UNIT-2

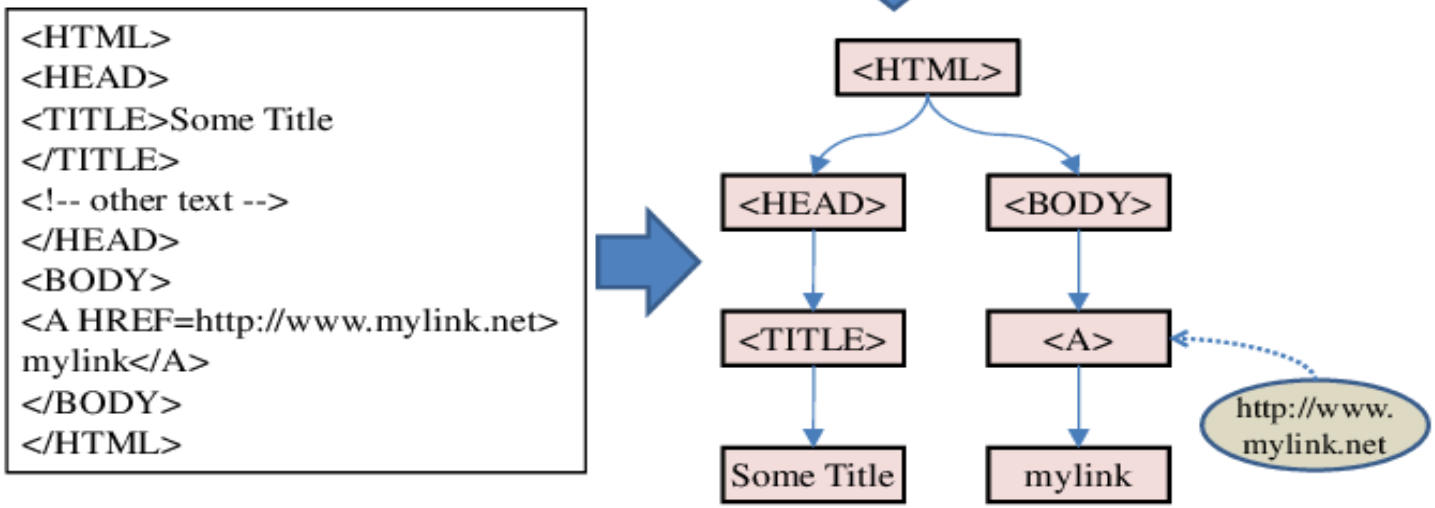
DOM – Document Object Model

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects.**
- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements

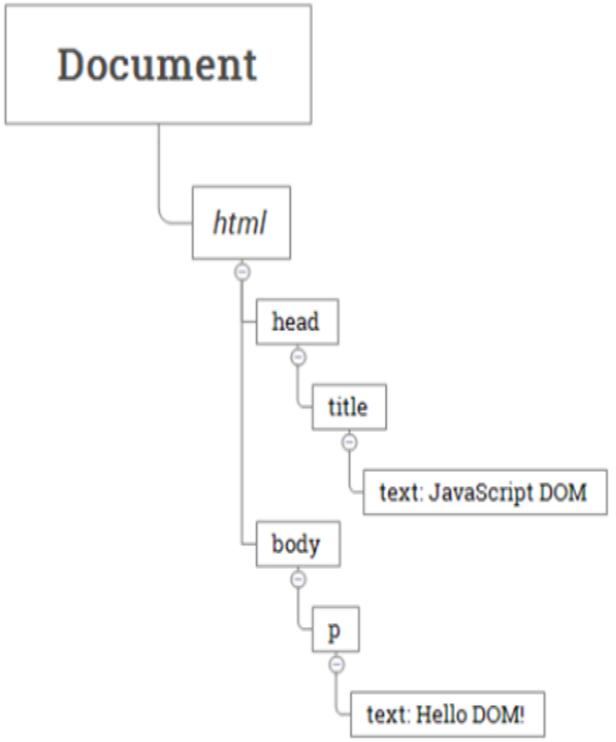
DOM – Document Object Model

```
<html>
<head> <title>Some Title</title> <!-- some text --> </head>
<body> <a href=http://www.mylink.net>mylink</a></body>
</html>
```



DOM – Tree Representation

```
<html>
   <head>
      <title>JavaScript DOM</title>
   </head>
   <body>
      <p>Hello DOM!</p>
   </body>
</html>
```



DOM – Document Object Model

- In this DOM tree, the document is the root node. The root node has one child which is the <html> element. The <html> element is called the *document element*.

- Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a node in the tree.

DOM – Document Object Model

- Whenever an *HTML document* is loaded into a *web browser*, it becomes a **document object**. The Document Object Model (DOM) represents that same document so it can be manipulated. The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.

DOM – Document Object Model

- In the HTML DOM, everything is referred to as a node. This means, HTML DOM is composed of units called nodes.

For example:

1. The document (web-page) is a document node.
2. All HTML elements (<*head*>, <*body*>) are element nodes.
3. All HTML attributes (<input *type*=*"text"* />)are attribute nodes.
4. Text inside the HTML elements are text nodes.
5. Even, the comments are comment nodes.

DOM – Selecting DOM elements

There are several ways to access form elements .They are :

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by HTML object collections

1) **Finding HTML Element by Id**

The easiest way to find an HTML element in the DOM, is by using the element id.

If the element is found, the method will return the element as an object If the element is not found, returns null.

DOM – Selecting DOM elements – by id

```
<html>
<body>
<h2>Finding HTML Elements by Id</h2>
<p id="intro">Hello World!</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>
<p id="demo"></p>
<script>
var myElement = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is " + myElement.innerHTML;
</script>
</body>
</html>
```

DOM – Selecting DOM elements - tagname

```
<html>
<body>
<h2>Finding HTML Elements by Tag Name</h2>
<div id="main">
<p>The DOM is very useful.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
</div>
<p id="demo"></p>
<script>
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;
</script>
</body>
</html>
```

DOM – Selecting DOM elements - classname

```
<html>
<body>
<h2>Finding HTML Elements by Class Name</h2>
<p>Hello World!</p>
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b>
method.</p>
<p id="demo"></p>
<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
</body>
</html>
```

querySelector()

The querySelector() method returns the first element that matches one or more CSS selectors. If no match is found, it returns null.

Syntax

```
var ele = document.querySelector(selector);
```

- ele – First matching element or null (if no element matches the selectors)

- selector – one or more CSS selectors, such as "#fooId", ".fooClassName", ".class1.class2", or ".class1, .class2"

```
<html>
<body>

<h2 class="example">A heading with class="example"</h2>
<p class="example">A paragraph with class="example".</p>

<p>Click the button to add a background color to the first element in the
document with class="example".</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.querySelector(".example").style.backgroundColor = "red";
}
</script>

</body>
</html>
```

- **querySelectorAll()**
- The querySelectorAll() method returns all elements in the document that matches a specified CSS selector(s), as a static NodeList object.

- The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

- You can use the length property of the NodeList object to determine the number of elements that matches the specified selector, then you can loop through all elements and extract the info you want.


```
<body>
<h2 class="example">A heading with class="example"</h2>
<p class="example">A paragraph with class="example".</p>
<p>Click the button to add a background color all elements with class="example".</p>
<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var x, i;
  x = document.querySelectorAll(".example");
  for (i = 0; i < x.length; i++) {
    x[i].style.backgroundColor = "red";
  }
}
</script>
```

DOM – Traversal

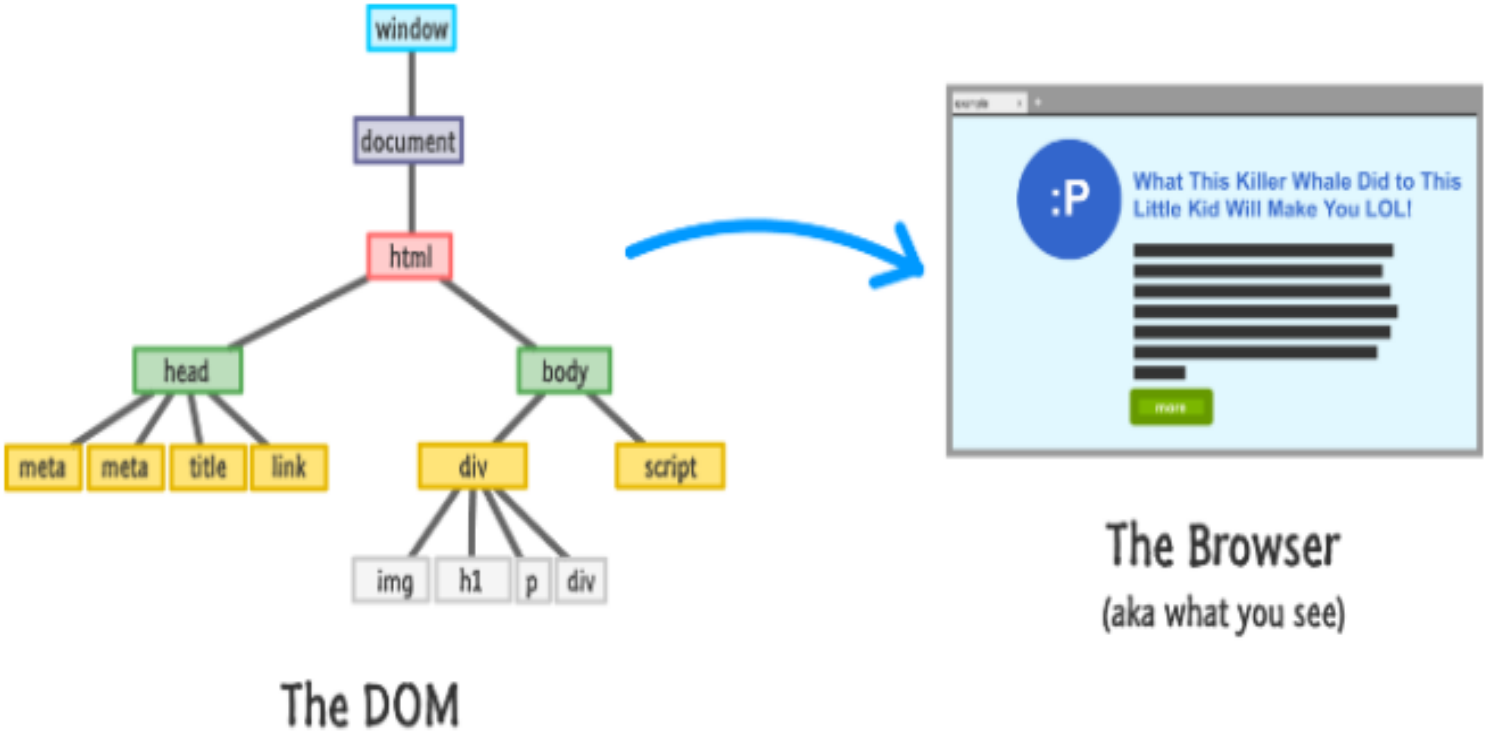
The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have any number of children
- Siblings (brothers or sisters) are nodes with the same parent

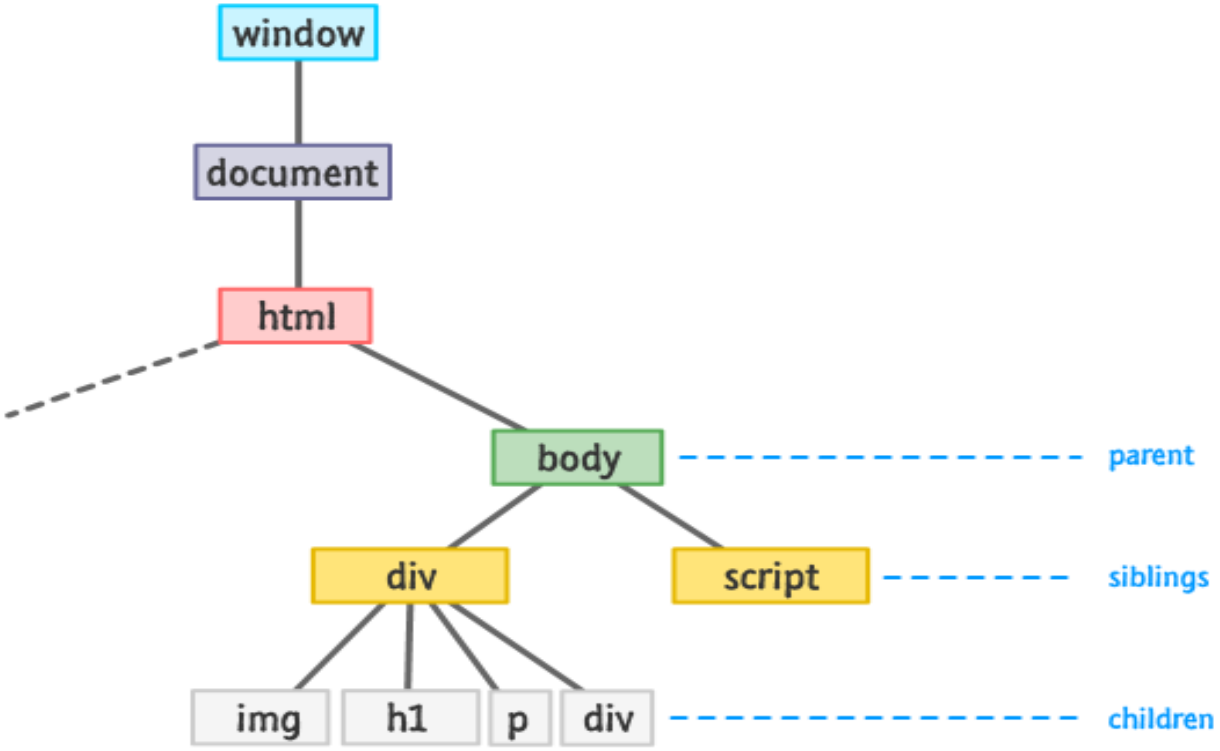
HTML5,Jquery and Ajax

DOM – Traversal

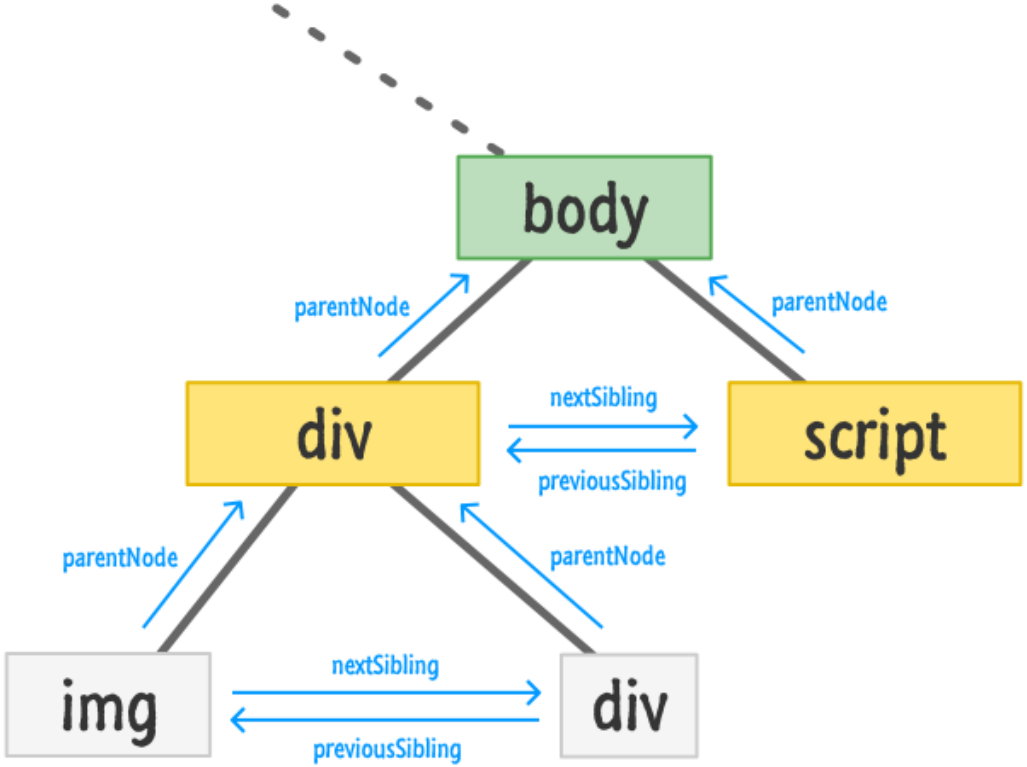


HTML5,Jquery and Ajax

DOM – Traversal



DOM – Traversal



DOM – Navigation

You can use the following node properties to navigate between nodes with JavaScript:

- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling
- previousSibling

DOM – Navigation

- The parentNode property returns the parent node of the specified node, as a Node object.
- In HTML, the document itself is the parent node of the HTML element, HEAD and BODY are child nodes of the HTML element.
- This property is read-only.

Syntax

node.parentNode

DOM – Navigation

```
<p>Example list:</p>
<ul>
  <li id="myLI">Coffee</li>
  <li>Tea</li>
</ul>
<p>Click the button to get the node name of the parent node of the li element in the
list.</p>

<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementById("myLI").parentNode.nodeName;
  document.getElementById("demo").innerHTML = x;
}
</script>
```

DOM – Navigation

- The parentElement property returns the parent element of the specified element.
- The difference between parentElement and parentNode, is that parentElement returns *null* if the parent node is not an element node:

Eg –

```
document.body.parentNode; // Returns the <html> element
document.body.parentElement; // Returns the <html> element

document.documentElement.parentNode; // Returns the Document node
document.documentElement.parentElement; // Returns null (<html> does not have a parent ELEMENT node)
```

DOM – Navigation

```
<style>
div {
  box-sizing: border-box;
  padding: 16px;
  width: 100%;
  background-color: red;
  color: #fff;
}
.closebtn {
  float: right;
  font-size: 30px;
  font-weight: bold;
  cursor: pointer;
}
</style>
```

```
</head>
<body>

<div>
  <span onclick="this.parentElement.style.display =
'none';" class="closebtn">&times;</span>
  <p>To close this container, click on the X symbol
to the right.</p>
</div>
```

nextSibling Property

- The nextSibling property returns the node immediately following the specified node, in the same tree level.
- The returned node is returned as a Node object.
- The difference between this property and nextElementSibling, is that nextSibling returns the next sibling node as an element node, a text node or a comment node, while nextElementSibling returns the next sibling node as an element node (ignores text and comment nodes).
- This property is read-only.

DOM – Navigation

```
<p>Example list:</p>
<ul><li id="item1">Coffee (first li)</li><li id="item2">Tea (second li)</li></ul>
<p>Click the button to get the HTML content of the next sibling of the first list item.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementById("item1").nextSibling.innerHTML;
  document.getElementById("demo").innerHTML = x;
}
</script>
```

PES UNIVERSITY ONLINE

nextElementSibling Property

- The nextElementSibling property returns the element immediately following the specified element, in the same tree level.

Syntax

```
node.nextElementSibling
```

DOM – Navigation

```
<html>
<body>
<p>Example list:</p>
<ul>
  <li id="item1">Coffee (first li)</li>
  <li id="item2">Tea (second li)</li>
</ul>
<p>Click the button to get the HTML content of the next sibling of the first list item.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementById("item1").nextElementSibling.innerHTML;
  document.getElementById("demo").innerHTML = x;
}
</script>
</body>
</html>
```

previousSibling Property

- The previousSibling property returns the previous node of the specified node, in the same tree level.
- The returned node is returned as a Node object.
- The difference between this property and previousElementSibling, is that previousSibling returns the previous sibling node as an element node, a text node or a comment node, while previousElementSibling returns the previous sibling node as an element node (ignores text and comment nodes).
- This property is read-only.

Syntax

node.previousSibling

childNodes Property

- The childNodes property returns a collection of a node's child nodes, as a NodeList object.
- The nodes in the collection are sorted as they appear in the source code and can be accessed by index numbers. The index starts at 0.
- You can use the length property of the NodeList object to determine the number of child nodes, then you can loop through all child nodes and extract the info you want.
- This property is read-only.
- To return a collection of a node's element nodes (excluding text and comment nodes), use the children property.

Syntax

```
element.childNodes
```

firstChild Property

- The firstChild property returns the first child node of the specified node, as a Node object.
- This property is read-only.
- Use the *element*.childNodes property to return any child node of a specified node. childNodes[0] will produce the same result as firstChild.
- To return the last child node of a specified node, use the lastChild property.

Syntax

node.firstChild

Event and Event-handling in JavaScript

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Event and Event-handling in JavaScript

```
<html>
  <head>
    <script type = "text/javascript">
                function sayHello() {
          alert("Hello World")
        }
</script>
  </head>
    <body>
    <p>Click the following button and see result</p>
    <form>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
    </form>
  </body>
</html>
```

Event and Event-handling in JavaScript

Event Handler	Description
onBlur	It executes when the input focus leaves the field of a text, textarea or a select option.
onChange	It executes when the input focus exits the field after the user modifies its text.
onClick	In this, a function is called when an object in a button is clicked, a link is pushed, a checkbox is checked or an image map is selected. It can return false to cancel the action.
onError	It executes when an error occurs while loading a document or an image.
onFocus	It executes when input focus enters the field by tabbing in or by clicking but not selecting input from the field.

Event and Event-handling in JavaScript

onLoad	It executes when a window or image finishes loading.
onMouseOver	The JavaScript code is called when the mouse is placed over a specific link or an object.
onMouseOut	The JavaScript code is called when the mouse leaves a specific link or an object.
onReset	It executes when the user resets a form by clicking on the reset button.
onSelect	It executes when the user selects some of the text within a text or textarea field.
onSubmit	It calls when the form is submitted.
onUnload	It calls when a document is exited.

Event and Event-handling in JavaScript

One severe shortcoming of the DOM Level 0 Event Model is that, because a property is used to store a reference to a function that's to serve as an event handler, only one event handler per element can be registered for any specific event type at a time

DOM Level 2 event handlers—also termed listeners—are established via an element method. Each DOM element defines a method named addEventListener() that's used to attach event handlers (listeners) to the element.

Event and Event-handling in JavaScript

Under the DOM Level 2 Event Model, when an event is triggered, the event first propagates from the root of the DOM tree down to the target element and then propagates again from the target element up to the DOM root. The former phase (root to target) is called **capture phase**, and the latter (target to root) is called **bubble phase.**

Event and Event-handling in JavaScript

The addEventListener() method

- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.

Event and Event-handling in JavaScript

- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.

Event and Event-handling in JavaScript

Syntax

element.addEventListener(*event, function, useCapture*);

- The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional. The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

Eg –

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

Event and Event-handling in JavaScript


```
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
document.getElementById("myBtn").addEventListener("click", displayDate);
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
</body>
</html>
```

Event and Event-handling in JavaScript

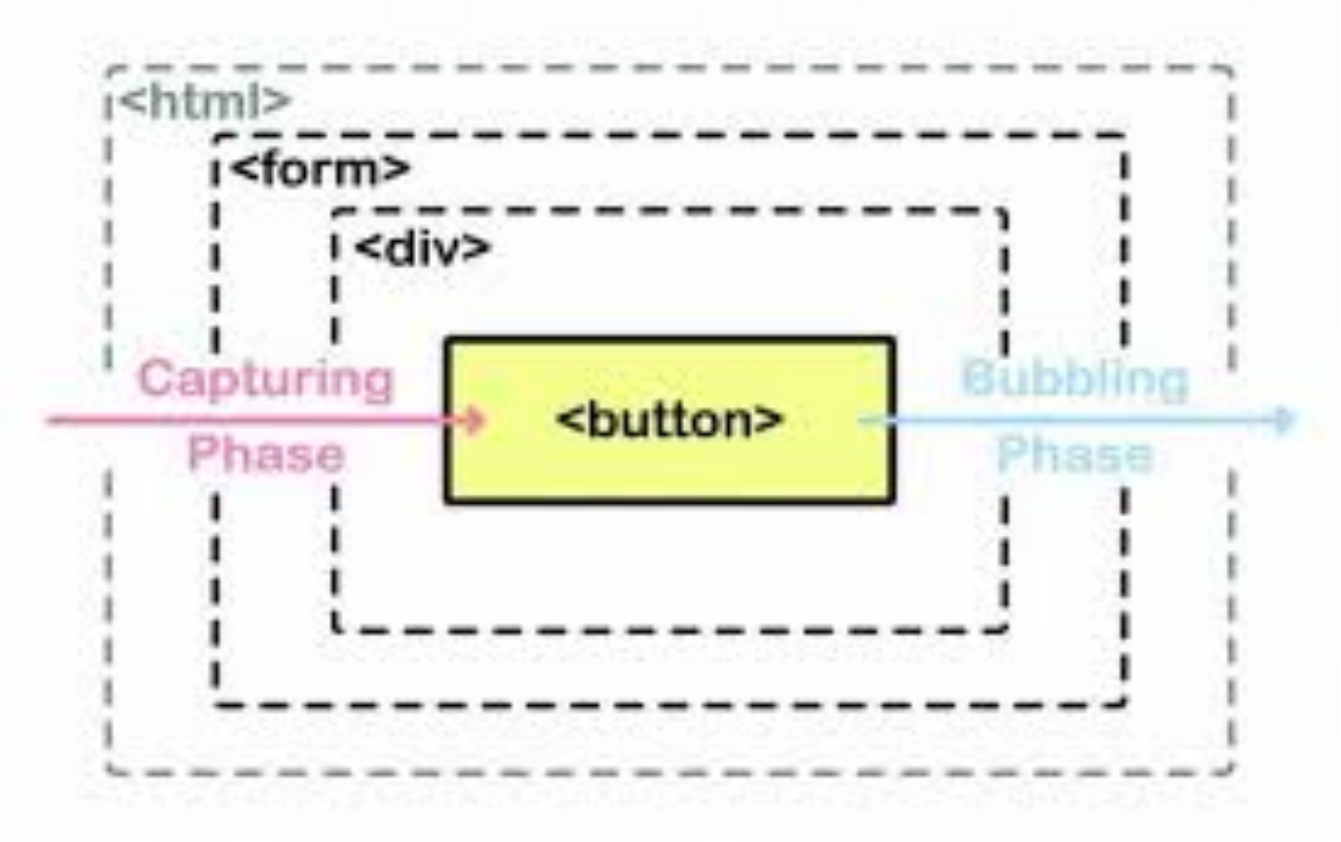
```
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<button id="myBtn">Try it</button>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);
```

```
function myFunction()
{
  alert ("Hello World!");
}
function someOtherFunction()
{
  alert ("This function was also
executed!");
}
</script>
</body>
</html>
```

Event and Event-handling in JavaScript

- There are two ways of event propagation in the HTML DOM, **bubbling and capturing.**
- Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?
- In ***bubbling*** the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the element's click event.
- In ***capturing*** the outer most element's event is handled first and then the inner: the element's click event will be handled first, then the <p> element's click event.
- With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

Event and Event-handling in JavaScript



HTML 5 New Features – New form input elements

- HTML5 specifications introduced new Input types and properties

New Input Types

- email: email address
- number: spinbox
- range: slider
- url: web addresses
- color: color pickers
- search: search boxes
- date: date
- time: time
- file: input file selection

New Input properties

- placeholder
- required
- pattern
- autofocus

HTML 5 New Features- Audio & Video tags

HTML <video> Tag

The <video> tag is used to embed video content in a document, such as a movie clip or other video streams.

The <video> tag contains one or more <source> tags with different video sources. The browser will choose the first source it supports.

The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element.

There are three supported video formats in HTML: MP4, WebM, and OGG.

HTML 5 New Features- Audio & Video tags

Attribute	Value	Description
audio	muted	Defining the default state of the the audio. Currently, only "muted" is allowed
autoplay	autoplay	If present, then the video will start playing as soon as it is ready
controls	controls	If present, controls will be displayed, such as a play button
height	*pixels*	Sets the height of the video player
loop	loop	If present, the video will start over again, every time it is finished
poster	*url*	Specifies the URL of an image representing the video
preload	preload	If present, the video will be loaded at page load, and ready to run. Ignored if "autoplay" is present
src	*url*	The URL of the video to play
width	*pixels*	Sets the width of the video player

HTML <audio> Tag

The <audio> tag is used to embed sound content in a document, such as music or other audio streams.

The <audio> tag contains one or more <source> tags with different audio sources. The browser will choose the first source it supports.

The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

There are three supported audio formats in HTML: MP3, WAV, and OGG.(**Ogg** is a free, open container **format** maintained by the Xiph.Org Foundation.)

HTML 5 New Features- Audio & Video tags

Attribute	Value	Description
autoplay	autoplay	Specifies that the audio will start playing as soon as it is ready.
controls	controls	Specifies that controls will be displayed, such as a play button.
loop	loop	Specifies that the audio will start playing again (looping) when it reaches the end
preload	preload	Specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present.
src	*url*	Specifies the URL of the audio to play

HTML 5 New Features

- The <progress> tag represents the completion progress of a task.
- Always add the <label> tag for describing the task!
- Use JavaScript to manipulate the value of the progress bar

Syntax:

```
<label for="file">Downloading progress:</label>
<progress id="file" value="32" max="100"> 32% </progress>
```

Attribute	Value	Description
max	*number*	Specifies how much work the task requires in total. Default value is 1
value	*number*	Specifies how much of the task has been completed

Canvas Element

The <canvas> tag is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> tag is transparent, and is only a container for graphics, you must use a script to actually draw the graphics.

Any text inside the <canvas> element will be displayed in browsers with JavaScript disabled and in browsers that do not support <canvas>.

Syntax

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

HTML5 - Canvas

Canvas – context methods

Method	Description
fillRect(x, y, width, height)	Draws a filled rectangle
strokeRect(x, y, width, height)	Draws a rectangular outline
clearRect(x, y, width, height)	Clears the specified rectangular area, making it fully transparent
moveTo(x, y)	Moves the pen to the coordinates specified by x and y
lineTo(x, y)	Draws a line from the current drawing position to the position specified by x and y
arc(x, y, r, sAngle, eAngle, anticlockwise)	Draws an arc centered at (x, y) with radius r starting at sAngle and ending at eAngle going anticlockwise (defaulting to clockwise).
arcTo(x1, y1, x2, y2, radius)	Draws an arc with the given control points and radius, connected to the previous point by a straight line

HTML5 – SVG

SVG Element

- SVG stands for Scalable Vector Graphics.
- SVG defines vector-based graphics using HTML elements
- SVG graphics do NOT lose any quality if they are zoomed or resized

```
<svg width="100" height="100">
     <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
    fill="yellow" />
</svg>
```

HTML5 – SVG

SVG – Predefined Shape Element


```
<html>
<body>

<h1>The svg element</h1>

<svg width="400" height="100">
  <rect width="400" height="100" style="fill:rgb(0,0,255);stroke-
width:10;stroke:rgb(0,0,0)" />
  Sorry, your browser does not support inline SVG.
</svg>

</body>
</html>
```

HTML 5

Semantic Elements – New Tags

HTML5

Geolocation API

The HTML Geolocation API is used to locate a user's position.

The HTML Geolocation API is used to get the geographical position of a user.

The getCurrentPosition() method is used to return the user's position.

The Navigator.geolocation read-only property returns a Geolocation object that gives Web content access to the location of the device. This allows a Web site or app to offer customized results based on the user's location.

Geolocation API

The getCurrentPosition() method returns an object on success. The latitude, longitude and accuracy properties are always returned. The other properties are returned if available:

Property	Returns
coords.latitude	The latitude as a decimal number (always returned)
coords.longitude	The longitude as a decimal number (always returned)
coords.accuracy	The accuracy of position (always returned)
coords.altitude	The altitude in meters above the mean sea level (returned if available)
coords.speed	The speed in meters per second (returned if available)

The Geolocation object also has other interesting methods:

- watchPosition() - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- clearWatch() - Stops the watchPosition() method.

```
navigator.geolocation.watchPosition(showPosition);
```

HTML5 – Geo Location

Associated Objects

- Success callback function receives **position** object with these read only properties
  - double latitude
  - double longitude
  - double accuracy
  - double altitude
  - double altitudeAccuracy
  - double heading (direction)
  - double speed

- Error callback function receives **error** object with these two properties
  - short code
    - 1, meaning PERMISSION_DENIED
    - 2, meaning POSSITION_UNAVAILABLE
    - 3, meaning TIMEOUT
  - DOMString message

- **Options** object (third parameter to getCurrentPosition or watchPosition)
  - enableHighAccuracy // true or false
  - timeout // milliseconds
  - maximumAge // milliseconds

HTML5 – Web Worker

Introduction

A **web worker** is a JavaScript running in the background, without affecting the performance of the page.

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page.
You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

HTML5 – Web Worker

Introduction

Check Web Worker Support

```
if (typeof(Worker) !== "undefined")
            {
             // Yes! Web worker support!
             // Some code.....
            }
            else {
             // Sorry! No Web Worker support..
            }
```

Create a Web Worker File

The script that counts is stored in the "demo_workers.js" file:

```
var i = 0;
function timedCount()
{
  i = i + 1;
  postMessage(i);
  setTimeout("timedCount()",500);
}
timedCount();
```

postMessage() method - used to post a message back to the HTML page.

Create a Web Worker Object

The web worker file need to be called from an HTML page.
The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if (typeof(w) == "undefined") {
  w = new Worker("demo_workers.js");
}
```

Then we can send and receive messages from the web worker.

HTML5 – Web Worker

Introduction

Add an "onmessage" event listener to the web worker.

```
w.onmessage = function(event)
{
  document.getElementById("result").innerHTML = event.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the terminate() method:

```
w.terminate();
```

Reuse the Web Worker

If you set the worker variable to undefined, after it has been terminated, you can reuse the code:

```
w = undefined;
```

XML

Introduction

- XML stands for Extensible Markup Language.
- It allows the user to define their own set of tags, and also makes it possible for others (people or programs) to understand it.
- It inherits the features of SGML and combines it with the features of HTML.
- It is a smaller version of SGML.
- XML is a meta language and it describes other languages.
- The data contained in an XML file can be displayed in different ways.
- Style sheets help transform structured data into different HTML views. This enables data to be displayed on different browsers.

Introduction

HTML Code

```
<UL>
    <LI> TOM CRUISE
  <UL>
    <LI> CLIENT ID : 100
    <LI> COMPANY : XYZ Corp.
    <LI> Email : tom@usa.net
    <LI> Phone : 3336767
    <LI> Street Address: 25th
  St.
    <LI> City : Toronto
    <LI> State : Toronto
    <LI> Zip : 20056
  </UL>
</UL>
```

XML Code

```
<Details>
  <CONTACT>
    <PERSON_NAME>TOM CRUISE
  </PERSON_NAME>
    <ID>     100     </ID>
    <Company>XYZ Corp. </Company>
    <Email>    tom@usa.net</Email>
    <Phone>  3336767   </Phone>
    <Street>  25th St.     </Street>
    <City>   Toronto     </City>
    <State>  Toronto     </State>
    <ZIP>    20056       </ZIP>
  </CONTACT>
</Details>
```

XML

Introduction

- This data format fully supports hierarchical data structures and is very appropriate when receiving complex data as a response.
- It is also very human readable. Most browsers have built in XML readers that allow you to inspect XML files.
- Since XML was the first standard hierarchical data format, most APIs have built in functionality to automatically convert XML data streams into native data structures like objects.
- This data format is about three times as large as CSV. This is because each data element has an associated open and close parameter tag.

XML

Introduction

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
   <employee id="be129">
      <firstname>Jane</firstname>
      <lastname>Doe</lastname>
      <title>Engineer</title>
      <division>Materials</division>
   </employee>
<employee id="be131">
      <firstname>Jack</firstname>
      <lastname>Dee</lastname>
      <title>Engineering Manager</title>
      <division>Materials</division>
   </employee>
</employees>
```

XML Parser

- The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

- However, before an XML document can be accessed, it must be loaded into an XML DOM object. All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

XML

Introduction

```
<html>
<body>
<p id="demo"></p>
<script>
var parser, xmlDoc;
var text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

JSON

Introduction

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent

JSON uses JavaScript syntax, but the JSON format is text only. Text can be read and used as a data format by any programming language.

JSON NOTATION

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value. The values can be a string, number, an object(JSON object),an array, a Boolean, null.

Eg-

{ "name": "John" } in JSON and { name: "John" } in JavaScript

JSON

Introduction


```
<html>
<body>
<p>Access a JavaScript object using the bracket notation:</p>
<p id="demo"></p>
<script>
var myObj, x;
myObj = { "name":"John", "age":30, "city":"New York" };
x = myObj["name"];
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

JSON

Introduction

<If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Eg-

```
<html>
<body>
<h2>Create JSON string from a JavaScript array.</h2>
<p id="demo"></p>
<script>
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

JSON

Introduction

If you receive data in JSON format, you can convert it into a JavaScript object:

Eg-

```
<html>
<body>
<h2>Convert a string written in JSON format, into a JavaScript object.</h2>
<p id="demo"></p>
<script>
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
</script>
</body>
</html>
```

JSON

Pros

- This data format supports hierarchical data while being smaller in size than XML.
- As its name implies, it was also created to more easily parse data into native Javascript objects, making it very useful for web applications.
- JSON is the best of both worlds with respect to CSV and XML. It's simple and compact like CSV, but supports hierarchical data like XML. Unlike XML, JSON formats are only about twice as large as CSV formats.

JSON

Cons

- This data format has a little bit less support than XML.
- Since JSON is relatively newer than XML, fewer APIs exist to automatically convert JSON to native data structures.
- However, this is rapidly changing because newer APIs and plugins are supporting both XML and JSON.

JSON

Cons

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

JSON Vs XML

JSON	XML
Based on JavaScript language	Derived from SGML
Way of representing objects	Markup language to represent data items
No support for namespaces	Supports namespaces
Support arrays	Doesn’t support arrays
Files are very easy to read as compared to XML	Documents are comparatively difficult to read and interpret
Doesn’t use end tag	Has start and end tags
Less secure	More secured than JSON
Doesn’t supports comments	Supports comments
Supports only UTF-8 encoding	Supports various encoding

jQuery
Introduction

- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.
- jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

Javascript Vs Jquery

Java script

```
function changeColor(color)
{
  document.body.style.background = color;
}
Onload=”changeColor('blue');”
```

jQuery

```
$ ('body') .css ('background', '#0000FF');
```

Here is the list of important core features supported by jQuery –

- **DOM manipulation** − The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called Sizzle.
- **Event handling** − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** − The jQuery helps you a lot to develop a responsive and feature rich site using AJAX technology.

jQuery
Introduction

- **Animations** – The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** – The jQuery is very lightweight library - about 19KB in size (Minified and zipped).
- **Cross Browser Support** – The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** – The jQuery supports CSS3 selectors and basic XPath syntax.

jQuery
Introduction

There are several ways to start using jQuery on your web site. You can:

1) Download the jQuery library from jQuery.com

Go to the https://jquery.com/download/ to download the latest version available.

Now put downloaded **jquery-3.5.1.js** file in a directory of your website.

2) Include jQuery from a CDN, like Google

We can include jQuery library into our HTML code directly from Content Delivery Network (CDN). Google and Microsoft provides content deliver for the latest version.

Introduction

```
<html>
<head>
<title> welcome to jquery</title>
<script src ="jquery-3.5.1.js"></script>
<script>
$(document).ready(function()
{
    alert("this is an alert box ");
});
</script>
</head>
<body>
<p> This is a sample program in jquery
</body>
</html>
```

jQuery
Introduction


```
<html>
<head>
 <title>The jQuery Example</title>
  <script type = "text/javascript"
   src = " https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js ">
   </script>
   <script >
   $(document).ready(function() {
    document.write("Hello, World!");
      });
    </script>
</head>
  <body>
   <p> <h1>Hello</h1></p>
 </body>
</html>
```

jQuery
Introduction

When using jQuery to read or manipulate the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the **$(document).ready()** function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

To do this, we register a ready event for the document as follows –

```
$(document).ready(function() {
        // do stuff when DOM is ready
      });
```

Shorthand form

```
$(function() {
   console.log( "ready!" );
});
```

jQuery
Introduction


```
<html>
 <head>
 <title>The jQuery Example</title>
 <script src = " jquery-3.5.1.js "> </script>
<script>
$(document).ready(function() {
$("div").click(function()
{
   alert("Hello, world!");
});
 });
</script>
</head>
<body>
 <div id = "mydiv">  Click on this to see a dialogue box. </div>
 </body>
 </html>
```

jQuery

Selecting Elements – using CSS Selectors

Select By	Example
ID	$(“#header”)
Class	$(“.updated”)
Tag Name	$(“table”)
Combination	$(“table.user-list”) or $(“#footer ul.menu li”)
Basic Filters	:first, :last, :even, :odd
Content Filters	:empty , :contains(text), :has(selector)
Attribute Filters	[attribute], [attribute=value], [attribute!=value]
Forms	:input, :text, :submit, :password, :enabled, :checked

jQuery
Selecting Elements

jQuery selectors start with the dollar sign and parentheses – **$()**. The factory function **$()** makes use of following three building blocks while selecting elements in a given document

1) **Tag Name-** Represents a tag name available in the DOM. For example **$('p')** selects all paragraphs <p> in the document.
2) **Tag ID -** Represents a tag available with the given ID in the DOM. For example **$('#some-id')** selects the single element in the document that has an ID of some-id.
3) **Tag Class -** Represents a tag available with the given class in the DOM. For example **$('.some-class')** selects all elements in the document that have a class of some-class.

jQuery

Selecting Elements

```
<html>
  <head>
    <title>The Selecter Example</title>
 <script src = " jquery-3.5.1.js " ></script>
  <script >
  $(document).ready(function() {
  $(“.big").css("background-color", "yellow");        });
  </script>
 </head>
 <body>
<div class = "big" id = "div1">
<p>This is first division of the DOM.</p>  </div>
<div class = "medium" id = "div2">
<p>This is second division of the DOM.</p> </div>
<div class = "small" id = "div3">
<p>This is third division of the DOM</p> </div>
</body>
</html>
```

Selecting Elements – using CSS Selectors

Multiple elements E,F,G - selects the combined results of all the specified selectors E, F or G.

Syntax

$('E, F, G,....')

where E,F,G are any valid selectors

Eg-

$('div, p') – selects all the elements matched by div or p.

$('p strong, .myclass') – selects all elements matched by strong that are descendants of an element matched by p as well as all elements that have a class of myclass.

$('p strong, #myid') – selects a single elements matched by strong that is descendant of an element matched by p as well as element whose id is myid.

jQuery

Selecting Elements – using CSS Selectors

```
<html>
  <head>
    <title>The Selector Example</title>
 <script src = " jquery-3.5.1.js " ></script>
  <script >
  $(document).ready(function() {
  $(“.big, #div3").css("background-color", "yellow");        });
  </script>
 </head>
 <body>
<div class = "big" id = "div1">
<p>This is first division of the DOM.</p>  </div>
<div class = "medium" id = "div2">
<p>This is second division of the DOM.</p> </div>
<div class = "small" id = "div3">
<p>This is third division of the DOM</p> </div>
</body>
</html>
```

Position selectors

- The **:first** selector selects the first element. This selector can only select one single element. Use the :first-child selector to select more than one element (one for each parent).

Syntax

```
$(":first")
```

- To select the last element in a group, use the :last selector. The :last-child selector is used to select more than one element (one for each parent).

Syntax

```
$(":last")
```

Position selectors

- The **:even** selector selects each element with an even index number (like: 0, 2, 4, etc.). The index numbers start at 0.

Syntax

```
$(":even")
```

- The **:odd** selector selects each element with an odd index number (like: 1, 3, 5, etc.). The index numbers start at 0.

Syntax

```
$(":odd")
```

Attribute selector

1) The **[attribute]** selector selects each element with the specified attribute.

Syntax

$("[*attribute*]") Eg- $("[id]")

2) The **[attribute=value]** selector selects each element with the specified attribute and value.

Syntax

$("[*attribute*=*value*]")

3) The **[attribute$=value]** selector selects each element with a specific attribute, with a value ending in a specific string.

Syntax

$("[*attribute*$='*value*']")

Custom selectors

The CSS selectors give us a great deal of power and flexibility to match the desired DOM elements, but sometimes we'll want to select elements based on a characteristic that the CSS specification did not anticipate.

- The **:checked** selector selects all checked checkboxes or radio buttons.

Syntax

$(":checked")

Selects all checked elements (checkboxes or radio buttons):

- The **:not()** selector selects all elements except the specified element.

This is mostly used together with another selector to select everything except the specified element in a group

Syntax

$(":not(*selector*)") Eg- $("p:not(.intro)")

Select all <p> elements except those with class="intro":

jQuery

Selecting Elements – using CSS Selectors

Selector	Description
:button	Selects any button (input[type=submit], input[type=reset], input[type=button], or button).
:checkbox	Selects only check box elements (input[type=checkbox]).
:checked	Selects only check boxes or radio buttons that are checked (supported by CSS).
:contains(foo)	Selects only elements containing the text *foo*.
:disabled	Selects only form elements that are disabled in the interface (supported by CSS).
:enabled	Selects only form elements that are enabled in the interface (supported by CSS).
:file	Selects all file elements (input[type=file]).
:header	Selects only elements that are headers; for example: <h1> through <h6> elements.
:hidden	Selects only elements that are hidden.
:image	Selects form images (input[type=image]).
:input	Selects only form elements (input, select, textarea, button).

Animating the display state of elements

Gradual transitions of a short duration help us know what's changing and *how* we got from one state to the other. And that's where the jQuery core effects come in, of which there are three sets:

- Show and hide, Toggle
- Fade in and fade out

Showing and hiding elements gradually

Syntax

$(*selector*).hide(*speed,callback*);
$(*selector*).show(*speed,callback*);
$(*selector*).toggle(*speed,callback*);

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.The optional callback parameter is a function to be executed after the hide() or show()or toggle() method completes

Fading elements into and out of existence

With jQuery you can fade elements in and out of visibility.

jQuery has the following fade methods:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

The jQuery **fadeIn()** method is used to fade in a hidden element.

Syntax

$(*selector*).fadeIn(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The jQuery **fadeOut()** method is used to fade out a visible element.

Syntax

$(*selector*).fadeOut(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The jQuery **fadeToggle()** method toggles between the fadeIn() and fadeOut() methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

Syntax

$(*selector*).fadeToggle(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The jQuery **fadeTo()** method allows fading to a given opacity (value between 0 and 1).

Syntax

$(*selector*).fadeTo(*speed,opacity,callback*);

The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The required opacity parameter in the fadeTo() method specifies fading to a given opacity (value between 0 and 1).

Eg –

```
$("button").click(function(){
$("#div1").fadeTo("slow", 0.15);
$("#div2").fadeTo("slow", 0.4);
$("#div3").fadeTo("slow", 0.7);
});
```

The jQuery **animate()** method lets you create custom animations.

Syntax

$(*selector*).animate({*params*},*speed*,*callback*);

The required params parameter defines the CSS properties to be animated.

multiple properties can be animated at the same time:

Eg –

```
$("button").click(function(){
   $("div").animate({
      left: '250px',
      opacity: '0.5',
      height: '150px',
      width: '150px'
   });
});
```

DOM manipulation methods

i)**append()** -The append() method inserts specified content at the end of the selected elements.

To insert content at the beginning of the selected elements, use the **prepend() method.

Syntax

append(content)

Appends the passed HTML fragment or elements to the content in all matched elements.

Eg –

```
$("button").click(function(){
 $("p").append("<b>Appended text</b>");
        });
```

ii) **appendTo(target)**

Moves all elements in the wrapped set to the end of the content of the specified target(s).

A number of related commands work in a fashion similar to append() and appendTo():

- **prepend() and prependTo()**—Work like append() and appendTo(), but insert the source element before the destination target's contents instead of after.
- **before() and insertBefore()**—Insert the element before the destination elements instead of before the destination's first child.
- **after() and insertAfter()**—Insert the element after the destination elements instead of after the destination's last child.

jQuery

Fetching/Returning attribute values

The **attr()** command can be used either as a read or as a write operation. The attr() command can be used to either fetch the value of an attribute from the first element in the matched set or set attribute values onto all matched elements.

Syntax

attr(name)

Obtains the values assigned to the specified attribute for the first element in the matched set.

Returns the value of the attribute for the first matched element. The value undefined is returned if the matched set is empty or the attribute doesn't exist on the first element.

jQuery

Fetching/Returning attribute values

There are two ways to set attributes onto elements in the wrapped set with jQuery.

First one allows us set a single attribute at a time (for all elements in the wrapped set).

Syntax

attr(name,value)

name (String) The name of the attribute to be set.

Value (String|Object|Function) Specifies the value of the attribute. This can be any Java- Script expression that results in a value, or it can be a function.

The **second set** variant of attr() allows us to conveniently specify multiple attributes at a time.

Syntax

$(selector).attr({attribute:value, attribute:value,...})

jQuery

Adding & removing classnames

The addClass() method adds one or more class names to the selected elements.

This method does not remove existing class attributes, it only adds one or more class names to the class attribute.

Tip: To add more than one class, separate the class names with spaces.

Syntax

addClass(names)

Adds the specified class name or class names to all elements in the wrapped set

Eg-

Add a class name to the first <p> element:

```
$("button").click(function(){
 $("p:first").addClass("intro");
    });
```

jQuery

Adding & removing classnames

The **removeClass()** method removes one or more class names from the selected elements.

Note: If no parameter is specified, this method will remove ALL class names from the selected elements.

Syntax

removeClass(names)

Removes the specified class name or class names from each element in the wrapped set

Eg –

Remove the class name "intro" from all <p> elements:

```
$("button").click(function(){
 $("p").removeClass("intro");
  });
```

jQuery

Adding & removing classnames

The toggleClass() method toggles between adding and removing one or more class names from the selected elements.
This method checks each element for the specified class names. The class names are added if missing, and removed if already set - This creates a toggle effect.

Syntax

toggleClass(name)

Adds the specified class name if it doesn't exist on an element, or removes the name from elements that already possess the class name.
Note that each element is tested individually, so some elements may have the class name added, and others may have it removed.

jQuery

Setting content

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields
- Add() - The **add()** method adds elements to an existing group of elements. This method adds elements on the whole document, or just inside context elements if the *context* parameter is specified.

jQuery

Chaining methods


```
<html>
<head>
<script src="jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#p1").css("color", "red").slideUp(2000).slideDown(2000);
  });
});
</script>
</head>
<body>
<p id="p1"><b>Advanced web is fun !!!!!</b></p>
<button>Click me</button>
</body>
</html>
```

jQuery
Callbacks

When a function simply accepts another function as an argument, this contained function is known as a Callback function. Using callback functions is a core functional programming concept either in simple functions like setInterval, event listening or when making API calls.

Callback functions can be named or anonymous functions.

Eg – anonymous function

```
setInterval(function() {
  console.log('hello!');
}, 1000);
```

// named callback functions

```
function greeting(name)
 {
  console.log(`Hello ${name}, welcome to
Functions!`);
}
```

jQuery
Callbacks

Multiple functions can be created independently and used as callback functions. These create multi-level functions. When this function tree created becomes too large, the code becomes incomprehensible sometimes and is not easily refactored. This is known as callback hell.

Eg –

```
function setInfo(name) {
  address(myAddress) {
    officeAddress(myOfficeAddress) {
      telephoneNumber(myTelephoneNumber) {
        nextOfKin(myNextOfKin) {
          console.log('done'); //let's begin to close each function!
        };
      };
    };
  };
}
```

Callback functions are useful for short asynchronous operations. When working with large sets, this is not considered best practice. Because of this challenge, Promises were introduced to simplify deferred activities.

Promises

A promise is used to handle the asynchronous result of an operation. JavaScript is designed to not wait for an asynchronous block of code to completely execute before other synchronous parts of the code can run.

Promises have three states:

- **Pending**: This is the initial state of the Promise before an operation begins
- **Fulfilled**: This means the specified operation was completed
- **Rejected**: The operation did not complete; an error value is usually thrown

The Promise object is created using the new keyword and contains the promise; this is an executor function which has a resolve and a reject callback. As the names imply, each of these callbacks returns a value with the reject callback returning an error object.

jQuery

Promises


```
const weather = true ;
const date = new Promise(function(resolve, reject)
    {
    if (weather) {
    const dateDetails = {
    name: 'Cubana Restaurant',
    location: '55th Street',
    table: 5
    };
    resolve(dateDetails)
    }
    else {
    reject(new Error('Bad weather'))
    }
    }
    );
```

```
date
 .then(function(done)
 {
console.log('We are going on a date!')
console.log(done)
 })
.catch(function(error)
 { console.log(error.message)
})
```

Using a promise that has been created is relatively straightforward; we chain .then() and .catch() to our Promise.

jQuery
Promises

.then() receives a function with an argument which is the resolve value of our promise.
.catch returns the reject value of our promise.
Note: Promises are asynchronous. Promises in functions are placed in a micro-task queue and run when other synchronous operations complete.

Chaining Promises

```
const myDate = function() {
  date
    .then(orderUber)
    .then(function(done) {
      console.log(done);
    })
    .catch(function(error) {
      console.log(error.message)
    })
}myDate();
```

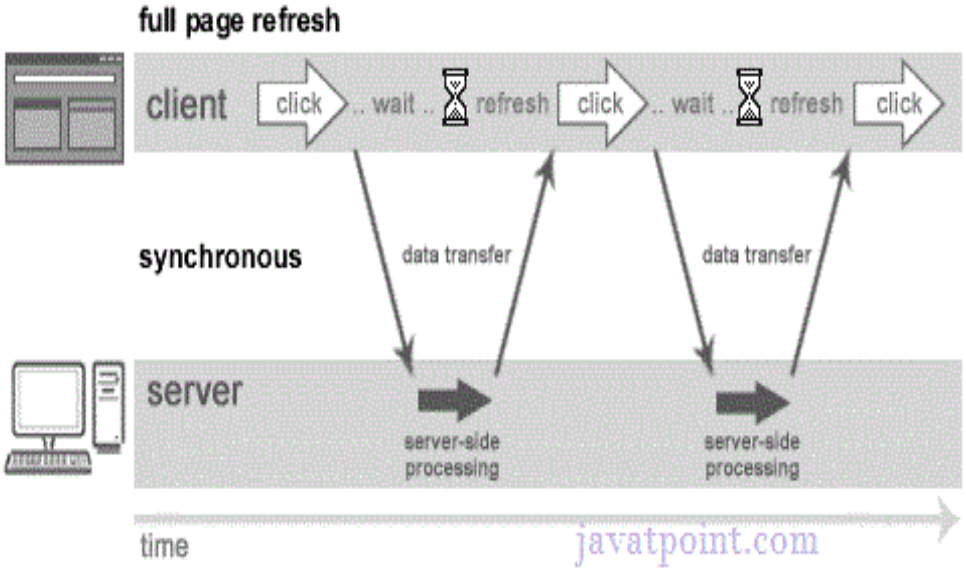
Creating another promise

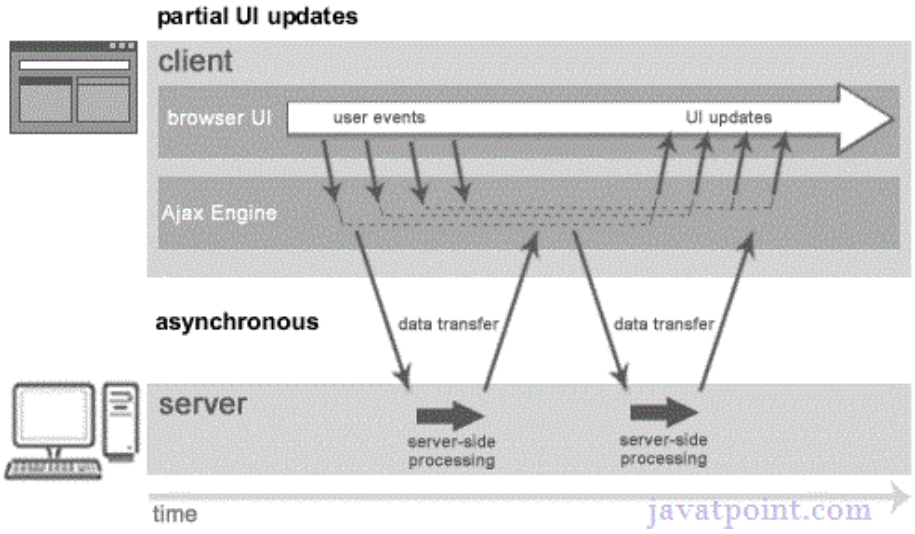
```
const orderUber = function(dateDetails)
 {
  const message = "Get me an Uber ASAP to
${dateDetails.location}, we are going on a date!";
  return Promise.resolve(message)
}
```

AJAX

A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.





jQuery
AJAX

Instead of the default method of the browser loading entire new pages, a single-page application (SPA) interacts with the web browser by dynamically rewriting the current web page with new data from the web server

- Resources are dynamically loaded and added to the page as necessary, usually in response to user actions

- The page does not reload at any point in the process, nor does it transfer control to another page

- Can be built using
    - AJAX
    - Frameworks like ReactJS, AngularJS

Traditional web applications, upon request from user like clicking a link or submitting a form, a new page is loaded with requested resources

- Asynchronous applications, upon user action, updates a part of the page without reloading the entire page
- Approaches include
    - Setting src property of iFrame or img element
    - A more elegant and complete approach is use of XHR or XMLHttpRequest object

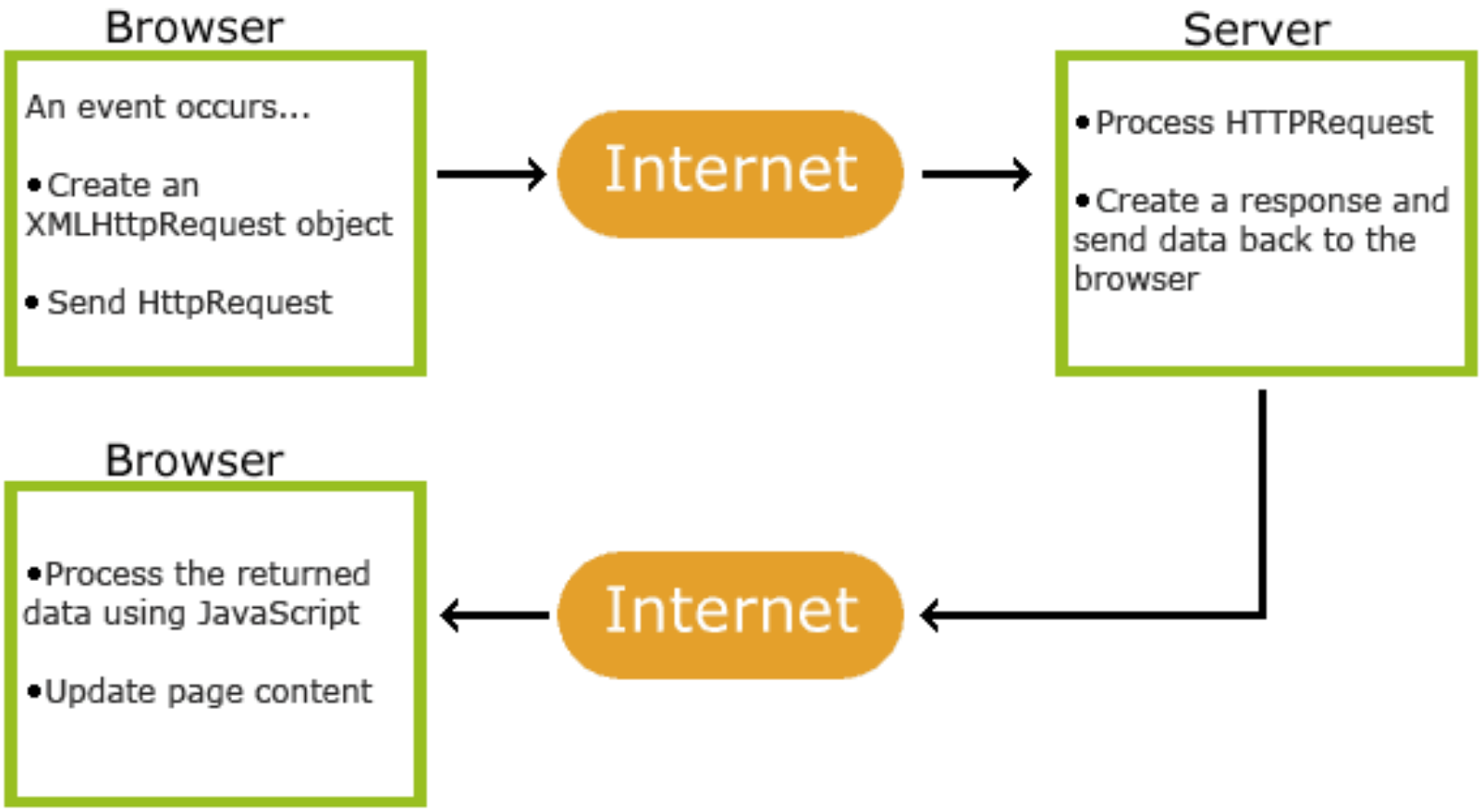
AJAX- XHR

An object of XMLHttpRequest is used for asynchronous communication between client and server.

XMLHttpRequest object plays a important role.

1.User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2.HTTP Request is sent to the server by XMLHttpRequest object.
3.Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4.Data is retrieved.
5.Server sends XML data or JSON data to the XMLHttpRequest callback function.
6.HTML and CSS data is displayed on the browser.

AJAX- XHR


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

jQuery

XHR- properties

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4.**0** UNOPENED open() is not called. **1** OPENED open is called but send() is not called. **2** HEADERS_RECEIVED send() is called, and headers and status are available. **3** LOADING Downloading data; responseText holds the data. **4** DONE The operation is completed fully.
reponseText	returns response as text.
responseXML	returns response as XML

jQuery

XHR- Methods

Method	Description
void open(method, URL)	opens the request specifying get or post method and url.
void open(method, URL, async)	same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	same as above but specifies username and password.
void send()	sends get request.
void send(string)	send post request.
setRequestHeader(header,value)	it adds request headers.


```
let xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", filepath, true);
xmlhttp.onreadystatechange=handler;
[xmlhttp.responseType=“json”|“document” |“blob”]   // default text
xmlhttp.send(null);
function handler() {
if(this.readyState == 4 && this.status == 200) {

// use this.response (json/blob) or this.responseText (text) or
// this.responseXML (document) to update a part of the page

}
}
```

jQuery

Jquery Ajax Methods

- jQuery provides methods that use XMLHttpRequest internally to make AJAX requests

- The methods are
  - $.ajax
  - $.get
  - $.post
  - $("elem").load

- In a single method call, the entire functionality of making an AJAX call using XMLHttpRequest and updating the page can be achieved

jQuery

Jquery Ajax Methods

The ajax() method is used to perform an AJAX (asynchronous HTTP) request.

All jQuery AJAX methods use the ajax() method. This method is mostly used for requests where the other methods cannot be used.

Syntax

```
$.ajax({name:value, name:value, ... })
```

The parameters specifies one or more name/value pairs for the AJAX request.

jQuery

Jquery Ajax Methods

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery-3.5.1.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $.ajax({url: "demo_test.txt",
          success: function(result){
      $("#div1").html(result);
    }});
  });
});
```

```
</script>
</head>
<body>

<div id="div1"><h2>Let jQuery AJAX Change
This Text</h2></div>

<button>Get External Content</button>

</body>
</html>
```

Another variation of ajax()

```
$.ajax('/jquery/submitData', {
type: 'POST’,                                    // http method
data: { myData: 'This is my data.' },            // data to submit
success: function (data, status, xhr) {          // success callback function
$('p').append('status: ' + status + ', data: ' + data);
},
error: function (jqXhr, textStatus, errorMessage) {
$('p').append('Error' + errorMessage);
}
});
```

jQuery

Jquery Ajax Methods

The $.get() method requests data from the server with an HTTP GET request.

Syntax:

```
$.get(URL,callback);
```

The required URL parameter specifies the URL you wish to request.

The optional callback parameter is the name of a function to be executed if the request succeeds.

$.get

Syntax

- $.get(url, [data],[callback]);

- Example
  - $.get('/jquery/getjsondata’, // url
    {name:'Steve’}, // request parameters
    function (data, textStatus, jqXHR) { // success callback function
    $('p').append(data.firstName);
    }
    );

- Other Variants
- $.getJSON(url, [data],[callback]);
- $.getScript(url, [data],[callback]);

jQuery

Jquery Ajax Methods

The $.post() method requests data from the server using an HTTP POST request.

Syntax:

```
$.post(URL,data,callback);
```

The required URL parameter specifies the URL you wish to request.

The optional data parameter specifies some data to send along with the request.
The optional callback parameter is the name of a function to be executed if the request succeeds.

$.post

Syntax

- $.post(url,[data],[callback],[type]);

- Example
  - $.post('/jquery/submitJSONData', // url
    { myData: 'This is my data.' }, // data to be submitted
    function(data, status, jqXHR) { // success callback function
    $('p').append('status: ' + status + ', data: ' + data);
    },
    "json" //response type
    );

- Internally uses $.ajax with method="post"

jQuery

Jquery Ajax Methods

The load(url, data, callback) method loads data from the server and places the returned HTML into the matched element.

Syntax

[selector].load(url, [data], [callback])

Parameters

url − A string containing the URL to which the request is sent.
data − This optional parameter represents a map of data that is sent with the request.
callback − This optional parameter represents a function that is executed if the request succeeds

Jquery Ajax Methods

Eg -

```
<script>
        $(document).ready(function() {
          $("#driver").click(function(event){
            $('#stage').load('result.html');
          });
        });
    </script>
```