Client-side Technologies

Dr. Niveen Nasr El-Den
iTi

Day 8

JavaScript Fundamentals cont.

Event Object

Event Object

- The event object gives information about an event that has occurred.
- When an event occurs, an **event** object is initialized automatically and passed to the event handlers.
- We can create event object via its constructor

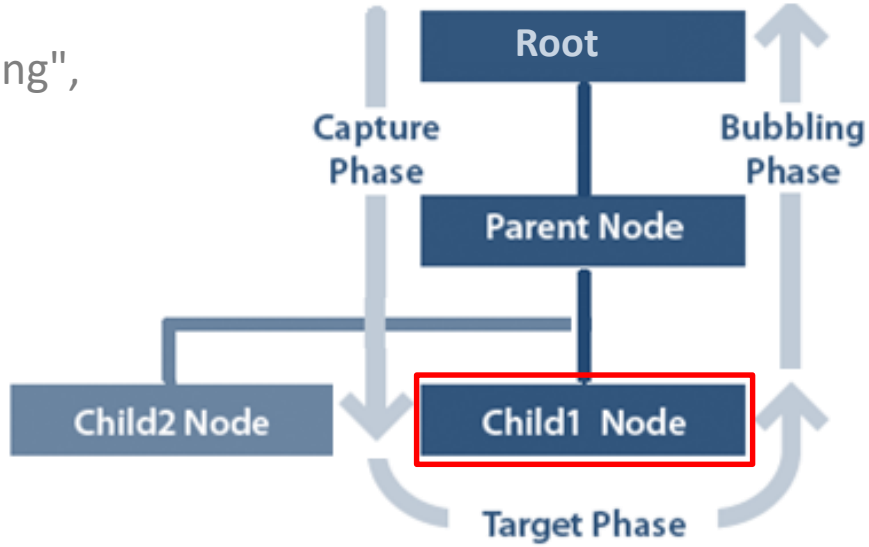
  var evt= new Event()

- The Event object represents the state of an event, such as the element in which the event occurred, the state of the keyboard keys, the location of the mouse, and the state of the mouse buttons.
- **Object Model reference:**

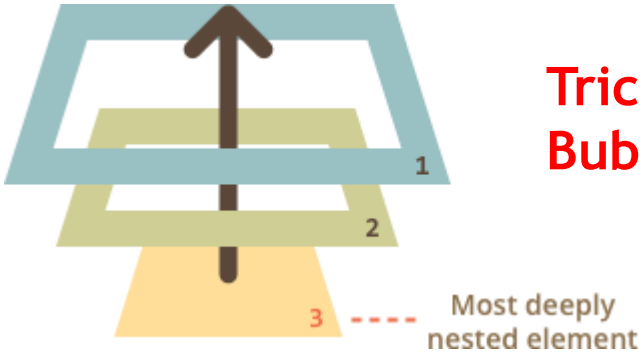
  [window.]event

Event Object

- Events always propagate from the root
- When an event occurs, it is dispatched to the target element first.
- 2 ways for objects to handle fired events
  - Event Capture (Phase1)
    - Capturing is also called "trickling",
    - Event goes down,
  - Event Bubbling (Phase3)
    - Event goes up



Event Object



Trickle down,
Bubble up

- If the event propagates up, then it will be dispatched to the ancestor elements of the target element in the DOM hierarchy.

- The propagation can be stopped with the stopPropagation() method and/or the cancelBubble property.

Event Object Properties

Event Object Property	Description
srcElement	The element that fired the event
target	
currentTarget	identifies the current target for the event, as the event traverses the DOM
type	String representing the type of event.
clientX (layerX)	Mouse pointer X coordinate at the time of the event occurs relative to upper-left corner of the window.
clientY (layerY)	Mouse pointer Y coordinate at the time of the event occurs relative to upper-left corner of the window.
offsetX	Mouse pointer X coordinate relative to element that fired the event.
offsetY	Mouse pointer Y coordinate relative to element that fired the event.

Event Object Properties

Event Object Property	Description
altKey	True if the alt key was also pressed
ctrlKey	True if the alt key was also pressed
shiftKey	True if the alt key was also pressed
button	Any mouse buttons that are pressed
keyCode(deprecated)	Returns UniCode value of key pressed use code property instead
which(deprecated)	
key	Represents the value of the key pressed (e.g. n)
code	Represents a physical key on the keyboard (e.g. keyN)

For alt,ctrl,shft keys

- keypress event don't fire when any of them is pressed
- Their properties is set to true only on onkeydown event

event.button value	Description
1	Left Mouse Button
2	Right Mouse Button
4	Middle Mouse Button

Example!

Event Object Properties

Event Object Property	Description
eventPhase	Any mouse buttons that are pressed
cancelBubble (deprecated)	Can cancel an event bubble

event.eventPhase value	Constant Property	Description
0	Event.NONE	No event is being processed at this time.
1	Event.CAPTURING_PHASE	The event is being propagated through the target's ancestor objects
2	Event.AT_TARGET	The event has arrived at target
3	Event.BUBBLING_PHASE	The event is propagating back up through the target's ancestors in reverse order

Example!

Event Object Methods

Methods	Description
event.stopPropagation()	Disables the propagation of the current event in the DOM hierarchy. (IE8 = cancelBubble)
event.stopImmediatePropagation()	prevents other listeners are attached to the same element for the same event fr om being called, no remaining listeners will be called.
event.preventDefault()	To cancel the event if it is cancelable, meaning that any default action normally taken by the implementation as a result of the event will not occur. (IE8 = returnValue)
event.composedPath()	Returns the event's path

Other Useful Methods

Methods	Description
elem.addEventListener()	Registers an event handler function for the specified event on the current object.
elem.removeEventListener()	method to remove an event listener that has been registered with the addEventListener method.
elem.dispatchEvent()	Initializes an event object created by the Event Constructor

Synthetic Events

- To create custom event use Event constructor

  var myEvent= new Event(p1,p2)

  - p1: the name of the custom event type
  - p2: an object with the following Optional properties with false as default value
    - bubbles: indicating whether the event bubbles.
    - cancelable: indicating whether the event can be canceled.
    - ~~composed: indicating whether the event will trigger listeners outside of a shadow root.~~

- To fire the event programmatically use dispatchEvent() on a specific element

  elem.dispatchEvent(myEvent)

Synthetic Events

- To create custom event use CustomEvent constructor

  var evt= new CustomEvent(p1,p2)

  - p1: the name of the custom event type
  - p2: is object with details property to add more data to the event object

- To fire the event programmatically use dispatchEvent() on the element registering the event

  elem.dispatchEvent(evt)

JavaScript Cookies

Cookies

- Cookies are small text strings that you can store on the computers of people that visit your Web site.
- Cookies were originally invented by Netscape to give ‘memory‘ to web servers and browsers.
- Normally, cookies are simple variables set by the server in the browser and returned to the server every time the browser accesses a page on the same server.
- A cookie is not a script, it is a mechanism of the HTTP server accessible by both the client and the server.

Need Of Cookies

- HTTP is a state-less protocol; which means that once the server has sent a page to a browser requesting it, it doesn't remember any thing about it.

- The HTTP protocol, is responsible for arranging:
  - Browser requests for pages to servers.
  - The transfer of web pages to your browser.

Need Of Cookies

- *Stateless protocols* have the advantage that they require fewer resources on the server
  -- the resources are pushed into the client.

- But the disadvantage is that the client needs to tell the server enough information on each request to be able to get the proper answer.

- As soon as personalization was invented, this became a major problem.

- **Cookies** were invented to solve this problem.

HTTP cookie

=

Web cookie

=

Browser cookie

Cookies

- ***Cookies*** are a method for a server to ask the client to store arbitrary data for use in future connections.
- They are typically used to carry persistent information from page to page through a user session or to remember data between user sessions.
- With JavaScript, you can create and read cookies in the client-side without resorting to any server-side programming.
- A cookie may be written and accessed by a script but the cookies themselves are simply passive text strings.

Types Of Cookies

- Cookies has two types:
  - Session Cookies/ Non-persistent : These cookies reside on the Web browser and have *no expiry date*. They expire as soon as the visitor closes the Web browser.
  - Persistent Cookies: These cookies have an *expiry date*, are stored on a visitor's hard drive and are read by the visitor's browser each time the visitor visits the Web
    site that sent the cookie

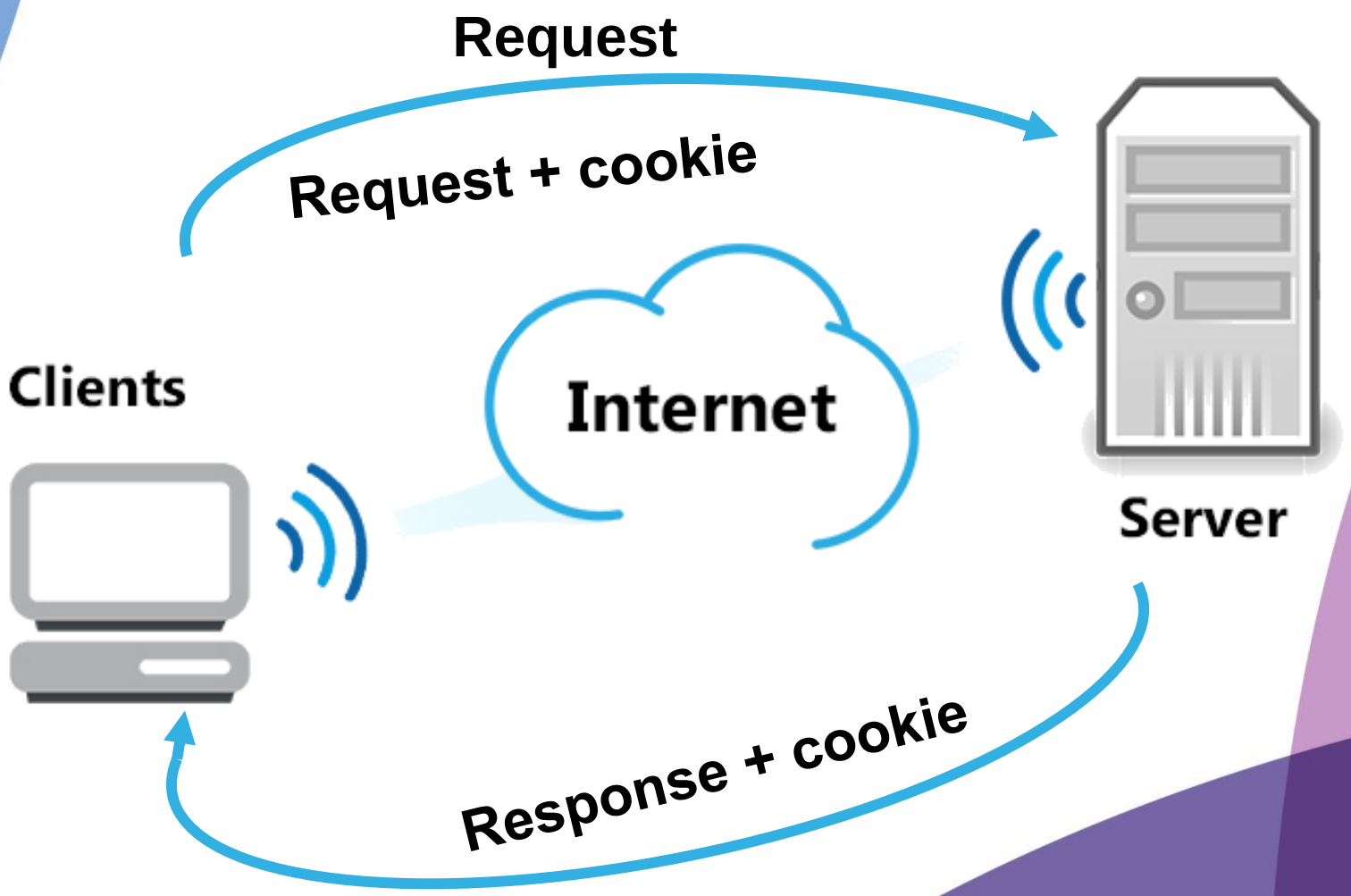
Benefits of Cookies

Session Management

- **Authentication**
  - no longer need to enter password
  - Greeting people by name.
  - **Saving time for returning visitors**
    - The user does not have to re-enter information
- **Research websites.**
- **Maintaining state**
  - Adventure games that use cookies to keep track of pertinent character data and the current state of the game.
- **Shopping carts**
  - By storing data as you move from one page (or frame) to another.

- User preferences, themes, and other settings → **Personalization**

- Tracking Recording and analyzing user behavior


Request
Request + cookie
Clients
Internet
Server
Response + cookie

Cookies Limitations

- All Browsers are preprogrammed to allow a total of 300 Cookies, after which automatic deletion based on expiry date and usage.
- Each individual domain name (site) can store 20 cookies.
- Each cookie having a maximum size of 4KB.

Cookies Facts

- A server can set, or deposit, a cookie only if a user visits that particular site.
  - i.e. one domain cannot deposit a cookie for another, and cross-domain posting is not possible.
- A server can retrieve only those cookies it has deposited.
  - i.e. one server cannot retrieve a cookie set by another.
- Cookies can be retrieved only by the Web site that created them. Therefore any cookie you create is safe from view of other Web sites.
- Cookies are sent with every request, so they can worsen performance (especially for mobile data connections).

Cookies Securing Facts

- Highly unreliable, from a programming perspective.
  - It's like having your data stored on a hard drive that sometimes will be missing, corrupt, or missing the data you expected.
- Cookie security is such that only the originating domain can ever use the contents of your cookie "*Same-origin policy*".
- Cookies just identify the computer being used, not the individual using the computer.
- Cookie files stored on the client computer are easily read by any word processing program, text editor or web browsing software unless an encryption mechanism is applied.

Cookies False Claims

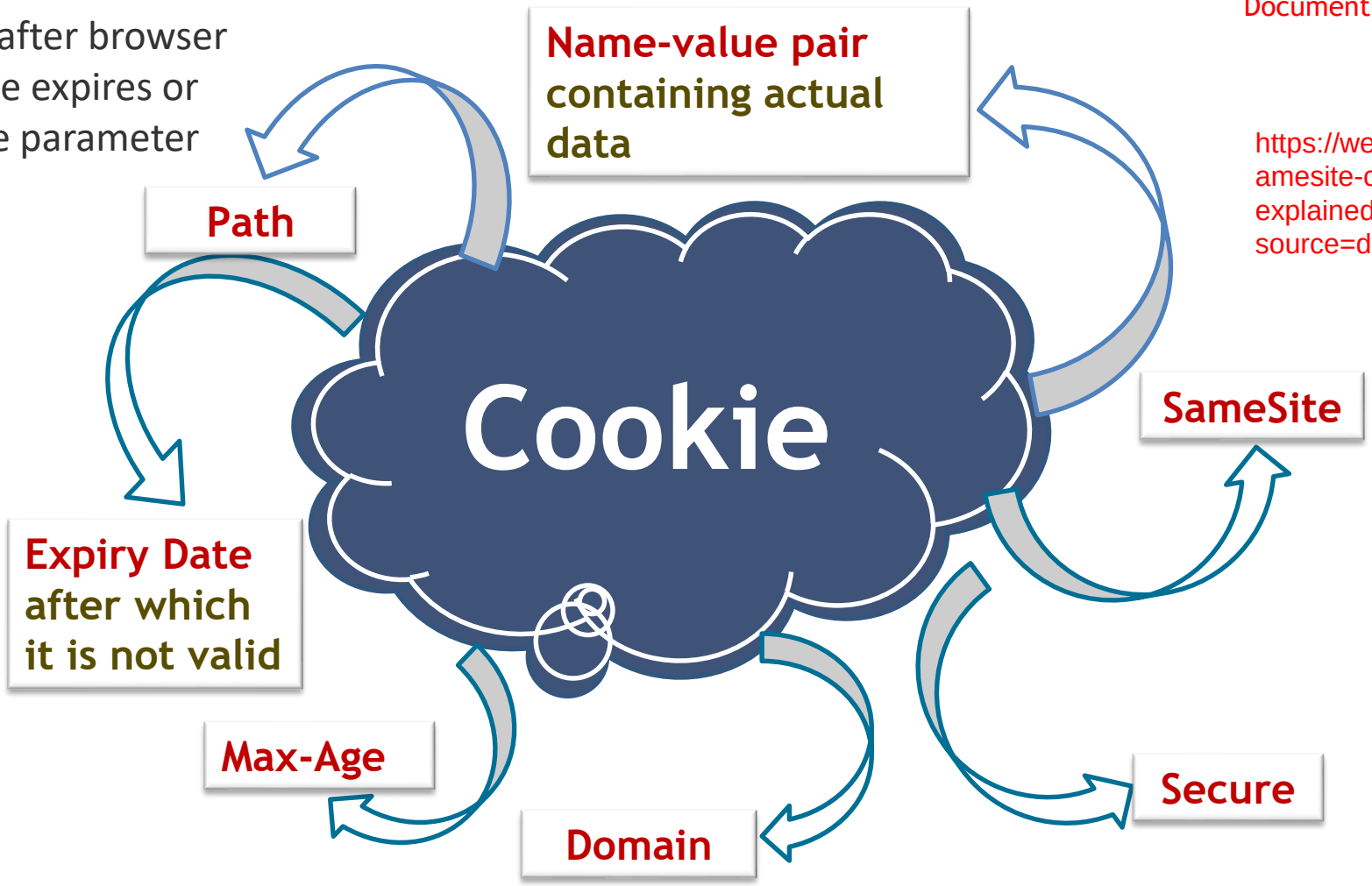
- Cookies are like **worms** and **viruses** in that they can erase data from the user's hard disks
- Cookies generate **popups**
- Cookies are used for **spamming**
- Cookies are only used for **advertising**

Cookies Parameters

https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

To let cookies survive after browser close use expires or max-age parameter

https://web.dev/samesite-cookies-explained/?utm_source=devtools



Cookies Parameters

https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

Parameter	Description	Example
name=value	This sets both cookies name and its value. The cookie value string can use encodeURIComponent() to ensure that the string is in a valid format and does not contain any disallowed char in cookie values e.g. commas, semicolons, or whitespace.	username=JavaScript
expires=date	This optional value set the date that the cookie will expire on. The date should be in the GMT format. If the expires value is not given, the cookie will be destroyed the moment the browser is closed	expires= today.toUTCString()
max-age=sec	Similar to expires but is a number of seconds till the cookie disappears. It has priority over expires If neither expires nor max-age specified it will expire at the end of session.	max-age=60*60*60*5 //5 hours

Working with Cookies

- Cookies can be *created*, *read* and *deleted* by JavaScript, under these conditions:
  1. The user's navigator must be cookie-enabled. This can be checked using "*navigator.cookieEnabled*" property .
  2. The cookie(s) that you set or accept are only accessible at pages with a *matching domain name, matching path.*
  3. The cookies must not have reached or passed their expiry date.

- When these criteria are met the cookies become available to JavaScript via the *document.cookie* property.

Creating a Cookie

- Assigning a value to the *document.cookie* property

document.cookie="name=value";

document.cookie="name=value;expires=date";

```
<head>
 <script language=”JavaScript”>
  document.cookie =  ”myCookie =“ +
                 encodeURIComponent(“This is my Cookie”);
  window.alert(“myCookie=” +
               encodeURIComponent(“This is my Cookie”));
 </script>
</head>
```

Creating a Cookie

- Assigning a value to the *document.cookie* property

document.cookie="name=value;expires=date"

```
<head>
  <script language=”JavaScript”>
    var myDate = new Date();
    document.cookie = “myCookie=” +
                        encodeURIComponent(“This is my Cookie”) +
                                “;expires=” + myDate.toGMTString();
  </script>
</head>
```

Displaying a Cookie

- **Retrieve created Cookie value**
  - Extract the name and value of the cookie to two variables.
  - The document.cookie will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value
  - We use strings' *split()* function to break the string into key and values.

```
<head>
    <script language="JavaScript">
        var newCookie = document.cookie;
        var cookieParts = newCookie.split("=");
        var cookieName = cookieParts[0];
        var cookieValue = decodeURIComponent(cookieParts[1]);
        window.alert(cookieName);
        window.alert(cookieValue);
    </script>
</head>
```

Clearing a Cookie

- If the user logs out or explicitly asks not to save his or her username in a cookie, hence, you need to delete a cookie to remove a username cookie.

- Simply reassign the cookie, but set the expiration date to a time has already passed.

```
<head>
 <script language=”JavaScript”>
  var newDate = new Date();
  newDate.setTime(newDate.getTime() - 86400000);
  document.cookie = “myCookie=;expires=“+ newDate.toUTCString();
 </script>
</head>
```

Multiple Cookies

- Most Web browsers set limits on the number of cookies or the total number of bytes that can be consumed by the cookies from one site.
- **Creating Multiple cookies**
  - Assign each cookie in turn to the document.cookie object and ensure that each cookie has a different name, and may have a different expiration date and time.
- **Accessing Multiple Cookies**
  - more complicated since accessing document.cookie,there will be a series of cookies separated by semicolons;

CookieName=firstCookieValue;secondCookieName=secondCookieValue;etc.

When we **update** or **delete** a cookie, we should use exactly the same **path** and **domain** options as when we set it.

Creating a Cookie Function Library

- Working with cookies requires a lot of string and date manipulation, especially when accessing existing cookies when multiple cookies have been set.
- To address this, you should create a small cookie function library for yourself that can:
  - create
  - access
  - delete

  } cookies

  without needing to rewrite the code to do this every time.

Creating a Cookie Function Library

- **getCookie (cookieName)**

  Retrieves a cookie value based on a cookie name.
- **setCookie (cookieName,cookieValue[,expiryDate])**

  Sets a cookie based on a cookie name, cookie value, and expiration date.
- **deleteCookie (cookieName):**

  Deletes a cookie based on a cookie name.
- **allCookieList ():**

  returns a list of all stored cookies
- **hasCookie (cookieName)**

  Check whether a cookie exists or not

Cookies
were once used for general client-side storage.
Now it is recommended to use Modern Storage APIs.
Web Storage API
&
IndexedDB.

Cookie

is a small piece of data that a server sends to the user's web browser.

The browser may store it and send it back with later requests to the same server.

Cookie

is used to tell if two requests came from the same browser keeping a user logged-in etc..

Assignments

Error Object

(Built-in Object)

JavaScript Built-in Objects

- String
- Number
- Array
- Date
- Math
- Boolean
- RegExp
- Error
- Function
- Object

Error Object Creation

- Whenever an error occurs, an instance of error object is created to describe the error.
- Error objects are created either by the environment (the browser) or by your code.
- Developer can create Error objects by 2 ways:
  - Explicitly:
    - var newErrorObj = new Error();
  - Implicitly:
    - thrown using the throw statement

Error Object Construction

- **Error constructor**
  - var e = new Error();
- **More than Six additional Error constructor ones exist and they all inherit Error:**

EvalError	**Raised by eval when used incorrectly**
RangeError	**Numeric value exceeds its range**
ReferenceError	**Invalid reference is used**
SyntaxError	**Used with invalid syntax**
TypeError	**Raised when variable is not the type expected**
URIError	**Raised when encodeURI( ) or decodeURI( ) are used incorrectly**

- **Using *instanceOf* when catching the error lets you know if the error is one of these built-in types.**

Error Object Properties

Property	Description
description	Plain-language description of error (IE only)
fileName	URI of the file containing the script throwing the error
lineNumber	Source code line number of error
message	Plain-language description of error (ECMA)
name	Error type (ECMA)
number	Microsoft proprietary error number

Error Object Standard Properties

- name →The name of the error constructor used to create the object
  - Example:
    - var e = new EvalError('Oops');
    - e.name;

      → "EvalError"

- Message → Additional error information:
  - Example:
    - var e = new Error('jaavcsritp is _not_ how you spell it');
    - e.message

      →" jaavcsritp is _not_ how you spell it"

Example!

Error Handling

JavaScript Error Handling

- There are two ways of catching errors in a Web page:

  1. *try...catch* **statement.**
  2. *onerror* **event.**

try...catch Statement

- The try...catch statement allows you to test a block of code for errors.
- The **try** *block* contains the code to be run.
- The **catch** *block* contains the code to be executed if an error occurs.
- Syntax

```
try {
    //Run some code here
}
catch(err) {
    //Handle errors here
}
```

Implicitly an Error object "err" is created

If an exception happens in "scheduled" code, like in setTimeout or any asynchournous behavior, then try..catch won't catch it

try...catch Statement

try {

✓ no error.

✓ no error.

an error! *control is passed to the catch block here.*

this will never execute.

}

catch(exception)

{

✓ error handling code is run here

}

✓ execution continues from here.

Example!

throw Statement

- The throw statement allows you to create an exception.
- Using throw statement with the try...catch, you can control program flow and generate accurate error messages.
- Syntax

  throw(exception)

- The exception can be a string, integer, Boolean or an object

try…catch & throw Example

```
try{
        if(x<100)
                throw “less100"
        else if(x>200)
                throw “more200"
    }
catch(er){
                if(er==“less100")
                    alert("Error! The value is too low")
                if(er == “more200")
                    alert("Error! The value is too high")
            }
```

Example!

Adding the *finally* statement

- If you have any functionality that needs to be processed regardless of success or failure, you can include this in the *finally* block.

try…catch…finally Statement

try {

- ✓ no error.
- ✓ no error.
- ✓ no error.

}

catch(exception)

{

- ✓ ~~error handling code will not run~~.

}

finally {

- ✓ This code will run even there is no failure occurrence.

}

✓ execution will be continued.

try…catch…finally Statement

try {

- ✓ no error.
- ✓ no error.

an error! *control is passed to the catch block here.*

this will never execute.

}

catch(exception)

{

- ✓ error handling code is run here
- ✓ error handling code is run here
- ✓ error handling code is run here

}

finally {

- ✓ This code will run even there is failure occurrence.

}

✓ execution will be continued.

Example!

try…catch…finally Statement

try {

✓ no error.

✓ no error.

an error! *control is passed to the catch block here.*

this will never execute.

}

catch(exception)

{

✓ error handling code is run here

an error!

~~error handling code is run here will never execute.~~

}

finally {

✓ This code will run even there is failure occurrence.

}

~~execution wont be continued.~~

Example!

onerror Event

- The old standard solution to catch errors in a web page.
- The *onerror* event is fired whenever there is a script error in the page.
- onerror event can be used to:
  - Suppress error.
  - Retrieve additional information about the error.

Suppress error

```
function supError()  {
        alert("Error occured")
   }
   window.onerror=supError
```

OR

```
   function supError()        {
        return true; //or false;
   }

   window.onerror=supError
```

The value returned determines whether the browser displays a standard error message.

true the browser does not display the standard error message.

false the browser displays the standard error message in the JavaScript console

Retrieve additional information about the error

```
onerror=handleErr
   function handleErr(msg,url,l,col,err) {
        //Handle the error here
        return true; //or false;
   }
```

where

- msg → Contains the message explaining why the error occurred.
- url → Contains the url of the page with the error script
- l → Contains the line number where the error occurred
- col → Column number for the line where the error occurred
- err → Contains the error object

Document Object Model
DOM

DOM

- DOM Stands for Document Object Model.
- W3C standard.
- Its an API that interact with documents like HTML, XML.. etc.
- Defines a standard way to access and manipulate HTML documents.
- Platform independent.
- Language independent

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

DOM

- The **document** object in the **BOM** is the top level of the **DOM** hierarchy.
- DOM is a representation of the whole document as nodes and attributes.
- You can access each of these nodes and attributes and change or remove them.
- You can also create new ones or add attributes to existing ones.

DOM is a subset of BOM.

In other word: **the document is yours!**

The document object in the BOM is the top level of the DOM hierarchy.

DOM Relationships

Scripting HTML

HTML DOM

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- It is a hierarchy of data types for HTML documents, links, forms, comments, and everything else that can be represented in HTML code.
- The general data type for objects in the DOM are *Nodes*. They have *attributes*, and some nodes can contain other nodes.
- There are several node types, which represent more specific data types for HTML elements.

DOM

- It allows code running in a browser to access and interact with every node in the document.
- Nodes can be created, moved and changed.
- Node types are represented by numeric constants.
- Event listeners can be added to nodes and triggered on occurrence of a given event.

DOM
is an API that represents and interacts with any **HTML** or **XML** document.

The DOM
is a document model loaded in
the browser and
representing the document
as a node tree,
where each node represents
part of the document

The DOM
is an application programming interface “API“

a set of functions or methods used to access some functionality

The DOM
Defines the logical structure of document and the way a document is accessed and manipulated

DOM
connects web pages to scripts or programming lan guages by representing the structure of a document.

HTML DOM

- According to the DOM, everything in an HTML document is a node.
- The DOM says:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text in the HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - Comments are comment nodes
- JavaScript is powerful DOM Manipulation

An **element**
is a specific type of **node**, one
that can be directly specified in
the **HTML** with
an **HTML tag**

Simple Example!

```
<html>
    <head>
        <title> Simple Example! </title>
    </head>
    <body>
        <h1>Heading</h1>
        <p id="my">A paragraph</p>
        <!--and that's all-->
    </body>
</html>
```

Root Node
➔ document.documentElement

Element Node

Text Node

document.body

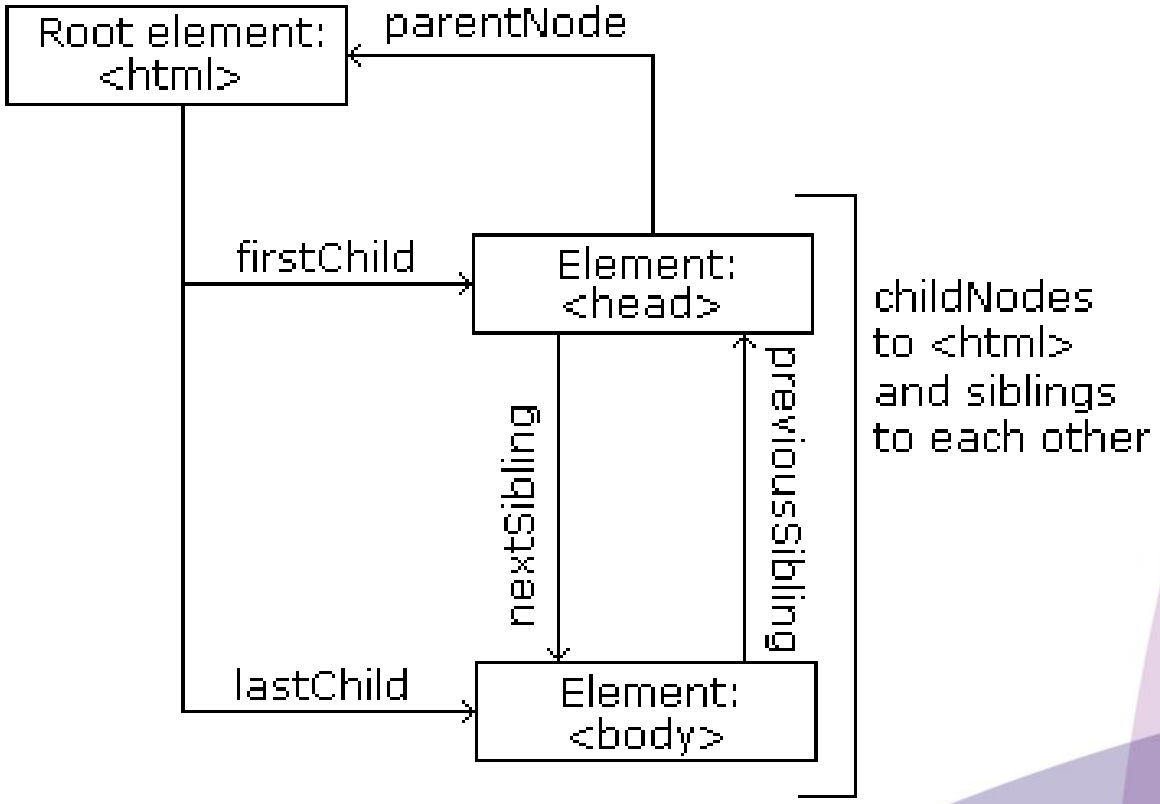
Attribute Node

Document Node

Comment Node

Node Tree

- The HTML DOM views HTML document as a node-tree.
- All the nodes in the tree have relationships to each other.
  - Parent
    - parentNode
  - Children
    - firstChild
    - lastChild
  - Sibling
    - nextSibling
    - previousSibling

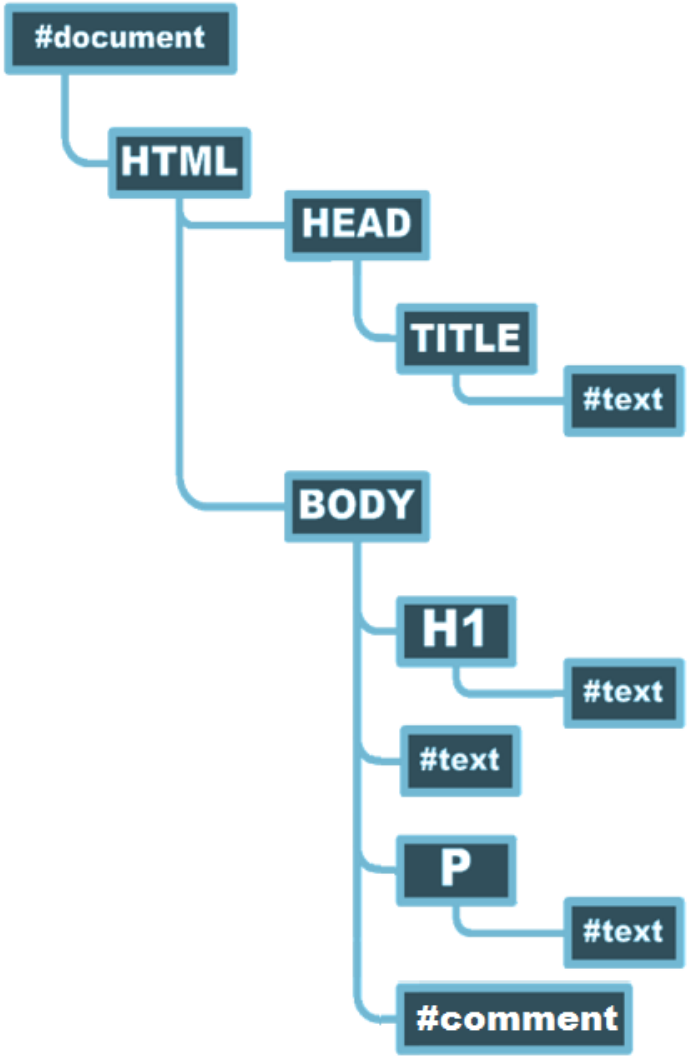


Nodes Relationships

- The terms **parent**, **child**, and **sibling** are used to describe the relationships.
  - Parent nodes have children.
  - Children on the same level are called siblings (brothers or sisters).
- Attribute nodes are not child nodes of the element they belong to, and have no parent or sibling nodes
- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children
- Siblings are nodes with the same parent

Simple Example!

```
<html>
        <head>
            <title>Simple Example!</title>
        </head>
        <body>
            <h1>Greeting</h1>
             Welcome All
            <p>A paragraph</p>
            <!-- and that's all-->
        </body>
 </html>
```



Node Properties

- All nodes have three main properties

Property	Description
nodeName	Returns HTML Tag name in uppercase display
tagname	
nodeType	returns a numeric constant to determine node type. There are 12 node types.
nodeValue	returns null for all node types except for text and comment nodes.

Using nodeName
If node is text it returns #text
For comment it returns #comment
For document it returns #document

Value	Description
1	Element Node
2	Attribute Node
3	Text Node
8	Comment Node
9	Document Node

To get the Root Element:
document.documentElement.

Node Collections

- Node Collections have One Property
  - **length : gives the length of the Collection.**
    - **e.g. childNodes.length:** returns number of elements inside the collection
- We can check if there is child collection using
  - **hasChildNodes():** Tells if a node has any children
- We can check if there is attribute collection using
  - **hasAttributes():** Tells if a node has any attributes

Collection	Description	Accessing
childNodes	Collection of element's children	childNodes[ ] childNodes.item()
attributes	Returns collection of the attributes of an element	attributes[ ] attributes.item()

Dealing With Nodes

- Dealing with nodes fall into four main categories:
  - Accessing Node
  - Modifying Node’s content
  - Adding New Node
  - Remove Node from tree

Accessing DOM Nodes

- You can access a node in **5** main ways:
  - [window.]document.getElementById("id")
  - [window.]document.getElementsByName("name")
  - [window.]document.getElementsByTagName("tagname")
  - By navigating the node tree, using the node relationships
  - New HTML5 Selectors.

Example!

New HTML5 Selectors

- In HTML5 we can select elements by ClassName

```
var elements = document.getElementsByClassName('entry');
```

- Moreover there's now possibility to fetch elements that match provided CSS syntax

```
var elements =  document.querySelectorAll(".someClasses)");
```

```
var elements = document.querySelectorAll("div,p");
```

```
var elements = document.querySelector("#someID");
```

```
var first_td =   document.querySelector("span");
```

Accessing DOM Nodes

Navigating the node tree, using the node relationships

firstChild	Move direct to first child node
lastChild	Move direct to last child node
parentNode	To access child's parent node
nextSibling	Navigate down the tree one node step
previousSibling	Navigate up the tree one node step
Using children collection → childNodes[ ]	

Example!

Accessing DOM Elements

Navigating the elements nodes, using the relationships

firstElementChild	Move direct to first Element child
lastElementChild	Move direct to last Element child
parentElement	To access child's Element parent
nextElementSibling	Navigate down the tree to next Element
previousElementSibling	Navigate up the tree to previous Element

Example!

Modifying Node's Content

- Changing the Text Node by using

innerHTML	Sets or returns the HTML contents (+text) of an element
textContent	Equivalent to innerText.
nodeValue → with text and comment nodes only	
setAttribute()	Modify/Adds a new attribute to an element
just using attributes as object properties	

Example!

Node's Class Attribute

- The global class attribute is get and set via className property
- The classList property returns a collection of the class attributes of the caller element, it has the following methods
  - add("className")
  - remove("className")
  - toggle("className")
  - replace("oldClassNm","newClassNm")

Manipulating Styles

- Modifying style properties of any HTML element is accessed using the style object.

- For inline style
  - Node.style[.prop_name]
  - Node.style.cssText

- To read internal or external styling in general
  - document.styleSheets
  - document.styleSheets[i].cssRules
  - document.styleSheets[i].cssRules[idx].selectorText
  - document.styleSheets[i].cssRules[idx].cssText

- To read none inline styling applied for specific element
  - getComputedStyle(elem).prop_nm
  - getComputedStyle(elem). getPropertyValue(prop_nm)

Creating & Adding Nodes

Method	Description
createElement()	To create new tag element
createTextNode()	To create new text element
createAttribute()	To creates an attribute element
createComment()	To creates an comment element

Creating & Adding Nodes

Method	Description
cloneNode(true|false)	Creating new node a copy of existing node. It takes a Boolean value true: Deep copy with all its children or false: Shallow copy only the node
b.appendChild(a)	To add new created node “a” to DOM Tree at the end of the selected elem ent “b”.
b.append(a)	Experimental function to o add new created node “a” to DOM Tree at the end of the selected element “b”.
b.prepend(a)	Experimental function to o add new created node “a” to DOM Tree at the top of the selected element “b”.

Example!

Creating & Adding Nodes

https://developer.mozilla.org/en-US/docs/Web/API/Element

Method	Description
insertBefore(a,b)	Similar to appendChild() with extra parameter, specifying before which element to insert the new node. a: the node to be inserted b: where a should be inserted before **document.body.insertBefore(a,b)**
e.insertAdjacentElement(pos,elem)	○ e: represents the target element ○ elem: represents the element to be added ○ pos: represents the position relative to the targetElem • 'beforebegin': Before the targetElement itself. • 'afterbegin': Just inside the targetElement, before its first child. • 'beforeend': Just inside the targetElement, after it s last child. • 'afterend': After the targetElement itself.

Example!

Removing DOM Nodes

Method	Description
removeChild()	To remove node from DOM tree
parent.replaceChild(n,o)	To remove node from DOM tree and put another one in its place n: new child o: old child
removeAttribute()	Removes a specified attribute from an element

- To quick replace a node set its outerHTML property

  elem.outerHTML=" something</div";

- A quick way to wipe out all the content of a subtree is to set the innerHTML to a blank string. This will remove all of the children of <body>

  document.body.innerHTML="";

Example!

Summary

- Access nodes:
  - Using parent/child relationship properties parentNode, childNodes, firstChild, lastChild, nextSibling, previousSibling
  - Using getElementsById(), getElementsByTagName(), getElementsByName()
- Modify nodes:
  - Using innerHTML or innerText/textContent
  - Using nodeValue or setAttribute() or just using attributes as object properties
- Remove nodes with
  - removeChild() or replaceChild()
- And add new ones with
  - appendChild(), cloneNode(), insertBefore()

Modeling HTML or XML documents as objects are not part of the core JavaScript language.

The DOM
Defines the logical structure of document and the way a document is accessed and manipulated

Dynamic HTML

the art of making dynamic and interactive web pages.

DHTML

- DHTML has no official definition or specification.
- DHTML stands for Dynamic HTML.
- DHTML is NOT a scripting language.
- DHTML is not w3c (i.e. not a standard).
- DHTML is a browser feature-that gives you the ability to make dynamic Web pages.
- "*Dynamic*" is defined as the ability of the browser to alter a web page's look and style *after* the document has been loaded.
- DHTML is very important in web development

DHTML

- DHTML uses a combination of:

  1. Scripting language
  2. DOM
  3. CSS

  to create HTML that can change even after a page has been loaded into a browser.

- DHTML is supported by 4.x generation browsers.

Assignments