JavaScript (JS)

Two types of programming on web

1. Client Side Web Scripting
2. Server Side Web Scripting

Alternatives

-Applets

-ActiveX

<script>

* Linked a.k.a external scripts
* execution of linked scripts can be deferred or run asynchronously

\*Problem

-before rendering of page, it have to fetch the script.js

* Embedded a.k.a internal scripts
* top-level code (i.e. code outside functions) execute as the script is loaded
* code inside functions are executed only when the function is involved

-preferred way

-anywhere

Exclusives

-defer attribute

-async attribute

* Inline scripts
* scripts may be embedded anywhere in the document and any top-level code is executed where it is concentered

<noscript>

* provides alternative content when the browser does not support or disabled
* JavaScript Execution Environment

-accessible for specific page only

-JavaScript was built on NetScape

* Core JavaScript (EcmaScript) language and API
* (Traditional) Browser Object Model (BOM) API (non-standardized)

- Window, Navigator, Screen, Location, History

* Document Object Model (DOM) API

- Node, Document, Element, Text, Attr, DocumentType, Comment, etc.

- Event, EventTarget, Event Listener, etc.

- CSSStyleSheet, CSSRule, etc.

* Miscellaneous JavaScript Web APIs
* Geolocation, IndexedDB, Local Storage, Push Notifications, Service Workers, Web Sockets, Web Workers, XML HttpRequest (AJAX), etc.
* Most of the APIs are accessed from scripts via the global object **Window**
* DOM (Document Object Model)
  + API that allows access to the HTML document from within scripts associated with the web page
    - A parsed HTML document is represented by a **DOM tree**, which contain **nodes** representing **elements**, **element attributes**, **textual content**, and other HTML document components (e.g DOCTYPE, comments)
    - accessed in the Document object property of the global Window object

\*the way they look in real time

\*most important

Functionalities:

* retrieve references to nodes (or node collections) in the DOM
  + getElementById()
  + getElementByTagName, getElementByClassName(), getElementByName()
  + querySelector(), querySelectorAll()
* traverse the DOM tree (from a given node)
* parentNode, parentElement
* childNodes, children
* firstChild, lastChild, nextSibling, previousSibling
* firstElementChild, lastElementChild, nextElementSibling, previousElementSibling
* construct/copy nodes
  + createElement(), createTextNode(), createAttribute()

JavaScript Language Fundamentals

Top Level

var globalVar; //global scope – property of window object

let someVar; //global scope – not a property of window object

const LEGAL\_AGE=21; //global constant- not a property of window object

avoidThis = 100; //global reference to window property

Inside Function

function func() {

var localVar1; //local function scope

avoidThisToo=0; //reference to window property

for (;;) lexical scoping technique

var localVar2; //local function scope, but only from this point

let blockVar; //block local scope

}

Primitive data type

Numbers

* decimal notation (integer, floating-point, scientific notation)
* binary, octal, hexadecimal
* special numbers ( Infinity, -Infinity, NaN)

Strings

* single-quote delimited
* double-quote delimited
* back-quote
  + template literals
    - allowing multi-line strings and expression interpolation

Boolean

* true, false
* 0, “”(empty string), null, undefined, NaN are “falsy” values

Null and undefined

* Var object= null;
* Var und1, und2 = undefined;

Object type

* New String
* New Number
* New Boolean
* Dates and times (in the browser environment), date object
* Regular expressions
  + RegExp Object
  + Var r1 = new regExp(‘[,.],’g’);
    - Object constructor
  + Var r2 = /[/:]/g;
    - Regular expression literal syntax
* Mathematical functions and constants (math objects)
  + Sqrt
  + Pi
  + Random
  + Trunc

JavaScript is loosely and dynamically-typed

* var var1 = 100; //v1 is a number
* var var1=’hello’; //v1 is now a string
* var varr1=true; //v1 is now a Boolean

equality and identity



implicit type conversion

(8+’9’) //’89’

(8\*’9’) //72

(8/false) //Infinity

(8+true) //9

(8-‘nine’) //NaN

(8+null) //8

(8+undefined) //NaN

(true+’love’) //’truelove’

Object functions for explicit type conversions

(8+Number(‘9’)); //17

(String(8) + String (true)); //’8true’

FUNCTIONS

function max( item 1, item 2) {

if (item 1 > item 2) {

return item1;

} else if (item 2 > item 1) {

return item2;

}

}

\*if item 1 = item 2, it returns undefined because there is no return if item 1 is equal to item 2

//’arguments’ object (only valid inside a function)

//default parameter values

//”rest” parameters

* (arg1, arg2, … other Args)

//function expressions

* Function (a,b)

//”arrow” function expression

* Var f5 ( a,b) = > {ar P-a\*b; return P; }

//functions are “first-class objects”

* Higher order function
  + Function that returns a function
* Can be passed as arguments to functions
* Nesting functions
  + Define functions inside a function

ARRAYS

* Array size is dynamic
* JavaScript allows “sparse “arrays
  + Array with no elements in other locations
* Length is higher than 1 than the index
* Can declare different types in an array (flexible)

**JavaScript Event Handling**

Onclick Event Type

-most used type of event that triggers when a user clicks using the left button of mouse.

Onsubmit Event Type

-event that triggers when a user tries to submit a form.

Onmouseover Event Type

-event that occurs when a user brings the mouse over any element

Onmouseout Event Type

-event that occurs when a user move the mouse out from the element.

How to Create and Dispatch Events

Var clickEvent = new Event(‘click’);

Var button = document.querySelector(‘#button’);

Button.dispatchEvent(clickEvent);

Var ce = new CustomEvent(‘pindot’);

Button.addEventListener(‘pindot’,pindotHandler);

Button.dispatchEvent(ce);

Event Propagation Order (DOM-compliant browsers)

* Capturing the phase
* At target
* Bubbling phase

Event Handler Registration

Method 1 (inline HTML event attribute)

* Non-standard, universal support, not recommended
* Ex. Onclick=’handler()’

Method 2 (pre-DOM L2)

* Non-standard, universal support
* Can register only one handler
* Remove handler by assigning null value to attribute
* Ex. button.onclick = handler();

Method 3(DOM L2 event registration)

* Standard, well-supported
* Can register more than one event handler
* Can register handlers on the bubble or capture phase
* Use removeEventListener() to remove event handler
* Use dispatchEvent(0 to trigger event programmatically
* Ex. button.addEventListener (‘click’, handler)

Method 4 (Microsoft- specific, non-standard, IEv6-v10…)

* Can register more than one event handler
* Can register handlers only on the bubble phase
* Use detachEvent() to remove event handler
* Use fireEvent() to trigger event programmatically
* Ex. button.attachEvent(‘onclick’, handler)

//cross-browser event registration

Var button – document.getElementById(‘button’);

If (button.addEventListener) {

Button.addEventListener(‘click’,handler);

} else if (button.attachEvent) {

Button.attachEvent(‘onclick’, handler);

} else {

Button.onclick = handler;

}

\*Whenever an event is triggered information about the event is made available via the ‘event’ object property of the ‘Window’ object

\*for method 1 registration, the event object may be passed explicitly in the call to the handler

\*for methods 2 and 3 registration, the event object is passed implicitly as the first argument of the event handler.

Event Object Properties and Methods

* Bubbles
* Eventphase
* Target (or srcElement (non-standard))
* currentTarget
* PreventDefault()
* stopPropagation ()
* stopImmediatePropagation()
* other events-specific information

AJAX (Asynchronous JavaScript And XML)

XMLHttpRequest

* var xar = new XMLHttpRequest();
* xhr.open(‘GET’, resource, false);
  + what resource will be fetched
* xhr.send()

\*JavaScript is inherited single-threaded.

2 modes of fetch

* synchronous
* asynchronous

Synchronous

-page is blocked, cannot interact, false

Asynchronous

-not frozen, can interact, true(default)

Callback

-monitor progress of response

JavaScript Promise

//defining the promise

Var Promise = new Promise(function(resolve,reject) {

//do operation (asynchronously done)

//success - > call resolve () …

//failure -> call reject () …

} )

//calls the promise

Promise().then(success,failure);

Function success(result) {}

Function failure(err) {} markusjirou18p