UNIT II

1.JAVA SCRIPT: AN INTRODUCTION TO JAVASCRIPT

JavaScript is a versatile, dynamically typed programming language used for interactive web applications, supporting both client-side and server-side development, and integrating seamlessly with HTML, CSS, and a rich standard library.

- JavaScript is a single-threaded language that executes one task at a time.
- It is an Interpreted language which means it executes the code line by line.
- The data type of the variable is decided at run-time in JavaScript that's why it is called dynamically typed.

"Hello, World!" Program in Browser Console

A "Hello, World!" program is the simplest way to get started with any programming language. Here's how you can write one using JavaScript.

```
<html>
<head></head>
<body>
   <h1>Check the console for the message!</h1>
   <script>
      // This is our first JavaScript program
      console.log("Hello, World!");
   </script>
</body
</html>
```

In this example

- The**<script>** tag is used to include JavaScript code inside an HTML document.
- **console.log()** prints messages to the browser's developer console. Open the browser console to see the "Hello, World!" message.

2.HTML DOM (Document Object Model)

The HTML DOM (Document Object Model) is a programming interface that represents the structure of a web page in a way that programming languages like JavaScript can understand and manipulate.

Think of it as a tree of objects where each part of your HTML document (elements, attributes, text) is represented as a node, allowing you to dynamically change or interact with the content and structure of the page.

Imagine your webpage as a tree

- The document is the root.
- HTML tags like <html>, <head>, and <body> are branches.
- Attributes, text, and other elements are the leaves.

2.1 The DOM is essential because

- **Dynamic Content Updates:** Without reloading the page, the DOM allows content updates (e.g., form validation, AJAX responses).
- **User Interaction:** It makes your webpage interactive (e.g., responding to button clicks, form submissions).
- **Flexibility:** Developers can add, modify, or remove elements and styles in real-time.
- **Cross-Platform Compatibility:** It provides a standard way for scripts to interact with web documents, ensuring browser compatibility.

2.2 How the DOM Works?

The DOM connects your webpage to JavaScript, allowing you to:

- Access elements (like finding an <h1> tag).
- Modify content (like changing the text of a <p> tag).
- React to events (like a button click).
- Create or remove elements dynamically.

2.3 Properties of the DOM

- **Node-Based:** Everything in the DOM is represented as a node (e.g., element nodes, text nodes, attribute nodes).
- **Hierarchical**: The DOM has a parent-child relationship, forming a tree structure.
- **Live:** Changes made to the DOM using JavaScript are immediately reflected on the web page.
- **Platform-Independent:** It works across different platforms, browsers, and programming languages.

2.4 Commonly Used DOM Methods

Methods	Description
getElementById(id)	Selects an element by its ID.
getElementsByClassName(class)	Selects all elements with a given class.

Methods	Description
querySelector(selector)	Selects the first matching element.
querySelectorAll(selector)	Selects all matching elements.
createElement(tag)	Creates a new HTML element.
appendChild(node)	Adds a child node to an element.
remove()	Removes an element from the DOM.
addEventListener(event, fn)	Attaches an event handler to an element.

Example: In this example, We use HTML element id to find the DOM HTML element.

```
<html>
<body>
    <h2>GeeksforGeeks</h2>
    <!-- Finding the HTML Elements by their Id in DOM -->
    <p id="intro">
        A Computer Science portal for geeks.
    </p>
    <p>
        This example illustrates the <b>getElementById</b> method.
    </p>
    <p id="demo"></p>
    <script>
        const element = document.getElementById("intro");
        document.getElementById("demo").innerHTML =
            "GeeksforGeeks introduction is: " + element.innerHTML;
    </script>
```

</body>

</html>

Output:

GeeksforGeeks

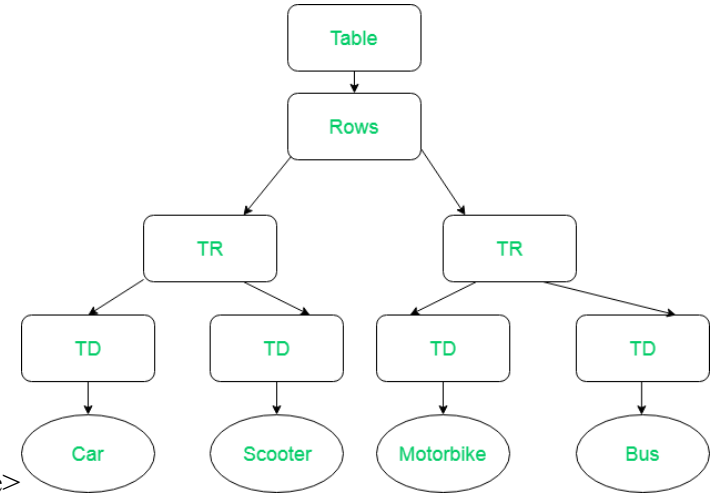
A Computer Science portal for geeks.

This example illustrates the **getElementById** method.

GeeksforGeeks introduction is: A Computer Science portal for geeks.

Example : This example describes the representation of the HTML elements in the tree structure.

```
<table>
   <ROWS>
      <tr>
         <td>Car</td>
         <td>Scooter</td>
      </tr>
      <tr>
         <td>MotorBike</td>
         <td>Bus</td>
      </tr>
   </ROWS>
</table>
```



HTML elements in tree-like structure

2.5 Levels of DOM

DOM consisted of multiple levels, each representing different aspect of the document.

- **Level 0:** Provides a low-level set of interfaces.
- **Level 1:** DOM level 1 can be described in two parts: **CORE** and **HTML**.

- **CORE** provides low-level interfaces that can be used to represent any structured document.
- **HTML** provides high-level interfaces that can be used to represent HTML documents.

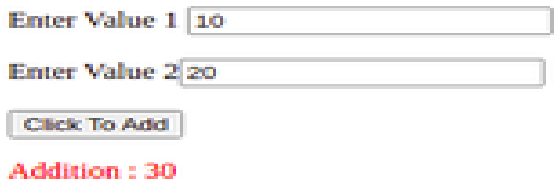
- **Level 2:** consists of six specifications: **CORE2**, **VIEWS**, **EVENTS**, **STYLE**, **TRAVERSAL**, and **RANGE**.
  - **CORE2**: extends the functionality of CORE specified by DOM level 1.
  - **VIEWS**: views allows programs to dynamically access and manipulate the content of the document.
  - **EVENTS**: Events are scripts that are either executed by the browser when the user reacts to the web page.
  - **STYLE**: allows programs to dynamically access and manipulate the content of style sheets.
  - **TRAVERSAL**: This allows programs to dynamically traverse the document.
  - **RANGE**: This allows programs to dynamically identify a range of content in the document.
- **Level 3:** consists of five different specifications: **CORE3**, **LOAD and SAVE**, **VALIDATION**, **EVENTS**, and **XPATH**.
  - **CORE3**: extends the functionality of CORE specified by DOM level 2.
  - **LOAD and SAVE**: This allows the program to dynamically load the content of the XML document into the DOM document and save the DOM Document into an XML document by serialization.
  - **VALIDATION**: This allows the program to dynamically update the content and structure of the document while ensuring the document remains valid.
  - **EVENTS**: extends the functionality of Events specified by DOM Level 2.
  - **XPATH**: XPATH is a path language that can be used to access the DOM tree.

Example: This example illustrates the dom-manipulation using *getElementById()* Method.

```
<html>
<head></head>
<body>
   <label>Enter Value 1: </label>
   <input type="text" id="val1" />
   <br />
```

```
    <br />
    <label>Enter Value 2: </label>
    <input type=".text" id="val2" />
    <br />
    <button onclick="getAdd()">Click To Add</button>
    <p id="result"></p>
    <script type="text/javascript">
        function getAdd() {
            // Fetch the value of input with id val1
            const num1 = Number(document.getElementById("val1").value);
            // Fetch the value of input with id val2
            const num2 = Number(document.getElementById("val2").value)
            const add = num1 + num2
            console.log(add)
            // Displays the result in paragraph using do
            document.getElementById("result").innerHTML = "Addition : " + add
            // Changes the color of paragraph tag with re
            document.getElementById("result").style.color = "red"
        }
    </script>
</body>
</html>
```

Output:



3.JavaScript Date and Objects

The Date Object in JavaScript helps you work with dates and times. A date can be created using the new Date(). It can handle dates and times with very precise milliseconds and can represent dates up to 100 million days from January 1, 1970, all the way to the year 275755.

With the Date object, you can easily get and set parts of the date, like the year, month, day, hour, minute, second, and millisecond, in local or UTC/GMT time. There are four main ways to create a Date object, but the most common is using a new Date().

Syntax:

new Date()
new Date(milliseconds)
new Date(dataString)
new Date(year, month, date, hour, minute, second, millisecond)

3.1 new Date()

The **Date()** constructor creates a Date object which sets the current date and time depend on the browser's time zone. It does not accepts any value.

Example: This example creates a new date objects and prints the date on the console.

```
// "newDate" is a Date object

let newDate = new Date();

// Prints today's date to the console

console.log(newDate);
```

Output

2024-10-08T07:42:51.861Z

3.2 new Date(milliseconds)

This method accepts single parameter **milliseconds** which indicates any **numeric** value. This argument is taken as the internal numeric representation of the date in milliseconds.

Example: In this example we pass a parameter in milliseconds.

```
// "newDate" is a Date object

let newDate = new Date(4500);

// Prints the date to the console

console.log("Today's date: " + newDate);
```

Output

Today's date: Thu Jan 01 1970 00:00:04 GMT+0000 (Coordinated Universal Time)

3.3 new Date(datastring)

This method accepts single parameter **dataString** which indicates any **string** value. It is a string representation of a date and return the datastring with the day.

Example: Here we add a string value as a parameter.

```
// "newDate" is a Date object
```

```
let newDate = new Date("October 13, 2013 11:13:00");
// Prints the date string to the console
console.log("Date string with day: " + newDate);
```

Output

Date string with day: Sun Oct 13 2013 11:13:00 GMT+0000 (Coordinated Universal Time)

3.4 new Date(year, month, date, hour, minute, second, millisecond)

This method accepts seven parameters as mentioned above and described below:

Example: This example shows the implementation of the above approach

```
// "newDate" is a Date object
let newDate = new Date(2014, 10, 24, 10, 33, 30, 0);
// Prints the date to the console
console.log(newDate);
```

Output

2014-11-24T10:33:30.000Z

3.5 Date Object Methods:

Here are some methods that define the usage of a Date object, These are non-static methods.

Below methods are returns all the values according to local time:

Method	Description
Date()	It returns presents day's date and time.
getDate()	It returns the day for the specified date.
getDay()	It returns the day of the week for the specified date.
getFullYear()	It returns the year of the specified date.
getYear()	This method returns the year in the specified date.
getHours()	It returns the hour in a specified date.

Method	Description
getMilliseconds()	It returns the milliseconds in the specified date.
getMinutes()	It returns the minutes in the specified date.
getMonth()	It returns the month in the specified date. This also find the month.
getSeconds()	This method returns the seconds in the specified date.
getTime()	This method returns the date in terms of numeric value as milliseconds.
setDate()	This method sets the day of the month for a specified date.
setFullYear()	This method sets the full year for a specified date.

Below methods are returns all the values according to universal time:

Methods	Description
getUTCDate()	It returns the day of a month for a specified date.
getUTCDay()	It returns the day of the week for a specified date.
getUTCFullYear()	This method returns the year for a specified date.
getUTCHours()	It returns the hours in a specified date.
getUTCMilliseconds()	This method returns the milliseconds form for a specified date.

Methods	Description
getUTCMinutes()	This method returns the minutes in a specified date.
getUTCMonth()	This method returns the month for a specified date.

4.JavaScript RegExp (Regular Expression)

A **regular expression** is a special sequence of characters that defines a search pattern, typically used for pattern matching within text. It's often used for tasks such as validating email addresses, phone numbers, or checking if a string contains certain patterns (like dates, specific words, etc.).

In JavaScript, RegExp is an object that is used to create regular expressions, which are patterns used to match character combinations in strings.

4.1 Creating a RegExp in JavaScript

There are two primary ways to create a RegExp in JavaScript

1. Using the RegExp Constructor

The RegExp constructor is useful when you need to create a regular expression dynamically, such as when the pattern or flags might change based on user input or other condition

```
let pattern = "hello";  // Pattern to match
let flags = "i";  // Case-insensitive flag
let regex = new RegExp(pattern, flags);
let s = "Hello world";
console.log(regex.test(s));
```

Output

True

2.Using Regular Expression Literal

This is the most common and simple way to create a regular expression in JavaScript. It's especially useful when the pattern is static (doesn't change dynamically).

```
let regex = /hello/i;
let s = "Hello world";
console.log(regex.test(s));
```

Output

True

Expressions	Descriptions
g	Find the character globally
i	Find a character with case-insensitive matching
m	Find multiline matching

Regular Expression Brackets can Find characters in a specified range

Expressions	Description
[abc]	Find any of the characters inside the brackets
[^abc]	Find any character, not inside the brackets
[0-9]	Find any of the digits between the brackets 0 to 9
[^0-9]	Find any digit not in between the brackets
(x | y)	Find any of the alternatives between x or y separated with |

Regular Expression Metacharacters are characters with a special meaning:

Metacharacter	Description
\.	Search single characters, except line terminator or newline.
\w	Find the word character i.e. characters from a to z, A to Z, 0 to 9
\d	Find a digit
\D	Search non-digit characters i.e all the characters except digits
\s	Find a whitespace character
\S	Find the non-whitespace characters.
\b	Find a match at the beginning or at the end of a word

Metacharacter	Description
\B	Find a match that is not present at the beginning or end of a word.
\0	Find the NULL character.
\n	Find the newline character.
\f	Find the form feed character
\r	Find the carriage return character
\t	Find the tab character
\v	Find the vertical tab character
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxxx

Regular Expression Quantifiers are used to define quantitiesoccurrence

Quantifier	Description
n+	Match any string that contains at least one n
n*	Match any string that contains zero or more occurrences of n
n?	Match any string that contains zero or one occurrence of n
m{X}	Find the match of any string that contains a sequence of m, X times
m{X, Y}	Find the match of any string that contains a sequence of m, X to Y times
m{X,}	Find the match of any string that contains a sequence of m, at least X times
m$	Find the match of any string which contains m at the end of it
^m	Find the match of any string which contains m at the beginning of it
?!m	Find the match of any string which is not followed by a specific string m.

Regular Expression Object Properties:

Property	Description
constructor	Return the function that created the RegExp object's prototype
global	Specify whether the **"g"** modifier is set or not
ignorecase	Specify whether the **"i"** modifier is set or not
lastindex	Specify the index at which to start the next match
multiline	Specify whether the **"m"** modifier is set or not
source	Return the text of RegExp pattern

Regular Expression Object Methods:

Method	Description
compile()	Used to compile the regular expression while executing of script
exec()	Used to test for the match in a string.
test()	Used to test for a match in a string
toString()	Return the string value of the regular expression

5.Javascript Error and Exceptional Handling With Examples

In JavaScript, errors can occur due to programming mistakes, incorrect user input, etc. Errors can disrupt code execution and lead to bad user experience. Effective error & exception handling is required for building robust, reliable and user friendly applications in JavaScript.
What is an Error?
An error is an event that occurs during the execution of a program that prevents it from continuing normally. Errors can be caused by a variety of factors, such as **syntax** errors, **runtime** errors, and **logical** errors.

5.1 Syntax Errors

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.
For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script>
  window.print();
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected, and the rest of the code in other threads gets executed, assuming nothing in them depends on the code containing the error.

5.2 Runtime Errors (Exceptions)

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because the syntax is correct here, but at runtime, it is trying to call a method that does not exist.

```
<script>
  window.printme();
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

There are many JavaScript runtime errors (exceptions), some are as follows −

- **ReferenceError** − Trying to access an undefined variable/ method.
- **TypeError** − Attempting an operation on incompatible data types.
- **RangeError** − A value exceeds the allowed range.

5.3 Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and do not get the expected result.

For example, when you divide any numeric value with 10, it returns undefined.

```
<script>
  let num = 10/0;
</script>
```

Exceptional Handling

Whenever any error occurs in the JavaScript code, the JavaScript engine stops the execution of the whole code. If you handle such errors in the proper way, you can skip the code with errors and continue to execute the other JavaScript code.

You can use the following mechanisms to handle the error.

- try...catch...finally statements
- throw statements
- the onerror() event handler property
- Custom Errors

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax −

```
<script>
  try {
    // Code to run
    [break;]
  }
  catch ( e ) {
    // Code to run if an exception occurs
    [break;]
  }
  [ finally {
    // Code that is always executed regardless of
    // an exception occurring
  }]
</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both). When an exception occurs in the try block, the exception is placed in e and the catch block is executed. The optional finally block executes unconditionally after try/catch.

Example

Here is an example where we are trying to call a non-existing function which in turn is raising an exception.

Let us try to catch this exception using try...catch and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

You can use finally block which will always execute unconditionally after the try/catch.

Open Compiler

```
<html>
<head>
<script>
  try {
    var a = 100;
    alert(myFunc(a));
  }
  catch (e) {
    alert(e);
  }
  finally {
    alert("Finally block will always execute!" );
  }
</script>
</head>
<body>
  <p>Exception handling using try...catch...finally statements</p>
</body>
</html>
```

Output

Exception handling using try...catch...finaly statements

6.JavaScript Validation

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

6.1Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
  // Form validation code will come here.
  function validate() {

    if( document.myForm.Name.value == "" ) {
      alert( "Please provide your name!" );
      document.myForm.Name.focus() ;
      return false;
    }
    if( document.myForm.EMail.value == "" ) {
      alert( "Please provide your Email!" );
      document.myForm.EMail.focus() ;
      return false;
    }
    if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value )
||
      document.myForm.Zip.value.length != 5 ) {

      alert( "Please provide a zip in the format #####." );
      document.myForm.Zip.focus() ;
      return false;
    }
    if( document.myForm.Country.value == "-1" ) {
      alert( "Please provide your country!" );
      return false;
```

```
      }
      return( true );
    }
  </script>
```

6.2 Data Format Validation

- Now we will see how we can validate our entered form data before submitting it to the web server.
- The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

- Example
- Try the following code for email validation.

```
<script type = "text/javascript">
  function validateEmail() {
    var emailID = document.myForm.EMail.value;
    atpos = emailID.indexOf("@");
    dotpos = emailID.lastIndexOf(".");

    if (atpos < 1 || ( dotpos - atpos < 2 )) {
      alert("Please enter correct email ID")
      document.myForm.EMail.focus() ;
      return false;
    }
    return( true );
  }
</script>
```

7.JavaScript Built-in Objects

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects

- Objects are always objects

JavaScript Primitives

- A primitive value is a value that has no properties or methods.
- A primitive data type is data that has a primitive value.
- JavaScript defines 5 types of primitive data types:
  - String
  - Number
  - Boolean
  - Null
  - undefined
- Primitive values are immutable (they are hardcoded and therefore cannot be changed).
- if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

7.1 Object Methods

- Methods are actions that can be performed on objects.
- Object properties can be both primitive values, other objects, and functions.
- An object method is an object property containing a function definition.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript objects are containers for named values, called properties and methods.

7.2 Creating Objects:

In JavaScript, Objects can be created using two different methodologies namely Literal Form and Constructed Form.

Literal Form: The literal form uses the construction of **object literals** that can be said as a collection of key-value pairs enclosed within a pair of curly braces. The syntaxial form is shown below.

```
let obj = {
   key1: value1,
   key2: value2,
   ...
};
```

Constructed Form: The Constructed form uses either an object constructor function or the new keyword to create an empty object ad then adds properties to the object one by one. The syntaxial forms are shown below.

Object Constructor Function: In this methodology, the user creates an explicit function to take required values as parameters and assign them as the properties of the desired object.

```
function obj(value1, value2, ...) {
   this.key1 = value1;
   this.key2 = value2;
   ...
}
```

Using New Keyword: This methodology uses the New keyword in front of any constructor method or any built-in constructor method (such as Object, Date, String, etc) and creates a new instance of the following object by mounting it on memory.

```
let obj = new Object();
obj.key1 = value1;
obj.key2 = value2;
...
```

8.EVENT HANDLING

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- 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.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- 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.

8.1 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

Example

```
<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>
```

8.2 onsubmit Event Type

- onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function validation() {
          all validation goes here
          .........
          return either true or false
        }
      //-->
    </script>
  </head>

  <body>
    <form method = "POST" action = "t.cgi" onsubmit = "return validate()">
      .......
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

9.DHTML Introduction

DHTML, or Dynamic HTML, is a technology that differs from traditional HTML. DHTML combines HTML, CSS, JavaScript, and the Document Object Model (DOM) to create dynamic content. It uses the Dynamic Object Model to modify settings, properties, and methods. Scripting is also an essential component of DHTML, which was part of earlier computing trends. It is supported by some versions of Netscape Navigator and Internet Explorer 4.0 and higher.

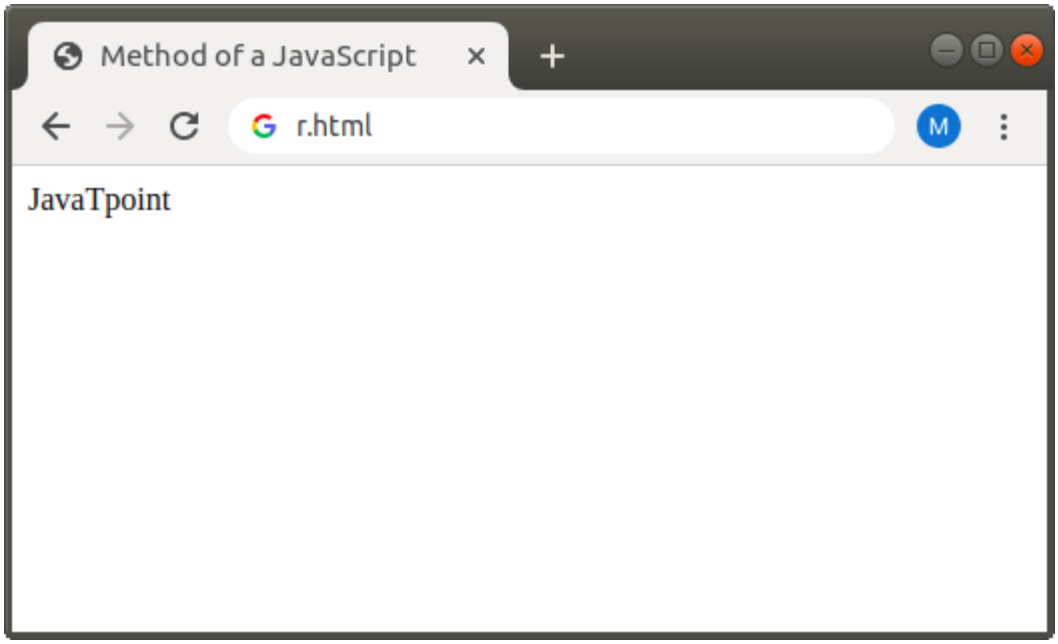
HTML: HTML stands for Hypertext Markup Language and it is a client-side markup language. It is used to build the block of web pages.
Javascript: It is a Client-side Scripting language. Javascript is supported by most of browsers, and also has cookie collection to determine the user's needs.
CSS: The abbreviation of CSS is Cascading Style Sheet. It helps in the styling of the web pages and helps in designing of the pages. The CSS rules for DHTML will be modified at different levels using JS with event handlers which adds a significant amount of dynamism with very little code.
DOM: It is known as a Document Object Model which acts as the weakest link in it. The only defect in it is that most of the browsers does not support DOM. It is a way to manipulate the static content.

9.1DHTML JavaScript

JavaScript can be included in HTML pages, which creates the content of the page as dynamic. We can easily type the JavaScript code within the <head> or <body> tag of a HTML page. If we want to add the external source file of JavaScript, we can easily add using the <src> attribute.
Following are the various examples, which describes how to use the JavaScript technology with the DHTML:
Document.write() Method
The document.write() method of JavaScript, writes the output to a web page.
Example 1: The following example simply uses the **document.write()** method of JavaScript in the DHTML. In this example, we type the JavaScript code in the **<body>** tag.

```
1. <HTML>
2. <head>
3. <title>
4. Method of a JavaScript
5. </title>
6. </head>
7. <body>
8. <script type="text/javascript">
9. document.write("JavaTpoint");
10. </script>
11. </body>
12. </html>
```

Test it Now
Output:

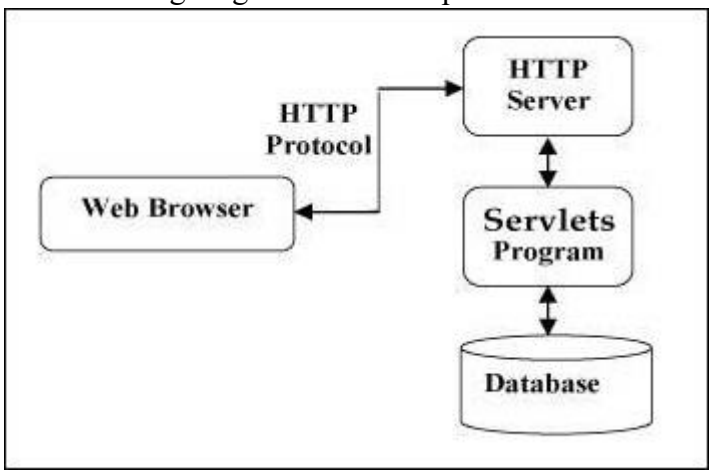


10.Java Servlets Architecture

Java Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, and then send a response back to the web server.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks −

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.

- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

11.Life Cycle of a Servlet

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it. So, before creating a Servlet object, let's first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.
Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

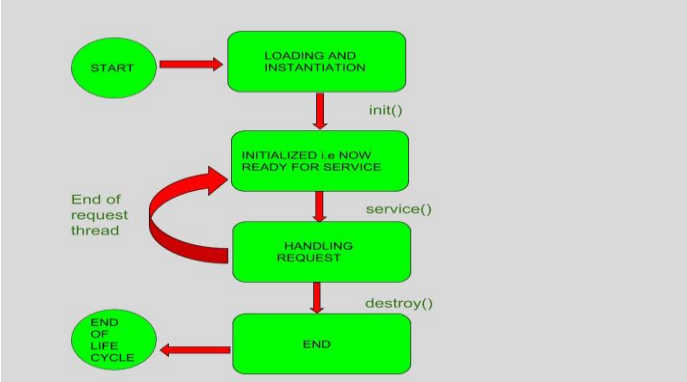
- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.

Let's look at each of these stages in details:

1. **Loading a Servlet**: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :
    - Initializing the context, on configuring the Servlet with a zero or positive integer value.
    - If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

- **Loading :** Loads the Servlet class.
- **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



2. **Initializing a Servlet**: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.
Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

3. **Handling request**: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :
    - It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request, then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.
    - After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.

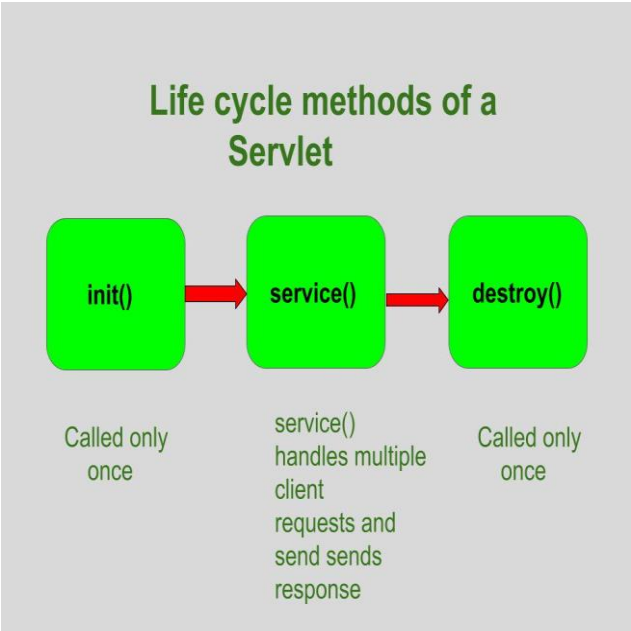
4. **Destroying a Servlet**: When a Servlet container decides to destroy the Servlet, it performs the following operations,
    - It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
    - After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Servlet Life Cycle Methods

There are three life cycle methods of a Servlet :

- init()
- service()
- destroy()



Let's look at each of these methods in details:

1. **init() method**: The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

```
2. //init() method
3.
4. public class MyServlet implements Servlet{
5.    public void init(ServletConfig config) throws ServletException {
6.        //initialization code
7.    }
8.     //rest of code
9. }
```

10. **service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.
    - This method uses **ServletRequest** object to collect the data requested by the client.
    - This method uses **ServletResponse** object to generate the output content.

```
11. // service() method
12.
13. public class MyServlet implements Servlet{
14.    public void service(ServletRequest res, ServletResponse res)
15.    throws ServletException, IOException {
16.          // request handling code
17.    }
18.    // rest of code
19. }
```

20. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

```
21. //destroy() method
22.
23. public void destroy()
```

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

12.GET method and POST method

12.1 GET method type and doGet() method

The doGet() method in servlets is used to process the HTTP GET requests. So, basically, the HTTP GET method should be used to get the data from the server to the browser. Although in some requests, the GET method is used to send data from the browser to the server also. In this case, you need to understand the below points on how the GET method will work.

- The data that is being submitted to the server will be visible in the URL using query parameters like this "*http://localhost:8080/HelloServlet/hello?myParam=myValue*".
- So, if you are sending any sensitive information like passwords, you should not use the GET method as the data entered can be clearly visible in the browser URL.

GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
  <body>
    <form action = "HelloForm" method = "GET">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

First Name: Last Name: Submit

12.2 POST method type and doPost() method

The doPost() method in servlets is used to process the HTTP POST requests. It is used to submit the data from the browser to the server for processing. The data submitted with POST method type is sent in the message body so it is secure and cannot be seen in the URL. And there is no limit on the data that can be sent through the POST method. Ideally, we need to use the POST method, to send the form data to the webserver. So, we will be using the doPost() method in this example. And we will learn how to handle some of the common HTML fields data such as *Text field, Checkbox, Radio button, Dropdown*, etc., values in the servlet.

```
<html>
  <body>
    <form action = "HelloForm" method = "POST">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

First Name: Last Name: Submit

13.Servlet – Session Tracking

HTTP is a "stateless" protocol, which means that each time a client requests a Web page, the client establishes a new connection with the Web server, and the server does not retain track of prior requests.

- The conversation of a user over a period of time is referred to as a **session**. In general, it refers to a certain period of time.
- The recording of the object in session is known as **tracking**.
- **Session tracking** is the process of remembering and documenting customer conversations over time. Session management is another name for it.
- The term "**stateful web application**" refers to a web application that is capable of remembering and recording client conversations over time.

Why is Session Tracking Required?

- Because the HTTP protocol is stateless, we require Session Tracking to make the client-server relationship stateful.
- Session tracking is important for tracking conversions in online shopping, mailing applications, and E-Commerce applications.
- The HTTP protocol is stateless, which implies that each request is treated as a new one. As you can see in the image below.

13.1 Session Tracking employs Four Different techniques

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

A. Cookies

Cookies are little pieces of data delivered by the web server in the response header and kept by the browser. Each web client can be assigned a unique session ID by a web server. Cookies are used to keep the session going. Cookies can be turned off by the client.

B. Hidden Form Field

The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user's view

C. URL Rewriting

With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.

D. HttpSession

A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.