

- Semplice
- If else
- If a cascata (if else if else….)

If semplice

```
If (condizione) {
//istruzioni
}
```

If else

```
If ( condizione) {  // se si verifica esegue la prossima istruzione
// istruzioni
}  else { // altrimenti questa
// istruzioni
}
```

If a cascata

If (condizione {

//istruzione

} else if {

//istruzione

} else {

//istruzione

}

Switch case

Quando siamo di fronte a diverse alternative anche un if a cascata può risultare difficile da leggere.In questi casi possiamo ricorrere all'istruzione switch, il cui schema è:

switch (espressione) {

case espressione1:

break;

case espressione2:

istruzioni2

break;

// …

Default:

istruzioni4

breack;

}

L'espressione di riferimento dello switch viene confrontata in sequenza con le espressioni dei vari case. Non appena viene individuata un'espressione corrispondente si esegue il blocco di istruzioni associato. Se non si trova nessuna corrispondenza verrà eseguito il blocco associato alla parola chiave default. La presenza di un caso di default è comunque opzionale.

Cicli

In JS ho while e for

While

While (condizione) { //finché vera esegue il ciclio

//istruzioni

}

Variante

Do {

//istruzione

}

While (condizione) // viene eseguita sicuramente una volta l'istruzione

For

For (inizializzazione; condizione; modifica) {

//istruzioni

}

Le inizializzazioni possono essere multiple o vuote

Per lavorare più comodamente con gli array JS prevede due varianti del for

For in e for of

Oltre al comando break esiste il comando continue che serve per interrompere l'esecuzione della singola interazione, saltando le istruzioni che seguono per riprendere da capo il blocco

Funzioni in JS

Una funzione è un insieme di istruzioni racchiuse in un blocco di codice, che può essere contraddistinto da un nome, può accettare argomenti o parametri di ingresso e restituire valori. Il suo utilizzo all'interno di uno script prevede due fasi:

- Una fase di definizione o dichiarazione della funzione in cui si assegna un nome ad un blocco di codice
- Una fase di invocazione o chiamata in cui il blocco di codice viene eseguito

Definire una funzione in JS

Function nome(argomenti) {

//istruzioni

}

Tra le istruzioni possibile esiste una fondamentale, return, che al termine della funzione ci restituisce un valore.

Invocare una funzione

Nome(valori);

Array arguments

Ci permette di accedere ai valori passati in fase di chiamata. La disponibilità di arguments ci consente di creare funzioni con un numero di parametri non definito.

Variabili globali e locali

Lo scope o ambito di visibilità di una variabile è la parte di uno script all'interno del quale si può fare riferimento ad essa. Le variabili dichiarate all'interno di una funzione sono dette "locali" alla funzione dal momento che sono accessibili soltanto all'interno del suo corpo.

Le variabili dichiarate fuori da qualsiasi funzione sono dette globali e sono accessibili da qualsiasi punto dello script, anche all'interno di funzioni.

Se abbiamo bisogno di creare uno scope specifico per una o più variabili possiamo ricorrere all'istruzione let che consente di dichiarare in modo analogo a var, ma limita lo scope della variabile al blocco di codice.

Funzioni predefinite

- parseInt()
- parseFloat()
- isNaN(), restituisce true se il suo valore è NaN
- isFinite(), restituisce true se diverso da Infinity o NaN
- escape(), sostituisce lo spazio con %
- unescape()
- encodeURI, versione migliorata di escape
- decodeURI
- encodeURIComponent(), funzione che include i caratteri mancanti da encodeURI
- decodeURIComponent()
- eval(), esegue una stringa come un codice JS

Oggetti in JS

Ciò che non è un tipo di dato primitivo è un oggetto. Un oggetto è un contenitore di valori eterogenei, messi insieme a formare una struttura dati unica e tale da avere generalmente una particolare identità.

Un oggetto tipicamente possiede:

- Dati, detti proprietà
- Funzionalità, detti metodi

Object literal

Var nome oggetto = {};

var nome oggetto={ nomeproprieta : valore , nome proprieta: valore, …..};

per accedere ad un valore:

var x = nomeoggetto.nomeproprieta;

le proprietà si possono aggiungere mano a mano definendole come una variabile

Metodi

A differenza delle proprietà di un oggetto che rappresentano i dati, i metodi rappresentano attività che un oggetto può compiere. La definizione di un metodo per un oggetto è abbastanza simile alla definizione di una funzione, esempio

Funzione visualizzaNomeCognome() { return " Domenico Pepino"}

Persona.nomeCognome = visualizzzaNomeCognome;

this

JS mette a disposizione la parola chiave this, che rappresenta l'oggetto a cui appartiene il metodo invocato. Esempio:

persona.nomeCognome= function () { return this.nome + " " + this.cognome;}

Passaggio di variabili alle funzioni: per valore o per riferimento

Il passaggio di valori relativi a tipi di dato primitivi avviene sempre per valore mentre il passaggio di oggetti avviene sempre per riferimento.

Object

Ogni oggetto, predifinito o meno che sia, è costruito su Object . Questo fa si che tutti gli oggetti in JS abbiano alcune caratteristiche comuni.

var nome variabile = new Object();

nomevariabile.nomeproprieta = valore;

ToString e ValueOf

Visto che tutti gli oggetti JS sono basati su Object, essi condividono alcuni metodi: toString() e valueOf().

Il primo metodo restituisce una versione in stringa dell'oggetto.

Il secondo restituisce il corrispondete valore del tipo di dato primitivo associato all'oggetto.

Number

L' oggetto Number fornisce metodi e proprietà per la manipolazione di valori numeri.

Var x = new Number(123);

Possiamo usare anche Object ma è meglio usare Number

Proprietà piu utilizzate:

- EPSILON, la più piccola differenza tra la rappresentazione di due numeri
- MAX_VALUE , il piu grande valore numerico positivo rappresentabile
- MIN_VALUE, il più piccolo numero positivo rappresentabile diverso da zero
- NaN , Un valore non numerico
- NEGATIVE_INFINITY
- POSITIVE_INFINITY

Metodi piu utilizzati

- isFinite(n)
- isInteger(n)
- isNaN(n)

Ciascun istanza di un oggetto Number ha disposizione tre metodi di rappresentazione

- toExponential(), rappresentazione esponenziale del numero.
- toFixed(n), restituisce una stringa con la cifra con un numero di decimali n.
- toPrecision(n), prende come argomento un numero opzionale e restituisce una stringa che rappresenta il numero della precisione specifica

Math

Non consente la creazione di nuove istanze, funziona come un API. È un oggetto statico che mette a disposizione proprietà e metodi richiamabili da qualsiasi punto di uno script

Proprietà :

- E, costante di Eulero
- LN2
- LN10
- LOG2E
- PI
- SQRT1_2
- SQRT2

Metodi:

- max()
- min()
- pow()
- sqrt()
- ceil(), approssimazione per eccesso
- floor(), per difetto
- round(), il piu vicino

Random()

genera un numero casuale compreso tra 0 e 1

altri metodi minori

abs(), valore assoluto

log(), log naturale

sin(), cos(), tan(), atan()

String

Tramite l'operatore new possiamo creare istanze di oggetti String.

Proprieta e metodi

- lenght, restituisce la lunghezza della stringa
- charAt(n) estrae il carattere n da una stringa
- replace() sostituisce una sottostringa con un'altra
- indexOf(), restituisce la posizione della prima occorrenza della stringa passata.
- lastIndefOf(), restituisce la posizione dell'ultima occorrenza
- substr(n,m), estrae la sottostringa dalla posizione n di lunghezza m
- substring(n,m) estrae la sottostringa dalla posizione n a m
- slice, come substring
- split(), c rea un array a partire da una stringa. Se inserisco un carattere sara esso il delimitatore dello split
- toLowerCase(), tutto minuscolo
- toUpperCase(), tutto maiuscolo
- trim(), elimina gli spazi
- startsWith() e endsWith(), verificano se una stringa inizia o finisce con la stringa nell'argomento

Template string
Permettono di risolvere i problemi di composizione di una stringa complessa con estrema semplicità ed eleganza. Una template string e una sequenza di caratteri delimitata da ‘ al posto di singoli o doppi apici. Per elaborazioni avanzate possiamo utilizzare le tagged template string.

RegExp

Serve per la creazione di un'espressione regolare (pattern)

Test

Consente di verificare se una stringa individuata dall'espressione regolare è contenuta nella stringa passa come argomento

Exec

Restituisce a differenza di test un array con la sottostringa individuata o null

Date e orari in JS

L’oggetto Date è utilizzato per la creazione di istanze di date.

Metodi

- getFullYear()
- getMonth()
- getDate()
- getDay()
- getHours()
- getMinutes()
- getSeconds()
- getMilliseconds()
- setFullYear()
- setMonth()
- setDate()
- setHours()
- setMinutes()
- setSeconds()
- setMilliseconds()
- setTime()
- getUTCDate()
- setUTCDate()
- getTimezoneOffset()
- toDateString()
- toISOString()
- toLocaleDateString()
- toLocaleTimeString()
- toLocaleString()
- toTimeString()
- toUTCString()

Array come oggetto

gli array possono essere creati anche sfruttando l'oggetto Array. Esempio:

var x = new Array();

proprietà e metodi :

- length
- push(),aggiungi elemento
- pop(), elimina elemento
- shift(), elimina il primo elemento
- unshift(), aggiunge un elemento in cima
- splice(), per l'aggiunta o rimozione di un elemento in un punto n
- concat(), per unire due array (accoda)
- sort(), ordina un array
- reverse(), ordina in ordine diverso
- indexOf(), da la posizione dell'elemento richiesto
- lastIndexOf(), posizione dell'ultimo elemento
- join(), converte un array in stringa

Typed Array

Oltre ai tradizionali array, ECMA 6 aggiunge nuove strutture dati al linguaggio arricchendolo di nuove potenzialità e di maggiore flessibilità, Typed Array, Set e Map.

I Typed Array sono strutture dati che consentono la manipolazione efficiente di dati binari. Le specifiche prevedono due tipi di oggetto:

- ArrayBuffer , oggetto che rappresenta un blocco di dati senza alcun specifico formato ne meccanismi per accedere al suo contenuto.
- ArrayBufferView, oggetto che fornisce un tipo di dati e una struttura per interpretare i dati binari trasformandoli in un effettivo Typed Array
- Fillbuffer() si occupa di caricare i dati binari nel buffer

Set

Può contenere dati di qualsiasi tipo ma senza duplicati.

Le operazioni più comuni sono l'aggiunta e la rimozione.

Metodi:

- Add(), aggiunge un elemento
- Size(), indica il numero di elementi contenuti
- Has(), indica se l'elemento è presente
- Delete(), elimina un elemento
- Clear(), elimina tutti gli elementi

MAP

Permette di creare mappe associate che ci consentono di abbinare un valore ad una chiave.

Utilizza gli stessi metodi di set, in piu abbiamo get() che consente di accedere al valore associato ad una chiave

Funzioni come oggetti

In JS una funzione è in realtà un oggetto di tipo Function.

Le funzioni inoltre a differenza di altri linguaggi non richiedono il nome di essa, se assegnata direttamente ad una variabile

Callback

Visto che le funzioni sono degli oggetti, possono essere passate come parametri di un'altra funzione. La funzione passata come parametro è detta generalmente funzioni di callback. Un aspetto a cui prestare attenzione quando scriviamo funzioni che accettano callback è quello di accertarsi che venga passata effettivamente una funzione prima di invocarla. Esempio,

```
function calcola(func, arg1, arg2) {
        if ( func && typeof func == "function") {
                return func (arg1, arg2);
        }
}
```

Le funzioni oltre ad essere inserite come argomento possono essere anche restituite con il return,Esempio

```
Var  incrementatore = function(incremento){
        Return function(valore) {
                Return incremento + valore;
        };
};
```

Corretto utilizzo di this

Si utilizzano due metodi per specificare il significato che intendiamo associare alla parola chiave this.

Il primo metodo è call() che permette di invocare una funzione impostando il primo parametro come oggetto di riferimento per this ed i parametri successivi, in numero variabile, come valori da passare alla funzione.

Il secondo metodo è apply(), del tutto simile a call() con la differenza che prevede due soli parametri: il primo è l'oggetto da associare a this mentre il secondo parametro è un array dei valori da passare alla funzione da invocare.

Nel ECMA 5 è stato introdotto un terzo, bind() che consente di creare una nuova funzione con l'oggetto this preimpostato.

Closure e scope

Abbiamo due tipi di scope:

- scope globale, accessibilità estesa all'intero script
- scope locale, accessibilità ristretta al solo codice di una funzione o di un blocco di codice

Il closure di una funzione invece è un meccanismo che stabilisce che ogni variabile che era accessibile quando una funzione è stata definita rimane racchiusa nello scope accessibile dalla funzione.

Arrow function

Sono funzioni anonime (senza nome) , si spiega meglio per esempio

```
Var somma = function(x,y) {

        Return x+y;

};
```

Implement con la arrow function

```
Var somma = (x , y) => x + y;
```

non è possibile utilizzarle per richiamare i metodi.

Varianti di arrow function

```
(x, y) => { return x+y}

X => x*2

() => "hello world"
```

Costruttori

Per evitare di dover ridefinire da zero oggetti che hanno la stessa struttura possiamo ricorrere ad un costruttore. Un costruttore non è altro che un normale funzione JS invocata mediante l'operatore new.

La flessibilità degli oggetti JS si esprime principalmente nella possibilità di modificare la struttura anche dopo la creazione. Per modificare la struttura si utilizza: prototype, esempio di utilizzo

Oggetto.prototype.nuovaproprietà = valore;

non viene fisicamente modificata la struttura ma viene agganciato , uso radice dell'albero

inoltre per creare un oggetto possiamo utilizzare il metodo:

Object.create();

Descrittori delle proprietà

Un descrittore è un oggetto che definisce caratteristiche e modalità di accesso alle proprietà di un oggetto. Possiamo distinguere due tipi di descrittori:

- data descriptor, un oggetto che definisce una proprietà specificando una serie di caratteristiche predefinite;
- accessor descriptor, descrive una proprietà tramite una coppia di funzioni di tipo getter e setter

esempio,

```
var marioRossi = Object.create(
  persona.prototype, {
        nome: {
              value : "Mario",
              writable: false,
              configurable: false}
};
```

proprietà data descriptor:

- writable, booleano che indica se il valore delle proprietà può essere modificato
- configurable, booleano che indica se il tipo di descrittore può essere modificato e se la proprietà può essere rimossa
- enumerable, booleano che indica se la proprietà è accessibile durante un ciclio sulle proprietà dell'oggetto
- value, indica il valore della proprietà

proprietà accessor descriptor:

- configurable
- enumerable
- get, funzione senza argomenti invocata quando si accede alla proprietà in lettura
- set, funzione chiamata quando si accede in scrittura. Il nuovo valore da assegnare alla proprietà viene passato come parametro.

Esempio

```
_email: { value: "", writable: true, configurable: true },

email: {
        get: function() {
                return this._email;
        },
        set: function(value) {
                var emailRegExp = /\w+@\w+\.\w{2,4}/i;
                if (emailRegExp.test(value)) {
                        this._email = value;
                } else {
                        console.log("Email non valida!");
                }
        }
}
```

Classi

Una classe è un modello per creare oggetti.

Class nome extends nome classe { // extends solo se si estende un'altra classe

```
        Constructor() {

        Super(); // solo se si esetende

        Istruzioni
}
```

La classe Proxy

Consente di creare oggetti che hanno la capacità di modificare il comportamento predefinito di altri oggetti.

Nella definizione di un proxy per un oggetto possiamo definire un handler e configurare trap per intercettare l'accesso alle sue proprietà ed eventualmente modificare il comportamento predifinito.

Per comprendere i concetti di base di un proxy, proviamo a fare un semplice esempio. Supponiamo di voler tracciare sulla console ogni accesso alle proprietà di un oggetto. Possiamo definire il seguente handler:

```
var handler = {
  get(target, propertyName) {
    console.log("Lettura di " + propertyName);
    return target[propertyName];
  },
  set(target, propertyName, value) {
    console.log("Assegnamento di " + value + " a " + propertyName);
    target[propertyName] = value;
  }
};
```

Copy

Questo handler non è altro che un oggetto con due metodi, get() e set(), che intercettano rispettivamente gli accessi in lettura e scrittura alle proprietà dell'oggetto che vogliamo monitorare. I metodi dell'handler

sono chiamati trap e consentono di intercettare accessi e manipolazioni relative all'oggetto di destinazione, il target.

Nello specifico, il metodo get() scrive sulla console e restituisce il valore della proprietà del target, mentre il metodo set() scrive il valore sulla console ed assegna il valore del parametro value alla proprietà del target. In questo caso vogliamo mantenere il comportamento standard dell'oggetto target, ma in generale possiamo restituire o assegnare alla proprietà del target qualsiasi valore, modificando quindi il comportamento predefinito.

Una volta definito l'handler, possiamo creare un proxy per un oggetto specificandolo nel costruttore della classe Proxy, come nel seguente esempio:

```
var persona = {nome: "Mario", cognome: "Rossi"};

var personaProxata = new Proxy(persona, handler);
```

Copy

Abbiamo creato un oggetto persona e lo abbiamo passato insieme all'handler al costruttore della classe Proxy. D'ora in poi ogni accesso alle proprietà dell'oggetto personaProxata avrà effetto sull'oggetto persona e verrà intercettato e loggato sulla console:

```
var nome = personaProxata.nome;

 //console: Lettura di nome

 personaProxata.nome = "Marco";

 //console: Assegnamento di Marco a nome

 console.log(persona.nome);

 //console: Marco
```

Copy

Naturalmente questo è un semplice esempio per introdurre i concetti di base per l'utilizzo della classe Proxy. È possibile utilizzare altre trap per definire manipolazioni avanzate dell'oggetto target. Oltre a get() e set(), infatti, possiamo sfruttare le seguenti trap che intercettano i corrispondenti metodi dell'oggetto target:

getPrototypeOf()

setPrototypeOf()

isExtensible()

preventExtensions()

getOwnPropertyDescriptor()

defineProperty()

has()

deleteProperty()

ownKeys()

apply()

construct()

Data binding

Un altro ambito in cui possiamo utilizzare la classe Proxy è nell'implementazione del data binding, cioè nel meccanismo che lega le proprietà di due oggetti in modo che le modifiche si propaghino da uno all'altro. Nel contesto del data binding si parla di un oggetto che fornisce dati (data source object) e di un oggetto che li riceve (data target object). L'esempio tipico di applicazione del data binding è quello che associa una proprietà di un oggetto con un elemento dell'interfaccia grafica, come ad esempio una casella di testo. Vediamo come sfruttare la classe Proxy per implementare il meccanismo di data binding.

Definiamo una classe Binder con un metodo bindTo() come mostrato di seguito:

```
class Binder {
   bindTo(dataSourceObj, dataSourceProperty, dataTargetObj, dataTargetProperty) {
      var bindHandler = {
        set: function(target, property, newValue) {
          if (property == dataSourceProperty) {
            target[dataSourceProperty] = newValue;
            dataTargetObj[dataTargetProperty] = newValue;
          }
        }
```

```
      };

      return new Proxy(dataSourceObj, bindHandler);

    }

 }
```

Copy

Il metodo bindTo() definisce una trap che cattura gli accessi in scrittura al data source object, in modo tale che ogni modifica alla proprietà specificata dal parametro dataSourceProperty aggiorni la proprietà associata del data target object. Il metodo bindTo() restituisce il proxy creato a partire dal data source object. Possiamo quindi usare la classe Binder come nel seguente esempio:

```
var persona = {

    nome: "Mario",

    cognome: "Rossi"

 };

 var txtNome = document.getElementById("txtNome");

 var binder = new Binder();

 var personaConBinding = binder.bindTo(persona, "nome", txtNome, "value");

 setTimeout(function() {

    personaConBinding.nome = "Marco";

 }, 5000);
```

Abbiamo creato il proxy dell'oggetto persona utilizzando il metodo bindTo() della classe Binder. Nella chiamata al metodo bindTo() abbiamo specificato che vogliamo mettere in relazione la proprietà nome dell'oggetto persona con la proprietà value dell'elemento txtNome. In questo modo, ogni modifica alla proprietà nome dell'oggetto personaConBinding si rifletterà automaticamente sia sull'oggetto originario persona, sia sulla casella di testo.

I due esempi di applicazione concreta che abbiamo riportato danno un'idea delle possibili utilizzi della classe Proxy nello sviluppo di applicazioni di una certa complessità.

Symbol

Con ECMA 5 è stato introdotto un nuovo tipo di dato primitivo: il tipo di dato Symbol. Non prevedono una sintassi letterale. Per crearlo si usa Symbol(); Ogni valore Symbol() sono unici.

Serializzare gli oggetti in JavaScript

La possibilità di memorizzare dati o di rappresentarli in maniera da poterli trasferire tra programmi anche diversi è fondamentale nella programmazione. In questo ambito si può rientrare la serializzazione, cioe il processo di trasformazione di un oggetto in un formato facilmente memorizzabile e/o trasmissibile, ed il relativo processo inverso, cioè la de serializzazione.

Rappresentazione JSON

JSON utilizza un sottoinsieme della notazione letterale degli oggetti di JS, per rappresentarli sottoforma di stringa , Esempio

{nome: “Domenico”, cognome: “Pepino”}

Viene rappresentato in JSON con la seguente stringa

‘{nome: “Mario”, cognome: “Rossi”}’

In JSON non è previsto la rappresentazione dei metodi di un oggetto

Per deserializzare un JSON si utilizzano due metodi:

- parse(), prende in input una stringa JSON e genera il corrispondente oggetto JS
- stringify(), genera la rappresentazione JSON dell’oggetto passato come argomento

Gestione delle eccezioni

La gestione degli errori in runtime è un aspetto molto importante della programmazione.

Un’eccezione è un errore che si verifica in runtime dovuto ad un’operazione non consentita.

In JS si ricorre al costrutto try catch.

```
Try {

//blocco di codice

} catch(e) {

 // gestione dell’eccezione

}
```

Prevede la clausola opzionale finally che consente di specificare un blocco di codice da eseguire in ogni caso.

Proprietà:

- name, identifica il tipo di eccezione
- message, indica il messaggio specifico dell'eccezione verificatasi

per generare le eccezioni si utilizza l'istruzione throw

Design Pattern

I design pattern sono soluzioni tecniche a problemi comuni di progettazione del software.

Browser API, l'oggetto window

Tipicamente i progetti js sono eseguiti all'interno sistemi ospite con i quali interagire grazie a insieme di oggetti che espongono interfacce standard, meglio note come API.

L'interazione con il browser viene realizzta tramite un'API che consente di acquisire informazioni sull'ambiente di esecuzione, di usufruire di alcune funzionalità e di effettuare specifiche impostazioni.

L'oggetto principale per l'interazione con il browser è window: esso rappresenta una finiestra che contiene un documento HTML. Questo oggetto, oltre ad identificare l'elemento visivo del browser, rappresenta anche il contesto di esecuzione globale per JS, cioè l'oggetto all'interno del quale vengono definite variabili e funzioni globali.

Alcune proprietà ci consentono di ottenere informazioni su diversi aspetti della configurazione corrente del browser.

- innerHeight e innerWidth, ci consentono di ottenere le dimensioni interne dell'area occupata dalla finestra espresse in pixel.
- screen, ci fornisce informazioni su alcune caratteristiche dello schermo corrente.
- availWidth e availHeight, per conoscere le dimensioni effettivamente disponibili
- frames, è un array di oggetti window che rappresentano i frame contenuti nella pagina corrente .

metodi :

- alert(), crea una finestra modale con un messaggio ed un pulsanteok
- confirm(), crea una finestra modale con un messaggio con un pulsante di conferma e uno di annulla
- prompt(), crea un finestra di inserimento
- open(), apre una nuova finestra o tab, attributi utili _blank (nuova scheda),_parent (sostituisce la finestra o il frame genitore della finestra corrente), _self (sostituisce il contenuto della finestra o frame corrente), _top (sostituisce il conenuto della radice della gerarchia di oggetti window)

- setInterval(), esegue una funzione periodicamente in base ad un intervallo di tempo specificato
- setTimeout(), esegue una funzione dopo un certo numero di millisecondi
- clearInterval(), Azzera un timer creato con setInterval()
- clearTimeout(), Azzera un timer creato con setTimeout()
-