EPAI, Jérôme Frossard (2019)

JAVASCRIPT

Développer une interface utilisateur graphique

1

Introduction

Historique

- 1995 (mai) : Brendan Eich crée un langage de script pour le compte de Netscape Communications Corporation
- 1995 (décembre) : SUN et Netscape annoncent la distribution libre de droits de la spécification du langage JavaScript
- 1996 (mars) : Netscape intègre JavaScript dans son navigateur Netscape Navigator 2.0
- 1996 (août) : Microsoft sort Internet Explorer 3.0 avec une implémentation du langage nommé JScript
- 1996 (novembre) : la spécification du langage est soumise à l'ECMA et les travaux de standardisation débute
- 1997 (juin) : le standard ECMA-262, *ECMAScript Langage Specification* est adopté

Java… Script ?

- JavaScript (ECMAScript) est :
  - Un **langage de programmation** conçu en 1995 pour la réalisation de scripts dans les navigateur web.
  - Le seul langage **supporté par tous les navigateurs.**
  - La seule option pour le « code à la demande » depuis la disparition de Flash, des applets Java et autre Silverlight.
  - **Sans lien avec Java et la JVM**. La ressemblance de leurs syntaxes a été voulue pour des raisons marketing.
  - un **langage de programmation à usage général** grâce la plateforme Node.js.

Navigateur web et document HTML

- Fonctionnalités de base du navigateur :
  - Permettre à l'utilisateur de saisir une URL.
  - Présente de la meilleures façon possible les représentations de ressource web (document HTML, image, video, son, …).
- Fonctionnalités de base d'un document HTML :
  - Permettre à l'utilisateur de suivre un lien hypertexte.
  - Permettre à l'utilisateur d'interagir avec les contrôles d'un formulaire HTML (zones de texte, listes déroulantes, etc.).
  - Permettre à l'utilisateur de soumettre le contenu des champs d'un formulaire au serveur (bouton *submit*).

Document HTML et JavaScript

- JavaScript permet d'ajouter des fonctionnalités à un document HTML.
- La structure d'objets du document (DOM) accessible à travers la variable globale `window`.
- Exemple d'utilisations possibles :
  - Validation côté client du contenu d'un formulaire.
  - Construction dynamique de partie du document.
  - Modification dynamique du document en réponse à un événement.
  - Envoi de requêtes HTTP (Ajax).

Balise <script>

- La balise script permet d'inclure du code JavaScript dans un document HTML.
  - Inclusion directe (à éviter autant que possible) :

```
<script>
    window.alert("hello world!");
</script>
```

  - Inclusion du contenu d'un fichier (bonne pratique) :

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

Traitement des balises <script>

- Les balises <script> sont **traitées dans l'ordre d'apparition** dans le code HTML, comme les autres balises.
- Le contenu de la balise est **exécuté au moment où elle est traitée** (le reste du document n'est pas encore chargé).
- Les scripts et l'affichage sont exécutés dans le même thread, **l'affichage est figé durant l'exécution d'un script.**
- Conséquences :
  - Au moment de l'exécution d'une balise script, seuls les éléments HTML qui se trouvent avant la balise sont accessibles.
  - L'ordre d'inclusion des fichiers est important.
  - Il est préférable d'inclure les scripts à la fin du document.

Le langage JavaScript

Caractéristiques

- Syntaxe proche de celle de Java.
- Multiparadigme :
  - Impératif, procédural et structuré (séquence, affectation, sous-programmes et structures de contrôles)
  - orienté-objet (objets, polymorphisme, héritage)
  - fonctionnel (les fonctions sont des objets de première classe)
- **Typage dynamique** et faible :
  - Pas de déclaration de type, late-binding
  - Conversions de type implicites
- Utilisation de prototypes pour la création d'objets (**pas de classes**)

Programme JavaScript

- Séquence d'instructions qui partagent une instance de l'objet Global (dans un navigateur : `window`)
- Un seul programme par document HTML (par fenêtre)…
- … et non pas un programme par balise `<script>`.
- Instructions :
  - déclarations (variables et fonctions)
  - affectations et appels de procédure
  - contrôles de flux (choix, boucles)

Typage dynamique

- Le typage dynamique s'oppose au type statique.
- Langages statiquement typé (Java, C#) :
  - On spécifie le type d'un identificateur lors de sa déclaration.
  - Le type d'un identificateur ne peut pas changer.
- JavaScript est un langage dynamiquement typé :
  - On ne spécifie pas le type d'un identificateur lors de la déclaration.
  - Une variable adopte le type de la valeur qui lui est affectatée.
  - Une fonction adopte le type de la valeur de retour.
  - Par défaut, la valeur d'une variable est *undefined de type Undefined*.

Types de données

- Types primitifs
  - Number : n'importe quel nombre (IEEE 754-2008).
  - Boolean : valeur booléenne (true ou false).
  - String : chaîne de caractères Unicode.
  - Undefined : non initialisée (une seule valeur : undefined).
  - Null : pas de valeur (une seule valeur : null).
  - Object : collection de propriété
- Objets intrinsèques (*well-knonw intrinsic objets*):
  - Array, Math, Date, RegExpr, Error
  - Function, JSON

Type Number

- Un seul type : IEEE 754 (double précision) pour représenter :
  - Des nombres entiers exacts jusqu'à 15 chiffres,
  - Des nombres à virgule flottantes,
  - Des valeurs spéciales :
    - `Number.NaN` : not a number, différent de tous les nombre y compris lui-même ⇒ utiliser la fonction `window.isNaN()`.
    - `Number.POSITIVE_INFINITY` : $+\infty$
    - `Number.NEGATIVE_INFINITY` : $-\infty$

Type boolean

- Représente une valeur booléenne : vrai ou faux.
- Valeurs équivalentes à faux (*falsy values*) :
  - La constante `false`.
  - Les valeurs `undefined`, `null`, `0`, `NaN`.
  - Une chaîne de caractère vide.
- Valeurs équivalentes à vrai (*truthy values*) :
  - La constante `true`.
  - Un nombre différent de 0.
  - Une chaîne de caractère non vide.
  - N'importe quelle référence à un objet.

Littéral

DÉF. : Un littéral, ou valeur littérale, est une valeur écrite explicitement dans le code source d'un programme.

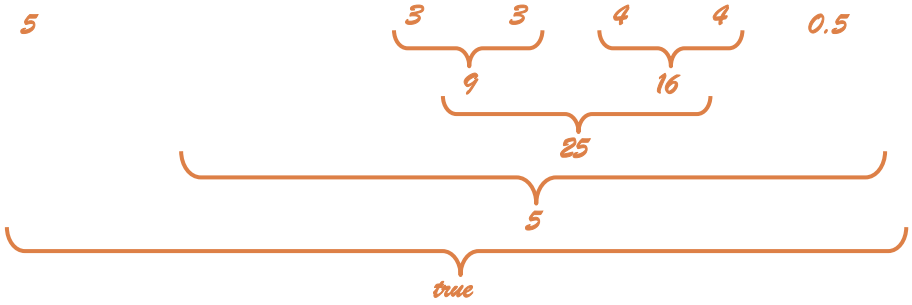
- En JavaScript, on dispose de notation littérale pour les types de valeurs suivantes :
  - les nombres (`2`, `-3`, `2.3`, `2e-10`, ...)
  - les booléens (`true`, `false`)
  - les chaînes de caractères (`"une chaîne"`, `'une chaîne'`)
  - les tableau (`[1, 2, 4, 8, 16, 32]`)
  - les objets (`{firstname: "john", lastname: "Backus"}`)

Expression (définition)

DÉF. : Une expression est une combinaison de littéraux, de variables et d'appels de fonction évaluée selon les règles de priorité et d'associativité du langage pour produire une nouvelle valeur.

- Exemple :

```
function isRightTiangle(a, b, c) {
    return c === Math.pow(a * a + b * b, 0.5);
}
```




```
const val = isRightTriangle(3, 4, 5);
```

Opérateurs

… arithmétiques	`+`, `-`, `*`, `/`, `%`, `++`, `--`
… de chaîne	`+`
… de comparaison	`===`, `!==`, `<`, `<=`, `>=`, `>`, `==`, `!=`
… booléens	`&&`, `||`, `!`
… logiques	`&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`
… d'affectation	`=`, `+=`, `-=`, `*=`, `/=`, `&=`, `|=`, `^=`, `<<=`, `>>=`, `>>>=`
… de condition	`?:`
… d'invocation	`()` appelle d'une fonction ou d'une méthode
… d'indexation	`[]` accès à une valeur indexée (tableaux, objets)
… divers	`.` , `()`, `typeof`, `new`, `instanceof`, `delete`, `void`, `,`

Instructions de déclaration

Identificateur

- Un identificateur est le nom d'une variable, d'une fonction, ou d'une propriété.
- Les minuscules et les majuscules ne sont pas équivalentes (sensible à la casse ou *case sensitive*).
- Doit respecter les contraintes suivantes :
  - Ne contient pas d'espace.
  - Commence par une lettre, _ ou $.
  - N'est pas un mot réservé :

```
break, do, instanceof, typeof, case, else, new, var, catch,
finally, return, void, continue, for, switch, while, debugger,
function, this, with, default, if, throw, delete, in, try, class,
enum,extends, super, const, export, import, implements, let,
private, public, yield, interface, package, protected, static
```

Portée (*scope*) d'un identificateur

DÉF. : La portée lexicale d'un identificateur est la portion de code dans laquelle cet identificateur est accessible.

- En JavaScript les différentes portées sont :
  - La portée globale : tout le programme.
  - La portée locale : le corps de la fonction où se trouve la déclaration, y compris les fonctions imbriquées.
- En JavaScript :
  - Si un identificateur n'a pas été déclaré avant d'être utilisé, il a une portée globale.
  - Un identificateur global est une propriété de l'objet Global.

Déclaration de variable

- Avec le mode strict activé (recommandé), la déclaration des variables est obligatoire.
- On déclare une variable à l'aide des mot clé **let** ou **const** suivi d'un **identificateurs** :

```
// déclare une variable nommée uneChaine dont la valeur
// ne peut pas être modifiée par une affectation.
const uneChaine = "Hello World";
// déclare deux variables nommées element et altitude dont
// la valeur peut être modifiée par une affectation.
let element, altitude;
```

- Le mot clé **var** ne devrait plus être utilisé et on utilise autant que possible le mot clé **const**.

Notion de sous-programmes

- En programmation, on distingue deux types de sous-programmes : les **procédures** et les **fonctions**
- Une procédure est un sous-programme qui :
  - **Ne doit jamais renvoyer une valeur.**
  - Doit avoir un effet (modification d'une variable globale ou d'une propriété d'un objet, requête Ajax avec la méthode POST, modification du document HTML, etc.)
- Une fonction est un sous-programme qui :
  - **Doit toujours renvoyer une valeur.**
  - Renvoie toujours la même valeur pour des paramètres donnés (transparence référentielle).
  - Ne devrait pas avoir d'effet.

Notion de sous-programme (suite)

- En JavaScript (comme avec beaucoup d'autres langages), on utilise le même mot clé (*function*) pour les deux types de sous-programmes.
- Le langage ne fait pas de différence entre procédure et fonction, il n'y aura pas d'erreur si les règles ne sont pas respectées.

Sous-programme (syntaxe simplifiée)


```
function <identificateur>(<paramètres formels>) {
    <instructions>;
    return <expression>;
}
```

- Exemple :

```
function computePosition(speed, time, initialPosition) {
    initialPosition = initialPosition || 0; //set default
    return initialPosition + speed * time;
}
```

- Paramètres formels :
  - Similaires à des variables locales.
  - Liés aux paramètres effectif par position.

Sous-programme (syntaxe générale)

- A la syntaxe simplifiée, on préfère la syntaxe générale utilisable dans tous les cas et qui consiste à **affecter une fonction anonyme (*lambda expression*) à une variable :**

```
const <identificateur> = function (<paramètres formels>) {
    <instructions>;
    return <expression>;
}
```

- Exemple :

```
const computePosition = function (speed, time, initialPosition) {
    initialPosition = initialPosition || 0; //set default
    return initialPosition + speed * time;
}
```

Fonction imbriquée

- Déclaration d'une fonction à l'intérieur d'une fonction (impossible avec la syntaxe simplifiée)
- Une fonction imbriquée se trouve dans la portée locale de la fonctions englobantes
- Utile pour les IIFE (*Immediately-invoked function expression*) pour éviter de polluer la portée globale (global scope).
- Utile pour réaliser des fermetures (*closure*).

Appel de fonction


```
<identificateur>(<paramètres effectifs>)
```

- Exemple :

```
var result, ;
result = computePosition(20, 60 * 60);
```

- Opérateur d'invocation
  - parenthèses immédiatement à la suite d'un identificateur
- Paramètres effectifs
  - valeurs initiales des paramètres formels
  - chaque valeur est lié au paramètre formel correspondant à sa sa position (speed reçoit 20, time reçoit 60 * 60 et initialPosition ne reçoit rien)

Expression avec effet de bord

Instruction et expression

- Dans un langage impératif, une instruction n'est utile que si elle modifie l'état du programme, on dit qu'elle a un effet.
- En JavaScript, toute expression est une instruction, mais seuls trois types d'expression ont un effet de bord :
  - L'affectation.
  - L'appel d'une procédure.
  - Les directives comme `"use strict";`

Affectation


```
<partie gauche (RHS)> = <partie droite (LHS)> ;
```

- L'affectation est une opération qui consiste à associer une valeur à une variable.
- Le signe égal est l'**opérateur d'affectation**.
- Une affectation a deux parties :
  - Partie gauche : le nom d'une variable.
  - Partie droite : une **expression**, c'est-à-dire la valeur qu'on souhaite associer à la variable.
- Exemple : `position = speed * time + initialPosition;`

Sémantique de l'affectation

- L'affectation exprime **l'association de l'identificateur** de la partie gauche **à la valeur** de la partie droite.
- Elle **n'exprime pas une égalité** : les parties gauche et droite ne sont pas interchangeables.
- C'est une expression dont la valeur est celle de l'expression de la partie droite.
- Bonne pratique : si une instruction contient une affectation, elle doit n'en contenir qu'une et ne doit rien contenir d'autre.

Appel de procédure (syntaxe)


```
<identificateur>(<paramètres effectifs>)
```

- Exemple : `alert("hello world!");`
- Opérateur d'invocation :
  - Parenthèses immédiatement à la suite d'un identificateur.
- Paramètres effectifs :
  - Valeurs initiales des paramètres formels
  - Chaque valeur est liée au paramètre formel qui correspond à sa position (speed reçoit 20, time reçoit 60 * 60 et initialPosition ne reçoit rien).

Structures de contrôle

Structure de contrôles

- Les structure de contrôles influence l'ordre d'exécution des instructions.
- Elles permettent de :
  - Soumettre l'exécution à une condition (`if`, `switch`)
  - Répéter l'exécution un nombre de fois défini (`for`)
  - Répéter l'exécution tant qu'une condition est remplie (`while`)
  - Répéter l'exécution tant qu'une condition est remplie, mais au moins une fois (`do-while`)

Instruction et block d'instructions

- Les instructions de contrôle de flux s'applique généralement à une instruction unique qui est soit :
  - une instruction (expression, contrôle de flux)
  - un block d'instructions : `{ <instructions> }`
- Pour éviter des erreurs difficiles à diagnostiquer :
  - toujours utiliser un bloc d'instruction là où une instruction unique est attendue, même si le bloc ne contient qu'une instruction
- Exception :
  - instructions `if` dans un `else` (enchaînement de plusieurs `if`).

Instruction if-then-else


```
if (<expression booléenne>) {
    <instructions>
} else if (<expression booléenne>) {
    <instructions>
} else {
    <instructions>
}
```

- L'expression est implicitement convertie en booléen

Instruction switch


```
switch(<expression>) {
    case <expresion>:
        <instructions>
        break;
    case <expresion>:
        <instructions>
        break;
    default:
        <instructions>
        break;
}
```

- Sans l'instruction `break`, les instruction des cas qui se trouvent en dessous sont également exécutées.
- On peut également utiliser une instruction `return` ou `throw`.

Boucle définie


```
for(let i = 0; i < n; i += 1){
    <instructions>;
}
```

- Boucle définie : le nombre d'itérations est connu au moment de l'exécution de l'instruction
- La syntaxe ci-dessus n'est pas générale, mais correspond au cas d'utilisation le plus fréquent :
  - i est la variable de boucle
  - n est le nombre d'itérations

Boucle indéfinie


```
while(<expression booléenne>){
    <instructions>;
}
do {
    <instructions>;
} while(<expression booléenne>);
```

- Boucle indéfinie, test au début :
  - Répète l'exécution du bloc d'instruction tant que la valeur de l'expression booléenne est `true`.
- Boucle indéfinie, test à la fin :
  - Répète l'exécution du bloc d'instruction tant que la valeur de l'expression booléenne est `true`, mais au moins une fois.

Exception


```
try {
    <instructions>;
} catch (<error object>) {
    <instructions>;
} finaly {
    <instructions>;
}
```

- Si le bloc `try` se termine sans exception :
  - Exécute le .bloc `finaly`.
- Si une exception survient dans le bloc du `try` :
  - Exécute le bloc du `catch`.
  - Exécute l'instruction `finaly`.

Lancement d'une exception


```
throw new Error(<message>)
```

- Lancement d'une exception :
  - un bloc `try` a été ouvert : le flux d'instruction est interrompu et reprend dans le bloc catch correspondant
  - un bloc `try` n'a pas été ouvert : le programme est interrompu avec une erreur

Autres instructions de contrôle de flux

- Les instructions du tableau suivant servent également au contrôle de flux :

`return` <expression>	Retour de fonction.
<identifier>`:`	Définition d'un label.
`break` <label>$_{opt}$	Interrompt une instruction `switch` ou une boucle.
`continue` <label>$_{opt}$	Interrompt l'itération en cours d'une boucle.
`goto` <label>	Reprend l'exécution au label.

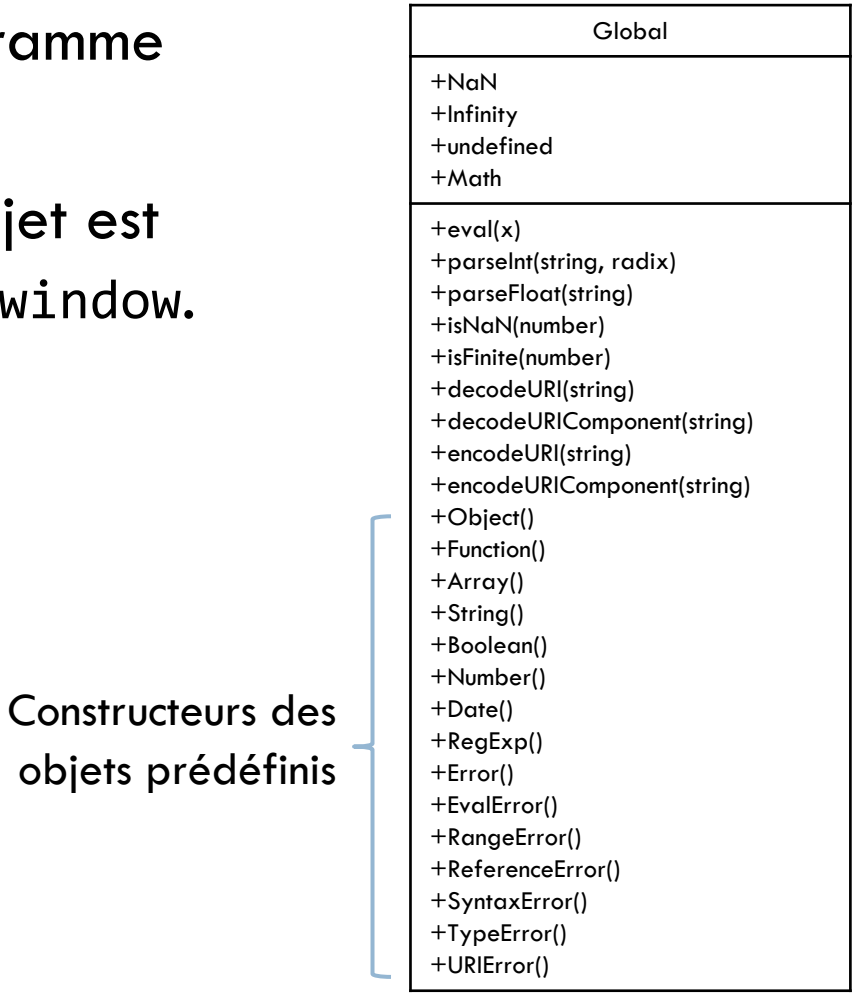
Utiliser des objets

Objets prédéfinis

- La norme ECMA-262 spécifie un certain nombre d'objet qu'une implémentation doit fournir et notamment :
  - Global
  - Math
  - Date
  - String
  - Array
  - Object

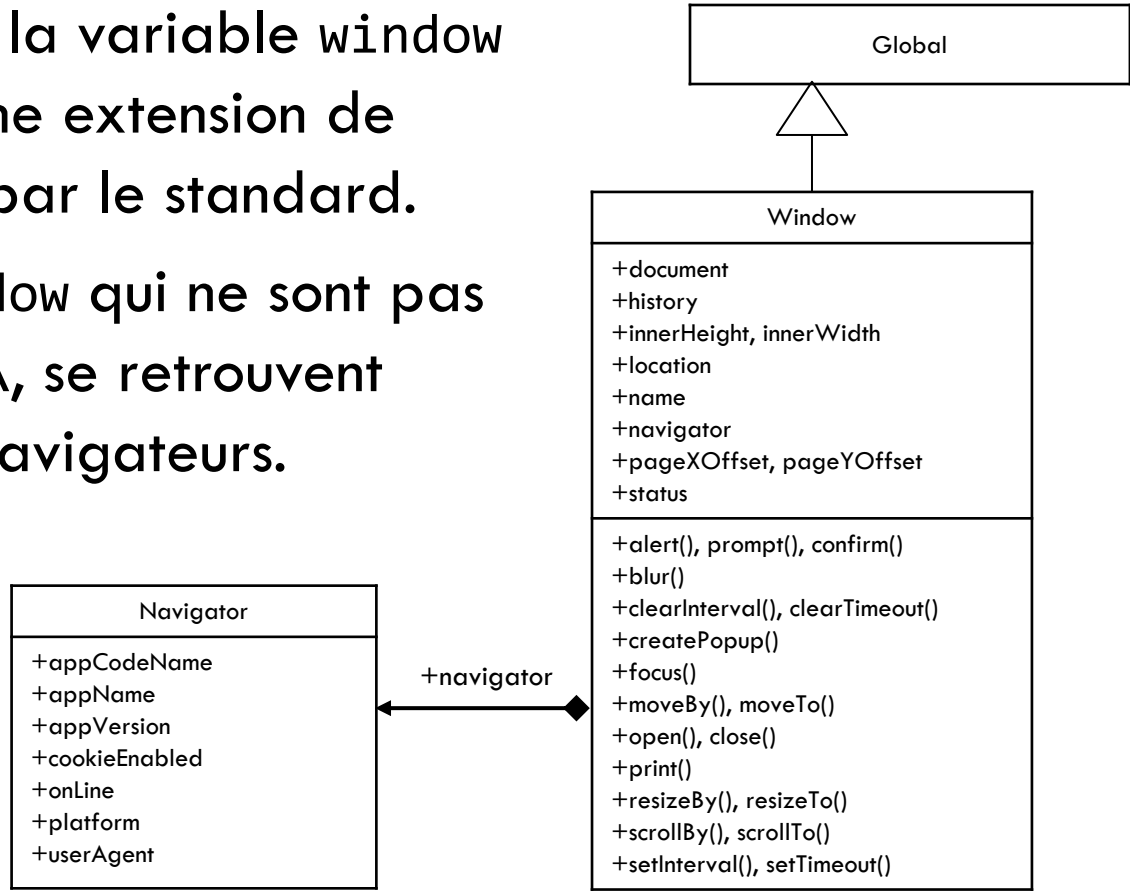
Objet Global

- Représente l'hôte du programme JavaScript.
- Dans un navigateur, cet objet est référencé par la variable `window`.



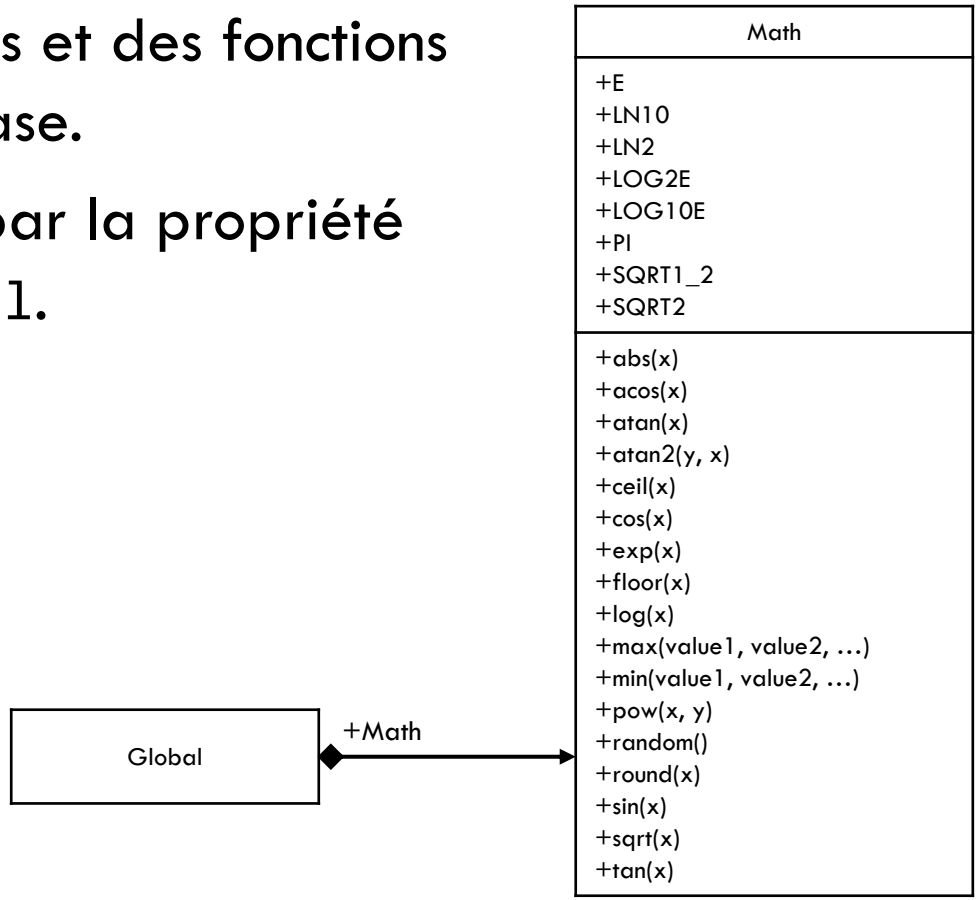
Objet Global – Navigateur

- L'objet référencé par la variable `window` d'un navigateur est une extension de l'objet `Global` défini par le standard.
- Les méthodes de `window` qui ne sont pas spécifiées par l'ECMA, se retrouvent dans la plupart des navigateurs.



Objet Math

- Expose des constantes et des fonctions mathématiques de base.
- Singleton référencé par la propriété `Math` de l'objet `Global`.



Objet Date

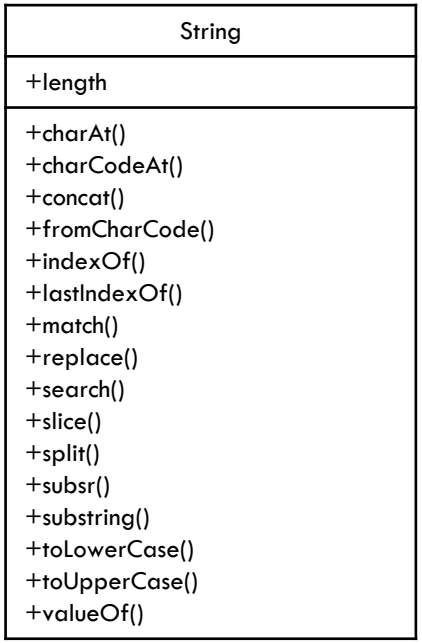
- Représente une date et heure locale.
- Nombre de millisecondes depuis le 1er janvier 1970 à 0h UTC.
- Exemple d'utilisation :

```
//crée un objet date à partir d'une
//chaîne au format ISO-8601 (date de
//naissance de Augusta Ada Byron)
const birthDate = new Date(
    Date.parse("1815-12-10T13:00Z"));
//écrit 10/12/1815 14:00:00
console.log(birthDate.toLocaleString())
```

Date
+getDate(), getUTCDate() +getDay(), getUTCDay() +getFullYear(), getUTCFullYear() +getHours(), getUTCHours() +getMillisenconds(), getUTCMilliseconds() +getMinutes(), getUCTMinutes() +getMonth(), getUTCMonth(); +getSeconds(), getUTCSeconds() +getTime() +getTimezoneOffset() +parse() +setDate(), setUTCDate() +setFullYear(), setUTCFullYear() +setHours(), setUTCHours() +setMilliseconds(), setUTCMilliseconds() +setMinutes, setUTCMinutes() +setMonth(), setUTCMonth() +setSeconds, setUTCSeconds() +setTime() +toDateString(), toLocaleDateString() +tolSOString() +toJSON() +toString(), toLocaleString() +toTimeString(), toLocalTimeString() +toUCTString() +UTC() +valueOf()

Objet String

- Représente une chaîne de caractères.
- Objet immuable (*immutable*) ⇒ les opération ne modifie pas les chaînes mais en crée de nouvelles.
- Les méthodes `match()`, `replace()` et `search()` délèguent la manipulation de la chaîne à un objet `RegExp`.



Objet Array (Tableau)

- Représente un ensemble de n valeurs indexées par un entier de 0 à n-1.
- Les valeurs peuvent avoir des types différents, mais l'ensemble devrait être homogène.
- Offre des méthodes pour simplifier la création de structures de données telles que piles (*stack*), queues, etc.

Array
+length
+concat() +indexOf() +join() +lastIndexOf() +pop() +push() +reverse() +shift() +slice() +sort() +splice() +toString() +unshift() +valueOf()

Objet Array – Création

- Deux manières de créer un tableau :
  - À l'aide d'un littéral (bonne pratique).

```
const list = []; // crée un tableau vide
const list = [10]; // crée un tableau d'une cellule
const list = ["zéro", "un"]; // crée un tableau de 2 cellules
```

  - À l'aide du constructeur (ambigu, à éviter).

```
const list = new Array(10); // 10 cellules vides
const list = new Array("zéro", "un"); // 2 cellules
```

Objet Array – Utilisation

- Utilisation de l'opérateur d'indexation :

```
const list = ["zéro", "un"]; // crée un tableau de 2 cellules
console.log(list[0]); // écrit "zéro" dans la console
list[1] = "une"; // replace la valeur de la cellule 1
```

- Structure dynamique :

```
console.log(list.length) // écrit 2 dans la console
list[9] = "neuf"; // étend le tableau de 2 à 10 cellules
console.log(list.length) // écrit 10 dans la console
list.push("dix"); // ajoute une valeur à la fin du tableau
console.log(list.length) // écrit 11 dans la console
element = list.pop(); // retire et renvoie la dernière valeur
console.log(list.length) // écrit 10 dans la console
```

- Fonctionne aussi avec une chaîne numérique :

```
console.log(list["0"]); // écrit "zéro" dans la console
list["2"] = "deux"; // replace la valeur de la cellule 2
console.log(list[2]); // écrit  "deux" dans la console
```

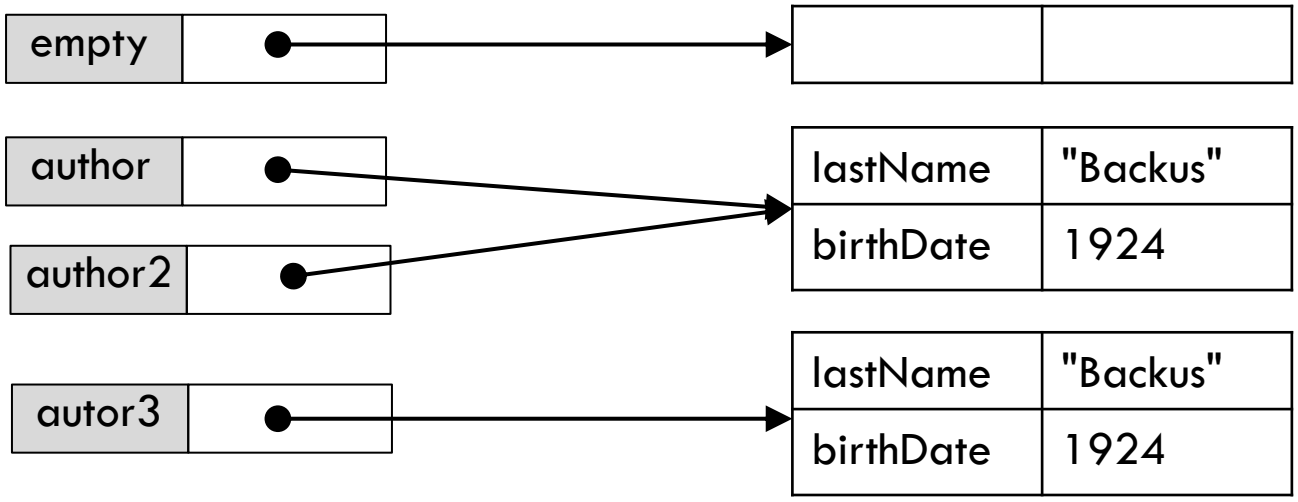
Objet Object

- En JavaScript, un objet est un ensemble hétérogène de couples noms-valeurs.
- C'est la structure de base de tous les objets, y compris les objets prédéfinis.
- La structure est dynamique : les couples noms-valeurs sont ajoutés ou supprimés au moment de l'exécution.

Objet Object – Création

- Création d'un obje à l'aide d'un littéral (bonne pratique) :

```
const empty = {};
const author = {lastName: "Backus", birthDate: 1924};
const author2 = author;
const author3 = {lastName: "Backus", birthDate: 1924};
```



Objet (Object)

- Opérateur d'accès aux propriétés :

```
const author = {lastName: "Backus", birthDate: 1924};
author.firstName = "John"; // crée la propriété firstName
const name = author.lastName; // renvoie "Backus"
```

- Opérateur d'indexation :

```
const author = {lastName: "Backus", birthDate: 1924};
author["firstName"] = "John"; // crée la propriété firstName
const name = author["lastName"]; // renvoie "Backus"
```

Object, Array et tableau associatif

- En JavaScript tous les objets sont des tableaux associatifs (valeurs indexées par un nom).
- Mais un objet `Array` n'est pas un tableau associatif. La propriété `length` ne fonctionne que si les index sont des entiers ou des chaînes numériques.

```
const list = [];
console.log(list.length); // écrit 0 dans la console
list[0] = "zéro";
console.log(list.length); // écrit 1 dans la console
list["test"] = "onze"; // ajoute la propriété "test" à l'objet
console.log(list.length); // écrit toujours 1 dans la console
console.log(list.test); // éccrit onze dans la console
```

Entités de première classe

- Un élément du langage est une entité de première classe (*first class citizen*) si et seulement si :
  - elle peut être affectée à une variable ou à une propriété
  - elle peut être passée en paramètre à une fonction
  - elle peut être renvoyée par une fonction
  - elle peut être créée dynamiquement
- Exemples : expressions
- Contre-exemples : opérateurs, structures de contrôle

Objet Function


```
const func = function (<paramètres formels>) {
    <instructions>
};
```

- La partie gauche de l'affectation est une expression dont la valeur est un objet de type Function.
- func est une référence à un objet qui peut être :
  - affectée à une autre variable
  - passée en paramètre à une fonction
  - utilisée comme valeur de retour d'un fonction
- Une fonction est donc une entité de première classe
- Une expression de type Fonction est une expression lambda.

Fonction de première classe – Exemple

- La méthode sort() de l'objet Array prend comme arguments une fonction qui prend reçois deux valeurs et qui doit :
  - Renvoyer une valeur > 0 si valeur 1 > valeur 2.
  - Renvoyer une valeur < 0 si valeur 1 < valeur 2.
  - Renvoyer 0 si valeur 1 = valeur 2.

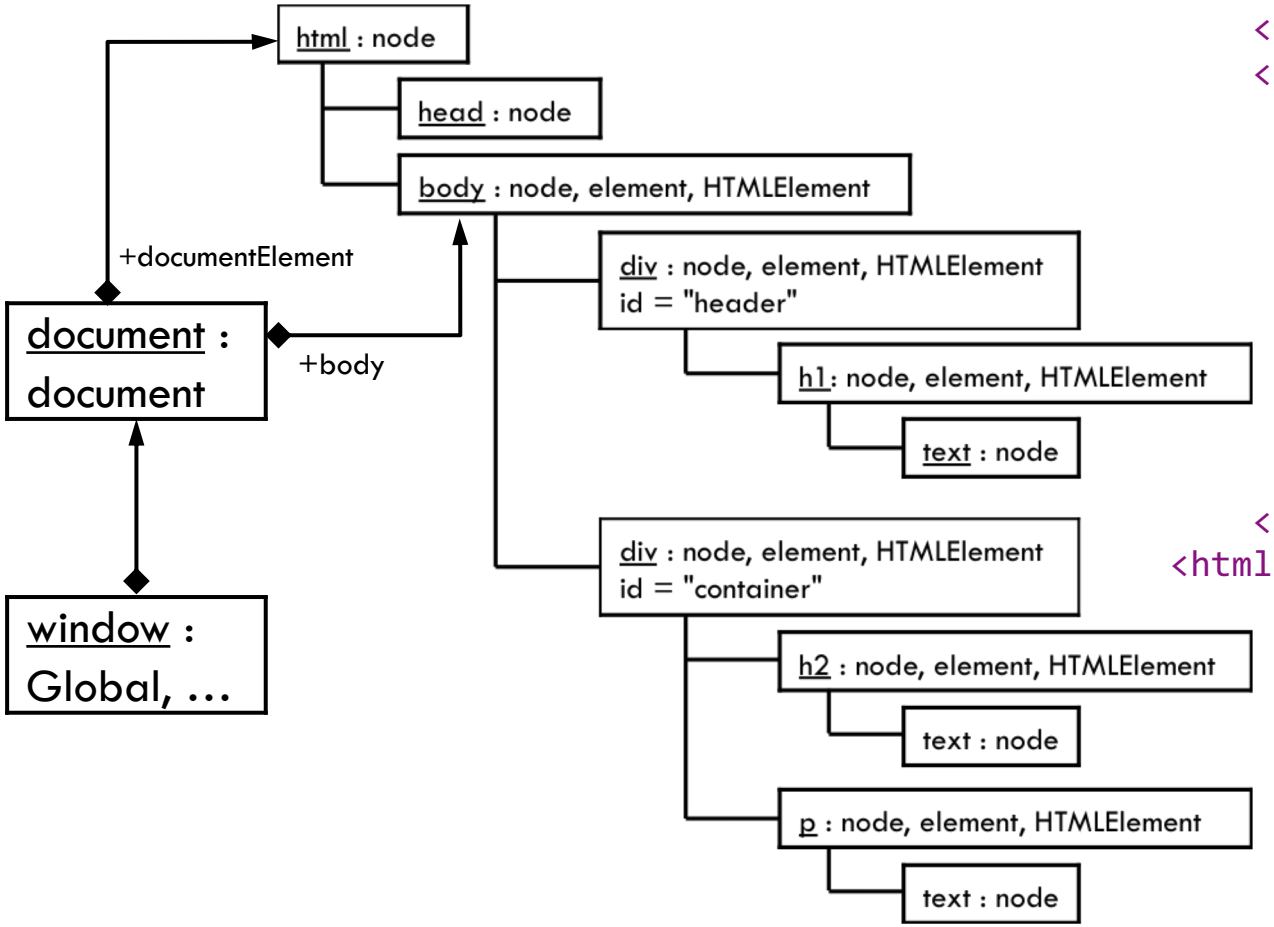
```
const list = [22, 33, 12, 34, 5, 42];
list.sort(function(val1, val2) {
    return val1 - val2;
});
console.log(list.join(","));
```

Document Objet Model (DOM)

JavaScript et HTML

- Lorsque JavaScript est utilisé dans un navigateur Web, il sert à étendre les fonctionnalités d'un document HTML.
- Pour cela, le document HTML est présenté au programme JavaScript sous la forme d'une hiérarchie d'objets : le DOM ou Document Object Model.
- Le DOM est spécifié par la W3C et ne fait pas partie de la spécification du langage (ECMA-262)
- La racine de la structure d'objet est la propriété `document` de la variable globale `window`.

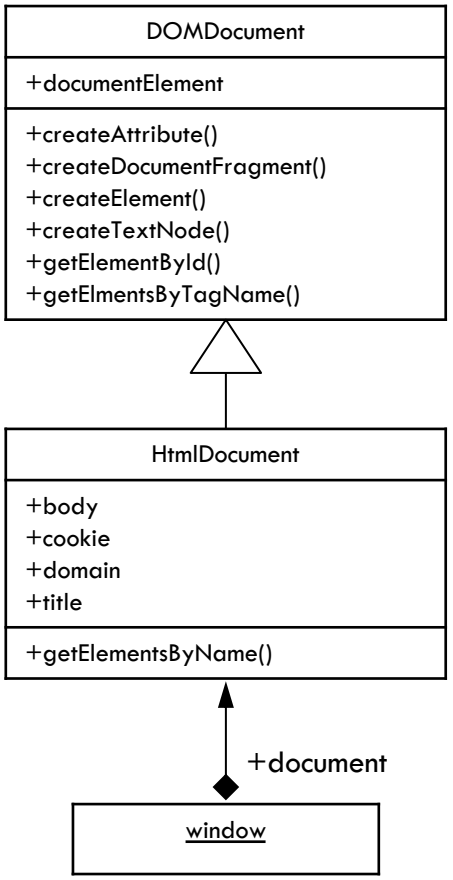
Structure du document




```
<html>
    <head>
    </head>
    <body>
        <div id="header">
            <h1>Titre 1</h1>
        </div>
        <div id="container">
            <h2>Titre 2</h2>
            <p>
                paragraphe1.
            </p>
        </div>
    </body>
<html>
```

Objet Document

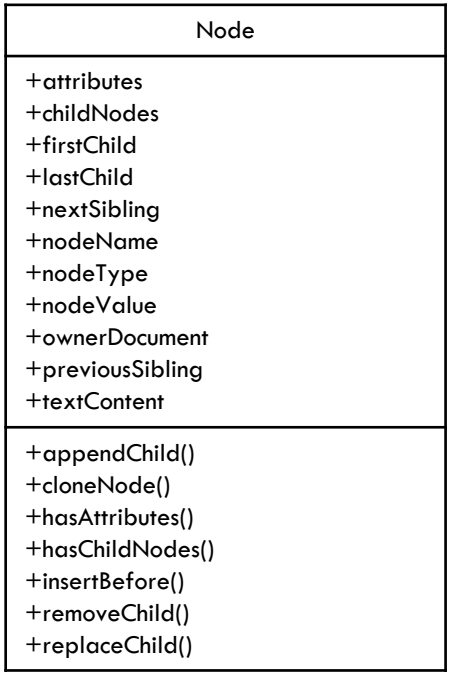
- Représente le document HTML.
- Permet de créer les éléments.
- Permet de rechercher les éléments :
  - Par leur identification (attribut id).
  - Par leur nom (attribut name).
  - Par le nom de leur balise.



Objet Node

- Représente un nœud du document HTML.
- Permet de modifier la structure de l'arbre.
- Permet p. ex. de parcourir l'arbre :

```
function visitNode(node, visitorFunc) {
    visitorFunc(node);
    const node = node.firstChild;
    while (node) {
        visitNode(node, func);
        node = node.nextSibling;
    }
}
```



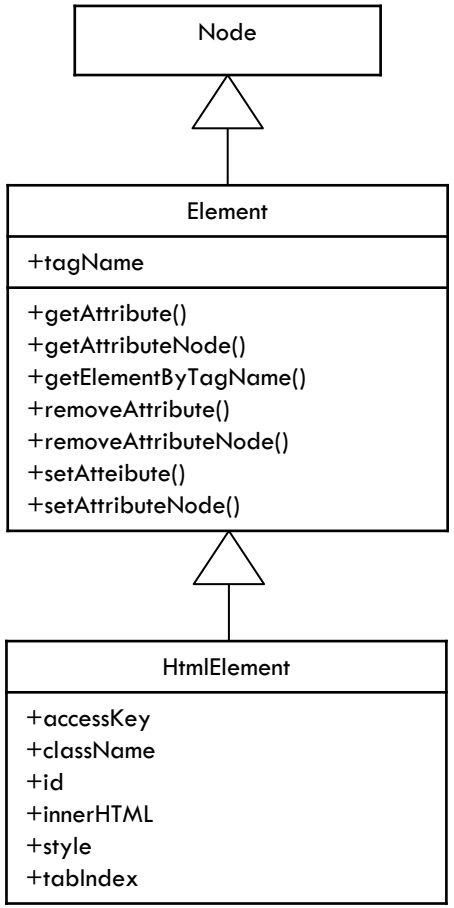
Objet Node – Exemple

- Exemple d'utilisation de visitNode :

```
function getElementsByClassName(root, className) {
    const elements = [];
    visitNode(root, function (node) {
        const className = visitedNode.className;
        if (className) {
            const classes = className.split(" ");
            for (const i = 0; i < classes.length; i += 1) {
                elements.push(node);
                break;
            }
        }
    });
    return elements;
}
```

Objet HTMLElement

- Représente un élément HTML.
- Permet de lire ou de modifier :
  - Le contenu
  - La classe
  - Le style
  - Les attributs



Propriétés d'événement

DÉF. : Les objets du DOM ont un certain nombre de propriétés auxquelles on peut affecter une fonction appelée gestionnaire d'événements (*event handler*). Cette fonction est ensuite invoquée par le navigateur en réponse à un événement du système ou à une action de l'utilisateur (click de souris).

- Exemple :

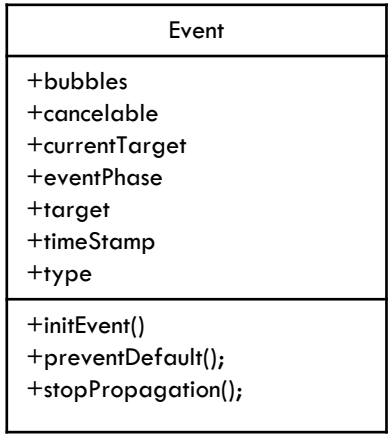
```
window.onload = function (e) {
   // le paramètre e contient une référence à un objet
   // de type Event qui représente l'événement.
   console.log("Le document chargé!");
   console.log(e);
}
```

addEventListener

- Affecter directement un gestionnaire d'événements à une propriété d'événement n'est pas une bonne pratique (risque d'écraser un gestionnaire déjà présent).
- La méthode `addEventListener` permet d'éviter ce problème. Elle maintient une liste de gestionnaires pour chaque événement. Lorsqu'un événement survient, les gestionnaires sont appelés l'un après l'autre.
- L'ordre d'appel des gestionnaires n'est pas défini.

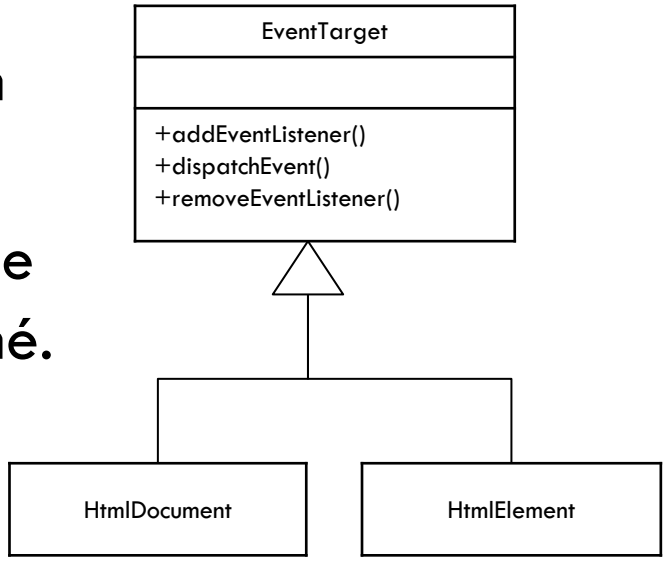
Objet Event

- Représente un événement.
- Passé en paramètre à un gestionnaire d'événement.
- La méthode `preventDefault()` permet d'annuler l'événement :
  - Utilisé par exemple pour annuler l'envoi des données d'un formulaire (`onsubmit`).



Objet EventTarget

- Représente une cible pour un événement événement (souvent un widget).
- Permet d'attacher ou de supprimer un gestionnaire d'événement.
- Permet d'invoquer tous les gestionnaire d'événement pour un événement donné.



Interfaces d'événements

Mouse Events
+onclick +ondblclick +onmousedown +onmousemove +onmouseout +onmouseup

Form Events
+onblur +onchange +onfocus +onreset +onselect +onsubmit

Keyboard Events
+onkeydown +onkeypress +onkeyup

Window Events
+onabort +onerror +onload +onresize +onscroll +onunload