1.What is meaning of Synchronous request and Asynchronous request? Mention the working of AJAX

Synchronous vs. Asynchronous Requests

Synchronous Request:

- A synchronous request is one where the code execution waits until the server returns the response.

- During this waiting period, the application is essentially "blocked" from performing any other operations.

- This can lead to a poor user experience as the user interface may freeze until the request completes.



Asynchronous Request:

- An asynchronous request allows the application to continue executing other code while waiting for the server's response.

- This is achieved by using callbacks, promises, or async/await in modern JavaScript.

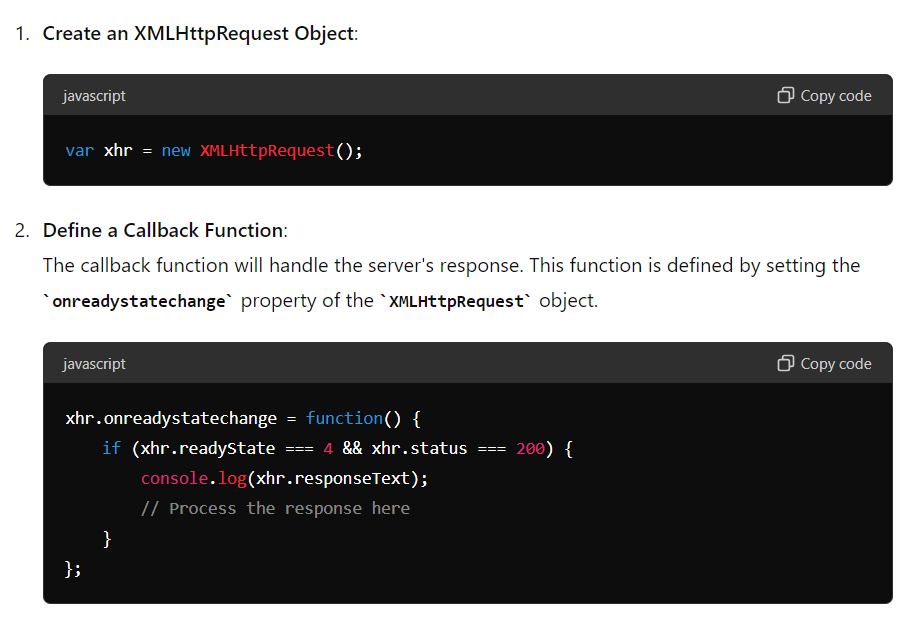
- This approach leads to a more responsive and better user experience.

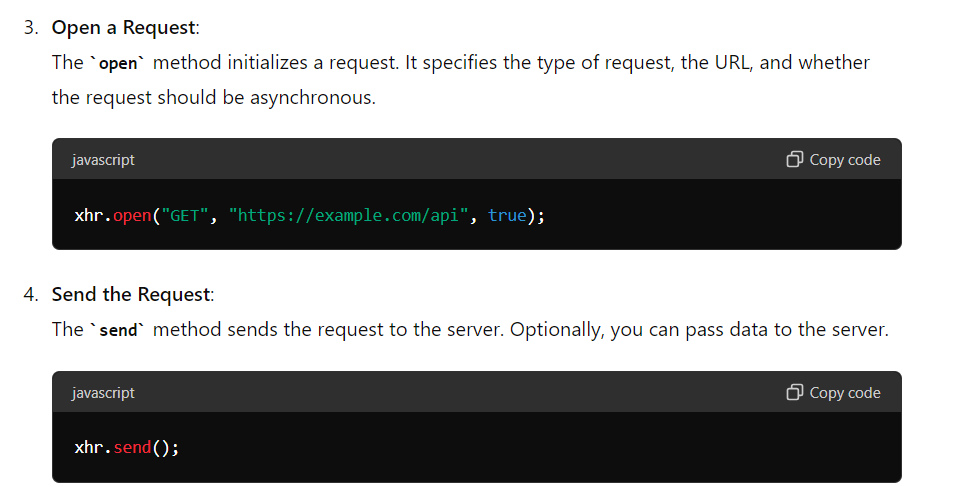


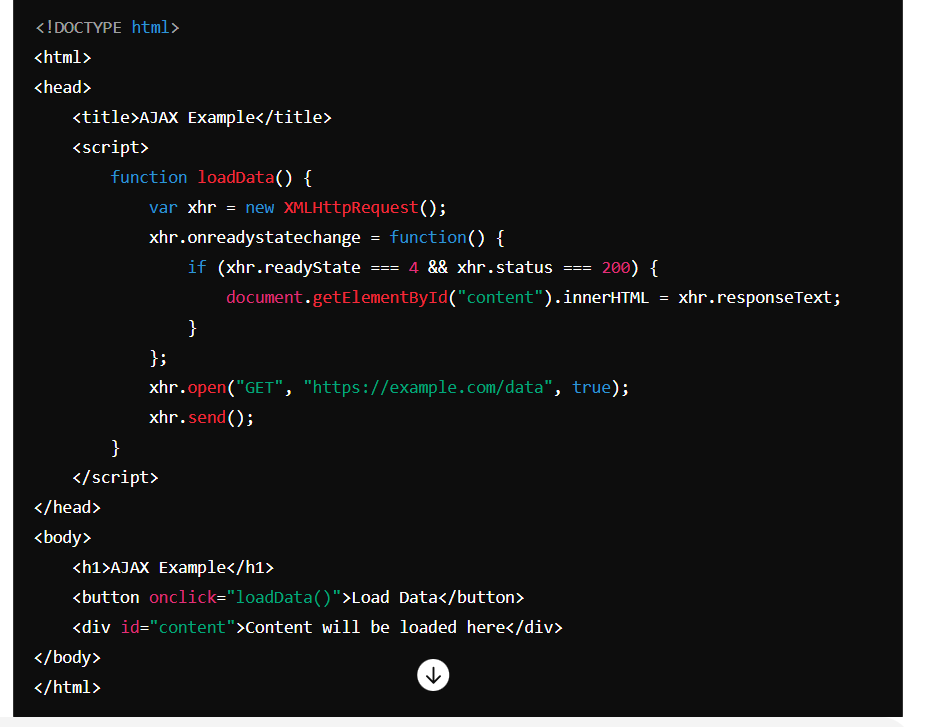
Working of AJAX (Asynchronous JavaScript and XML)

AJAX is a technique for creating fast and dynamic web pages. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works:







Key Points:

- XMLHttpRequest Object: Used to interact with servers.

- open(method, url, async): Prepares the request. The `async` parameter determines whether the request is asynchronous.

- onreadystatechange: An event handler for the `readystatechange` event, which is triggered every time the `readyState` property of the `XMLHttpRequest` changes.

- readyState: Holds the status of the `XMLHttpRequest`. Values range from 0 to 4:

- 0: UNSENT

- 1: OPENED

- 2: HEADERS\_RECEIVED

- 3: LOADING

- 4: DONE

- status: The HTTP status code of the response.

- send(): Sends the request to the server.

By using AJAX, you can create dynamic and interactive web applications where parts of the page can be updated without needing to reload the entire page. This results in a smoother and more responsive user experience.

2.Write the ways using that we can deploy an application. Write the steps.

Sure, I'll explain the deployment processes step-by-step without code.

1. Manual Deployment

Steps:

1. Prepare the Application:

- Ensure your application is fully tested and ready for deployment.

- Package your application files, ensuring all dependencies are included.

2. Upload Files to the Server:

- Use an FTP client or SCP to transfer your application files from your local machine to the server.

- Place the files in the appropriate directory on the server.

3. Configure the Server:

- Log in to the server.

- Configure the web server (e.g., Apache, Nginx) to serve your application.

- Edit the configuration files as necessary to point to your application directory and set up domain names, ports, etc.

4. Set File Permissions:

- Adjust file and directory permissions to ensure the web server has the necessary access.

- Ensure sensitive files are protected.

5. Restart the Web Server:

- Restart the web server to apply the changes.

- Check the server logs to ensure the application is running correctly.

2. Automated Deployment

Steps:

1. Version Control Setup:

- Ensure your application code is stored in a version control system like Git.

- Create a repository on a platform such as GitHub, GitLab, or Bitbucket.

2. Write Deployment Scripts:

- Write scripts to automate the deployment process. These scripts can handle tasks like installing dependencies, running tests, and deploying files.

3. Use CI/CD Tools:

- Set up Continuous Integration/Continuous Deployment (CI/CD) tools like Jenkins, GitLab CI/CD, Travis CI, or GitHub Actions.

- Create a pipeline that defines the steps to build, test, and deploy your application.

4. Configure Deployment Triggers:

- Set up triggers in the CI/CD tool to initiate the deployment process automatically upon code changes (e.g., pushing to a specific branch).

5. Deploy the Application:

- Run the deployment pipeline.

- Monitor the pipeline for any errors and ensure the application is deployed successfully.

3. Containerization and Orchestration

Steps:

1. Containerize the Application:

- Create a container image of your application using Docker or another containerization tool.

- Store the container image in a container registry (e.g., Docker Hub, Amazon ECR).

2. Set Up Orchestration Tool:

- Use an orchestration tool like Kubernetes to manage your containerized applications.

- Define the desired state of your application, including replicas, services, and networking, using Kubernetes configuration files.

3. Deploy Containers:

- Use the orchestration tool to deploy your containers.

- Monitor the deployment process and ensure the containers are running as expected.

4. Manage Scaling and Updates:

- Use the orchestration tool to scale your application up or down based on demand.

- Deploy updates to your application by updating the container image and redeploying.

4. Serverless Deployment

Steps:

1. Develop Serverless Functions:

- Write your application logic as serverless functions using a serverless framework or service like AWS Lambda, Google Cloud Functions, or Azure Functions.

2. Configure Serverless Framework:

- Set up a serverless framework to manage your serverless functions, define resources, and handle deployment.

- Configure the framework to specify triggers, such as HTTP requests or events, that invoke your functions.

3. Deploy to Serverless Platform:

- Use the serverless framework to deploy your functions to the cloud provider.

- Monitor the deployment process to ensure the functions are correctly deployed and configured.

4. Set Up API Gateway:

- If necessary, configure an API Gateway to expose your serverless functions as HTTP endpoints.

- Define routing, request validation, and security settings for the API Gateway.

5. Monitor and Maintain:

- Use monitoring tools provided by the cloud provider to track the performance and usage of your serverless functions.

- Implement logging and error handling to manage issues and optimize performance.

These steps provide a high-level overview of various deployment strategies for web applications. Each approach has its own advantages and may be suitable for different scenarios depending on the specific requirements and environment.

How can we use pseudo-classes and pseudo-elements in CSS? Provide examples.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CSS Pseudo-classes and Pseudo-elements Example</title>

<style>

/ Pseudo-class for styling links /

a:link {

color: blue; / Unvisited link color /

text-decoration: none; / Remove default underline /

}

/ Pseudo-class for styling visited links /

a:visited {

color: purple; / Visited link color /

}

/ Pseudo-class for styling links when hovered over /

a:hover {

color: red; / Hovered link color /

}

/ Pseudo-class for styling links when focused /

a:focus {

outline: 2px solid green; / Add a green outline when focused /

}

/ Pseudo-class for styling links when active /

a:active {

color: orange; / Active link color /

}

/ Pseudo-element to style the first letter of a paragraph /

p::first-letter {

font-size: 150%; / Increase font size of the first letter /

font-weight: bold; / Make the first letter bold /

color: green; / Color the first letter green /

}

/ Pseudo-element to insert content before an element /

p::before {

content: "🌟 "; / Insert a star emoji before each paragraph /

}

/ Pseudo-element to insert content after an element /

p::after {

content: " 🌟"; / Insert a star emoji after each paragraph /

}

</style>

</head>

<body>

<p>This is a paragraph with some text.</p>

<p>Another paragraph here.</p>

<p>And one more paragraph.</p>

<a href="https://www.example.com">Visit Example.com</a>

</body>

</html>

What is the meaning of HTTP Status Code. Explain codes 200, 500,400, 403 & 404.

HTTP status codes are standardized codes that a server sends in response to a client's request to indicate the outcome of the request. They provide information about whether a request was successful, encountered an error, or requires further action from the client.

Here's an explanation of the HTTP status codes you mentioned:

1. 200 OK:

- This status code indicates that the request was successful.

- The server has fulfilled the request and sent a response back to the client.

- This is the standard response for successful HTTP requests.

2. 400 Bad Request:

- This status code indicates that the server cannot process the request due to client error.

- It typically occurs when the client sends a malformed request, such as missing or incorrect parameters in the request.

- The client should correct the request and resend it.

3. 403 Forbidden:

- This status code indicates that the server understood the request but refuses to authorize it.

- It is often used when the server detects that the client does not have permission to access the requested resource.

- The client may need to authenticate or provide proper credentials to access the resource.

4. 404 Not Found:

- This status code indicates that the server cannot find the requested resource.

- It is commonly used when the requested URL does not exist on the server.

- This could be due to a mistyped URL, deleted resource, or moved resource without proper redirection.

- The client should verify the URL or inform the user that the requested page does not exist.

5. 500 Internal Server Error:

- This status code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

- It is a generic error message, indicating that something went wrong on the server side but the server could not specify the exact problem.

- This could be due to a programming error, misconfiguration, or issues with the server's infrastructure.

Understanding HTTP status codes is essential for troubleshooting and debugging web applications, as they provide valuable information about the outcome of HTTP requests and help in diagnosing issues between clients and servers.

Explain jQuery with a suitable example.

jQuery is like a helper that makes it easier to work with JavaScript. It's a library of pre-written JavaScript code that you can use to do common tasks on webpages.

Imagine you're building a house, and instead of cutting every piece of wood by hand, you have a set of power tools that do the work for you. jQuery is like those power tools for JavaScript.

With jQuery, you can do things like:

1. Select Elements: Easily find and work with HTML elements on your webpage.

2. Manipulate Content: Add, remove, or change the content of your webpage dynamically.

3. Handle Events: React to user actions, like clicks and keypresses.

4. Make AJAX Requests: Fetch data from other websites without refreshing the page.

5. Animate Elements: Create animations and transitions to make your webpage more interactive.

In short, jQuery simplifies web development by providing an easy-to-use interface for common JavaScript tasks, allowing you to write less code and accomplish more.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>jQuery Example</title>

<!-- Include jQuery library -->

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

</head>

<body>

<h1>Click the button to load some content!</h1>

<button id="loadButton">Load Content</button>

<div id="content"></div>

<script>

// jQuery code

$(document).ready(function() {

// When the page is fully loaded, this function is called

// We're telling jQuery to run this code after the page is ready

$('loadButton').click(function() {

// When the button is clicked, this function is called

// We're using jQuery to make an AJAX request to a website

// This is like asking another website for some information

// When we get the information back, we'll put it into the 'content' div

$.ajax({

url: 'https://jsonplaceholder.typicode.com/posts/1',

method: 'GET',

success: function(data) {

// This function is called if the request is successful

// We're updating the 'content' div with the data we got back

$('content').html('<h2>Title: ' + data.title + '</h2><p>Body: ' + data.body + '</p>');

},

error: function(xhr, status, error) {

// This function is called if there's an error with the request

// We'll log the error to the console

console.error('Error:', error);

}

});

});

});

</script>

</body>

</html>

The `<jsp:setProperty>` tag in JavaServer Pages (JSP) is used to set the value of a JavaBean property from a request parameter or an expression. JavaBeans are reusable software components that follow specific conventions for property accessors (getters and setters).

The purpose of the `<jsp:setProperty>` tag is to simplify the process of setting the properties of a JavaBean from the parameters passed in an HTTP request. This tag is particularly useful when you want to populate JavaBean properties with values submitted from an HTML form.

Here's a basic example of how `<jsp:setProperty>` can be used:

Let's say you have a JavaBean called `UserBean` with properties `username` and `password`, and you want to set these properties from the request parameters `usernameParam` and `passwordParam`, respectively.

```jsp

<jsp:useBean id="user" class="com.example.UserBean" scope="request"/>

<jsp:setProperty name="user" property="username" param="usernameParam"/>

<jsp:setProperty name="user" property="password" param="passwordParam"/>

```

In this example:

- The `<jsp:useBean>` tag creates an instance of the `UserBean` class and stores it in the request scope with the name `user`.

- The `<jsp:setProperty>` tags set the `username` property of the `UserBean` instance (`user`) to the value of the request parameter `usernameParam`, and similarly, the `password` property to the value of the request parameter `passwordParam`.

Overall, the `<jsp:setProperty>` tag simplifies the process of setting JavaBean properties from request parameters in JSP, helping to keep the code concise and maintainable.

What is JDBC? Write the steps to fetch data from a database.

//Java program to implement a simple JDBC application

package com.vinayak.jdbc;

import java.sql.;

public class JDBCDemo {

public static void main(String args[])

throws SQLException, ClassNotFoundException

{

String driverClassName

= "sun.jdbc.odbc.JdbcOdbcDriver";

String url = "jdbc:odbc:XE";

String username = "scott";

String password = "tiger";

String query

= "insert into students values(109, 'bhatt')";

// Load driver class

Class.forName(driverClassName);

// Obtain a connection

Connection con = DriverManager.getConnection(

url, username, password);

// Obtain a statement

Statement st = con.createStatement();

// Execute the query

int count = st.executeUpdate(query);

System.out.println(

"number of rows affected by this query= "

+ count);

// Closing the connection as per the

// requirement with connection is completed

con.close();

}

} // class

JDBC is a Java database API used for making connection between java applications with various databases. Basically, JDBC used for establishing stable database connection with the application API. To execute and process relational database queries (SQL or Oracle queries), multiple application can connect to different types of databases which supports both standard (SE) and enterprise (EE) edition of java.

What is a bean ? use setter and setter method in a bean program.

A bean in Java is a reusable software component that follows specific conventions for property accessors (getters and setters) and constructor initialization. In simple terms, a bean is a Java class that has private member variables (fields) with corresponding public methods (getters and setters) to access and modify those variables.

Here's an example of a simple Java bean class:

```java

public class Person {

private String name;

private int age;

// Constructor

public Person() {

}

// Getter for name

public String getName() {

return name;

}

// Setter for name

public void setName(String name) {

this.name = name;

}

// Getter for age

public int getAge() {

return age;

}

// Setter for age

public void setAge(int age) {

this.age = age;

}

}

```

In this example:

- We have a `Person` class, which serves as our Java bean.

- It has two private member variables: `name` and `age`.

- For each member variable, we provide a getter method (`getName()` and `getAge()`) to retrieve the value and a setter method (`setName()` and `setAge()`) to set the value.

Now, let's see how we can use this `Person` bean in a program:

```java

public class Main {

public static void main(String[] args) {

// Create a new Person object

Person person = new Person();

// Use setter methods to set the values of the properties

person.setName("John Doe");

person.setAge(30);

// Use getter methods to retrieve and display the property values

System.out.println("Name: " + person.getName());

System.out.println("Age: " + person.getAge());

}

}

```

In this program:

- We create a new `Person` object using the default constructor.

- We use the setter methods (`setName()` and `setAge()`) to set the values of the `name` and `age` properties, respectively.

- We use the getter methods (`getName()` and `getAge()`) to retrieve the values of the properties and print them to the console.

This demonstrates the basic usage of a Java bean class with getter and setter methods. Beans are commonly used in Java development for encapsulating and managing data, providing a clean and organized way to work with object properties.

With the help of Java script code block which checks the contents entered in a text box of a form. If the test entered is in the lower case, convert to upper case using the builtin function.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Box Content Conversion</title>

<script>

function convertToUpperCase() {

// Get the value entered in the text box

var textBoxValue = document.getElementById('textBox').value;

// Check if the text is in lowercase

if (textBoxValue === textBoxValue.toLowerCase()) {

// Convert the text to uppercase

textBoxValue = textBoxValue.toUpperCase();

// Set the converted value back to the text box

document.getElementById('textBox').value = textBoxValue;

}

}

</script>

</head>

<body>

<form>

<label for="textBox">Enter Text:</label>

<input type="text" id="textBox" oninput="convertToUpperCase()">

<button type="button" onclick="convertToUpperCase()">Convert to Uppercase</button>

</form>

</body>

</html>

Explain the use of JSP/ Servlet in web application. Write down the directory structure of the application using a servlet.

JavaServer Pages (JSP) and Servlets are fundamental components of Java-based web applications. They work together to handle dynamic content generation, data processing, and interaction with clients (browsers).

Here's how JSP and Servlets are used in a web application:

1. JSP (JavaServer Pages):

- JSP pages are used for creating dynamic web content by embedding Java code within HTML.

- They allow developers to separate the presentation logic (HTML markup) from the business logic (Java code).

- JSP pages are compiled into servlets by the application server during runtime, making them efficient for generating dynamic content.

2. Servlets:

- Servlets are Java classes that extend the functionality of a web server.

- They handle requests from clients (browsers), process data, and generate dynamic responses.

- Servlets run on the server-side and can interact with databases, other web services, and external resources.

- They provide a powerful way to implement the business logic of a web application and manage session state.

Now, let's discuss the directory structure of a web application using Servlets:

```

webapp/

├── WEB-INF/

│ ├── classes/

│ │ └── com/

│ │ └── example/

│ │ └── servlets/

│ │ └── MyServlet.class

│ ├── lib/

│ ├── web.xml

├── index.jsp

```

Explanation of the directory structure:

- `webapp/`: This is the root directory of the web application.

- `WEB-INF/`: This directory contains configuration files and classes that are not directly accessible by clients.

- `classes/`: This directory contains compiled Java classes (servlets) and other utility classes.

- `com/example/servlets/`: This is the package structure for servlet classes. Servlet classes are stored here.

- `MyServlet.class`: Example servlet class file.

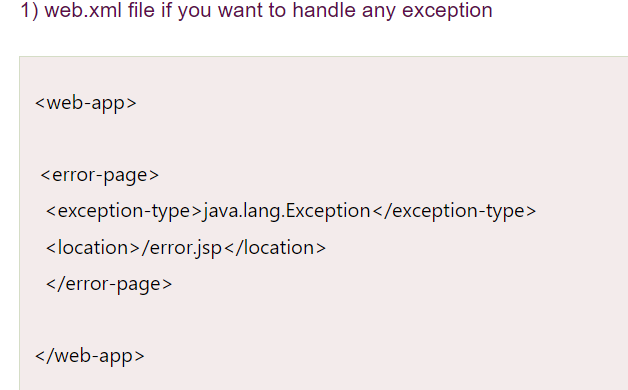
- `lib/`: This directory contains JAR files of third-party libraries used by the application.

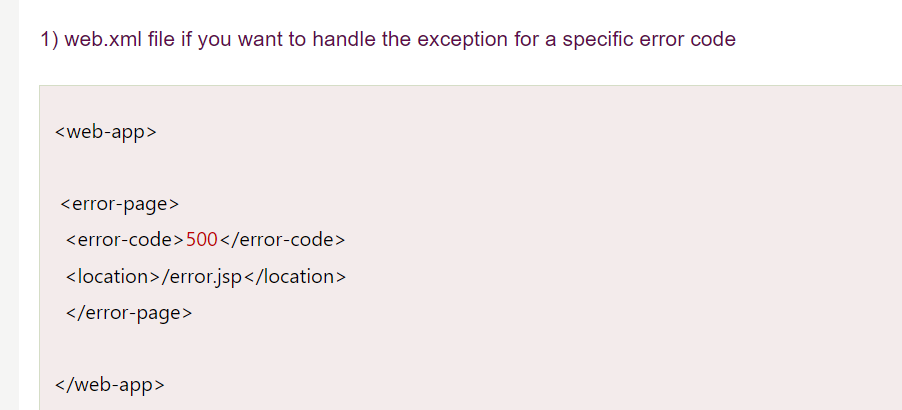
- `web.xml`: This is the deployment descriptor file for the web application. It contains configuration information such as servlet mappings, welcome pages, and initialization parameters.

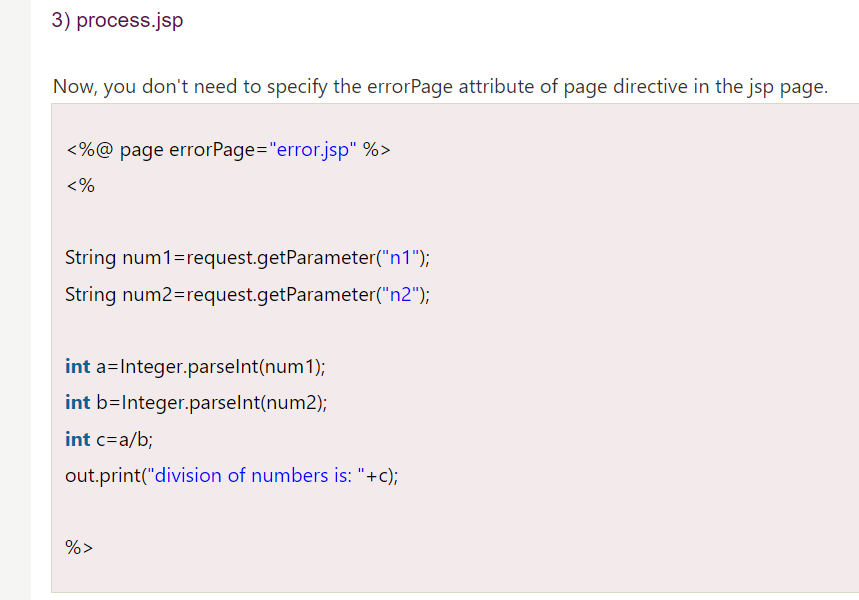
- `index.jsp`: This is a JSP file that serves as the entry point of the application. It may contain HTML markup, client-side scripting, and embedded Java code to generate dynamic content.

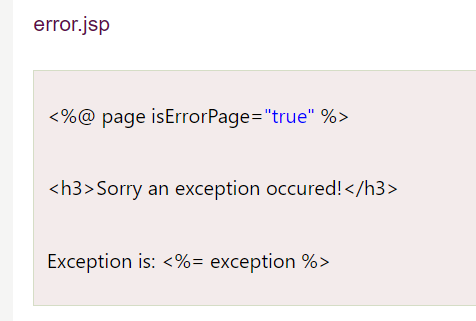
In summary, the directory structure of a web application using Servlets typically includes directories for servlet classes, configuration files, JSP files, and other resources. This structure helps organize the code and resources of the application and ensures that servlet classes are properly deployed and accessible to clients.

How can we handle the exceptions in Java Server Pages (JSP)? Explain with a suitable example.









|  |  |  |
| --- | --- | --- |
| **Feature** | **HTML 4** | **HTML 5** |
| **Release Date** | 1997 | October 2014 |
| **Doctype Declaration** | Lengthy and complex | Simplified: <!DOCTYPE html> |
| **Vector Graphics** | Requires plug-ins like Adobe Flash | Native support with <svg> |
| **Audio/Video Support** | Requires external plugins (like Flash) | Native support with <audio> and <video> |
| **Semantics** | Limited semantic elements | Rich semantic elements like <article>, <nav>, <section>, <header>, <footer> |
| **APIs and DOM** | Limited API support | Extensive APIs (offline storage, drag-drop, etc.) |
| **Graphics** | Limited; mainly through external plugins | Canvas API for 2D/3D graphics |
| **Storage** | Cookies for client-side storage | Local storage, session storage |
| **Accessibility** | Basic | Improved with new semantic elements |
| **Browser Compatibility** | Consistent across modern browsers | Varies; older browsers require polyfills |
| **Mobile Support** | Limited | Enhanced, with features for mobile optimization |
| **Form Controls** | Basic controls | New form controls like calendar, date, time, color picker |
| **Scripting** | Heavily reliant on external scripts | More integrated scripting capabilities |
| **Structure** | Less emphasis on document structure | Strong focus on document structure |

Why do we use action tags in JSP? Explain with a suitable example.

In JavaServer Pages (JSP), the <jsp:action> tags are used to perform specific actions or control flow within the JSP page. One common use of action tags is to perform conditional logic or iteration.

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Greeting Page</title>

</head>

<body>

<jsp:useBean id="date" class="java.util.Date" />

<jsp:setProperty name="date" property="" />

<%

// Get the current hour from the Date object

int hour = date.getHours();

// Perform conditional logic using <jsp:choose> and <jsp:when> tags

%>

<jsp:choose>

<jsp:when test="<%= hour < 12 %>">

<h1>Good morning!</h1>

</jsp:when>

<jsp:when test="<%= hour < 18 %>">

<h1>Good afternoon!</h1>

</jsp:when>

<jsp:otherwise>

<h1>Good evening!</h1>

</jsp:otherwise>

</jsp:choose>

</body>

</html>

What is the use of JDBC drivers? Explain different types of JDBC drivers.

JDBC (Java Database Connectivity) drivers are essential components that enable Java applications to interact with different types of databases using the JDBC API. These drivers act as intermediaries between the Java application and the database management system (DBMS), allowing the application to send SQL queries to the database and process the results.

There are four types of JDBC drivers:

1. Type 1: JDBC-ODBC Bridge Driver:

- This driver uses the JDBC-ODBC (Open Database Connectivity) bridge to connect to the database.

- It requires the presence of an ODBC driver on the client machine.

- The JDBC-ODBC bridge converts JDBC calls into ODBC calls, which are then executed by the ODBC driver.

- This driver is suitable for testing and prototyping but is not recommended for production use due to performance overhead and platform dependency.

2. Type 2: Native-API Driver:

- Also known as the Native API partly Java driver, this driver uses native methods to connect to the database.

- It communicates directly with the database through native libraries provided by the database vendor.

- While it offers better performance compared to the JDBC-ODBC bridge driver, it is still platform-dependent because it relies on native libraries.

3. Type 3: Network Protocol Driver:

- Also known as the Middleware driver or the All-Java driver, this driver communicates with the database through a middleware server.

- The client-side JDBC driver communicates with the middleware server using a network protocol (such as TCP/IP).

- The middleware server then translates the JDBC calls into the vendor-specific database protocol and forwards them to the database server.

- This driver provides platform independence and can be used in environments where a direct connection to the database server is not possible.

4. Type 4: Thin Driver:

- Also known as the Native-protocol driver or the Pure Java driver, this driver communicates directly with the database server using the vendor-specific protocol.

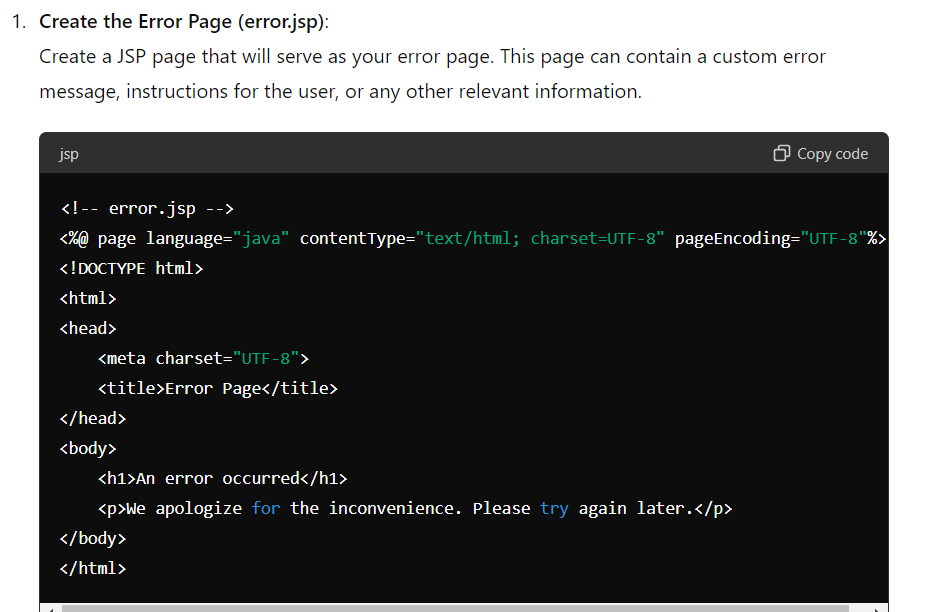
- It does not require any native libraries or middleware servers.

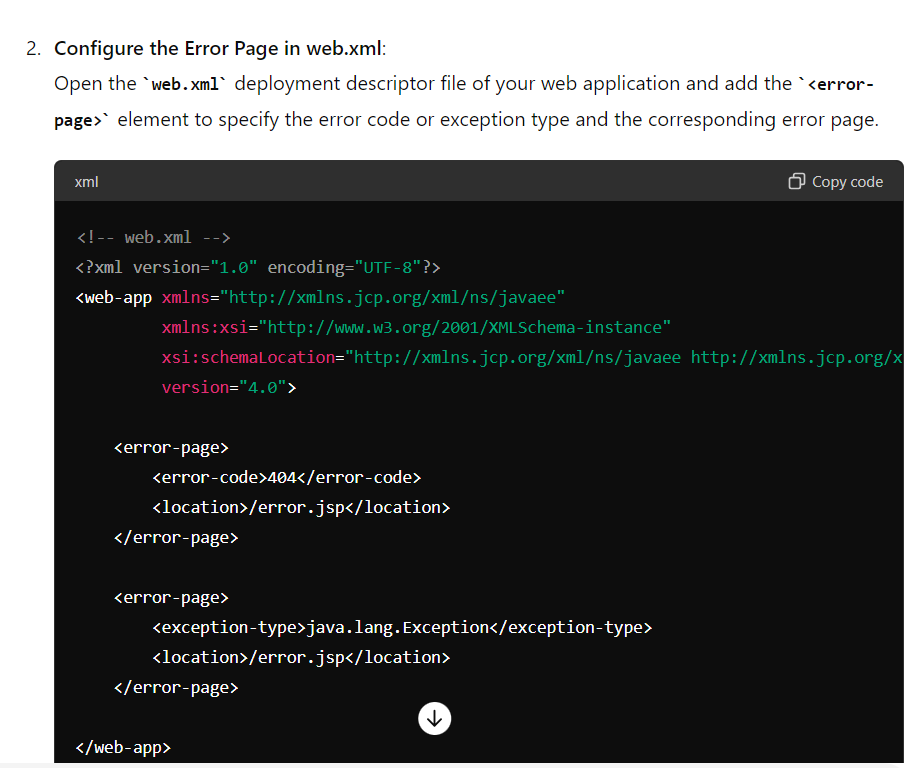
- The entire driver is written in Java, making it platform-independent and suitable for deploying Java applications in various environments.

- Type 4 drivers typically offer the best performance and are recommended for most production deployments.

Each type of JDBC driver has its advantages and limitations, and the choice of driver depends on factors such as performance requirements, platform compatibility, and network infrastructure. In general, Type 4 drivers are preferred for their simplicity, performance, and platform independence.

How can we create a dedicated error page in Java Server Pages (JSP)?





What are the different types of Javascript errors?

JavaScript errors can be categorized into several types based on their nature and cause. Here are some common types of JavaScript errors:

1. Syntax Errors:

- Syntax errors occur when the JavaScript code violates the language syntax rules.

- Examples include missing semicolons, parentheses, curly braces, or incorrect keywords.

2. Reference Errors:

- Reference errors occur when the code tries to access a variable or function that does not exist.

- This can happen when using undeclared variables, mistyping variable names, or trying to access properties of `null` or `undefined`.

3. Type Errors:

- Type errors occur when a value is not of the expected type or when an operation is performed on an inappropriate type.

- Examples include trying to call a non-function object, accessing properties or methods of a non-object, or performing arithmetic operations on non-numeric values.

4. Range Errors:

- Range errors occur when an operation tries to access an invalid index or numeric value that is outside the permissible range.

- This often happens with array indexing, accessing string characters by index, or using invalid parameters for built-in functions.

5. Runtime Errors:

- Runtime errors occur during the execution of the code and are often caused by unexpected conditions or external factors.

- Examples include division by zero, stack overflow (exceeding the call stack limit), or invalid JSON parsing.

6. Network Errors:

- Network errors occur when there are problems with fetching or sending data over the network, such as failed HTTP requests or timeouts.

- These errors are common in asynchronous operations like AJAX requests or WebSocket connections.

7. Custom Errors:

- Custom errors are errors that are explicitly thrown by the code using the `throw` statement.

- Developers can create custom error objects to handle specific exceptional conditions in their applications.

Understanding the different types of JavaScript errors can help developers diagnose and troubleshoot issues more effectively. Using browser developer tools, such as the console, can provide detailed information about the type, message, and stack trace of JavaScript errors encountered during runtime.

Compare iteration and recursion using Javascript.

Iteration and recursion are two fundamental techniques used in programming, including JavaScript, for solving problems that involve repetitive tasks. Let's compare them in terms of their implementation, advantages, and considerations:

Iteration:

- Implementation: Iteration involves using loops (e.g., `for`, `while`, `do-while`) to repeatedly execute a block of code until a certain condition is met.

- Advantages:

- Straightforward implementation: Iterative solutions are often easier to understand and debug, especially for simpler problems.

- Efficiency: Iteration is generally more efficient in terms of performance and memory usage, particularly for problems with large input sizes, as it avoids the overhead of function calls and stack manipulation.

- Considerations:

- State management: Iterative solutions typically rely on mutable state variables to track progress and maintain loop control, which can lead to potential side effects and make the code harder to reason about.

- Code duplication: Iterative code may require more lines of code and can be prone to redundancy, especially when dealing with nested loops or complex loop conditions.

Recursion:

- Implementation: Recursion involves defining a function that calls itself with modified arguments, gradually reducing the problem size until reaching a base case where the solution is trivial.

- Advantages:

- Elegant solution for certain problems: Recursion is particularly well-suited for problems that exhibit a recursive structure, such as tree traversal, backtracking, or divide-and-conquer algorithms.

- Simplified logic: Recursive solutions often result in cleaner and more concise code by encapsulating repetitive behavior within a single function.

- Considerations:

- Performance overhead: Recursion may incur additional overhead due to function calls and stack manipulation, which can be significant for deeply nested or unoptimized recursive algorithms.

- Risk of stack overflow: Recursive functions consume stack space with each recursive call, potentially leading to a stack overflow error if the recursion depth is too large.

- Debugging complexity: Recursive code can be more challenging to debug and analyze, especially when dealing with complex recursion patterns or base case conditions.

Example:

Let's compare the iterative and recursive implementations of calculating the factorial of a number:

Iterative:

```javascript

function factorialIterative(n) {

let result = 1;

for (let i = 2; i <= n; i++) {

result = i;

}

return result;

}

```

Recursive:

```javascript

function factorialRecursive(n) {

if (n <= 1) {

return 1;

}

return n factorialRecursive(n - 1);

}

```

In summary, both iteration and recursion have their own strengths and weaknesses, and the choice between them depends on the specific problem, its constraints, and the desired trade-offs between simplicity, efficiency, and elegance.

Write a Servlet which includes the content of another Servlet (ie data of other servlet is coming to your servlet).

SERVLET 1

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Servlet1 extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Set content type

response.setContentType("text/html");

// Retrieve request dispatcher for Servlet2

RequestDispatcher dispatcher = request.getRequestDispatcher("/Servlet2");

// Include content of Servlet2 within Servlet1

dispatcher.include(request, response);

}

}

SERVLET 2

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Servlet2 extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Set content type

response.setContentType("text/html");

// Write content to response

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head><title>Servlet2</title></head>");

out.println("<body>");

out.println("<h1>This is Servlet2</h1>");

out.println("<p>Hello from Servlet2</p>");

out.println("</body>");

out.println("</html>");

}

}

XML

<servlet>

<servlet-name>Servlet1</servlet-name>

<servlet-class>com.example.Servlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet1</servlet-name>

<url-pattern>/Servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>Servlet2</servlet-name>

<servlet-class>com.example.Servlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet2</servlet-name>

<url-pattern>/Servlet2</url-pattern>

</servlet-mapping>

Write a Servlet which receives initialization values from web.xml

<!-- web.xml -->

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<servlet>

<servlet-name>InitParamServlet</servlet-name>

<servlet-class>com.example.InitParamServlet</servlet-class>

<init-param>

<param-name>username</param-name>

<param-value>admin</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>secret</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>InitParamServlet</servlet-name>

<url-pattern>/initParamServlet</url-pattern>

</servlet-mapping>

</web-app>

SERVLET

package com.example;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.PrintWriter;

public class InitParamServlet extends HttpServlet {

private String username;

private String password;

@Override

public void init() throws ServletException {

// Retrieve initialization parameters from the servlet configuration

username = getServletConfig().getInitParameter("username");

password = getServletConfig().getInitParameter("password");

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Set content type

response.setContentType("text/html");

// Write the response

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head><title>Initialization Parameters</title></head>");

out.println("<body>");

out.println("<h1>Servlet Initialization Parameters</h1>");

out.println("<p>Username: " + username + "</p>");

out.println("<p>Password: " + password + "</p>");

out.println("</body>");

out.println("</html>");

}

}

Construct an HTML code using various text formatting tags

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Formatting Example</title>

</head>

<body>

<h1>Text Formatting in HTML</h1>

<h2>Bold Text</h2>

<p>This is <b>bold</b> text.</p>

<p>This is <strong>strong</strong> text.</p>

<h2>Italic Text</h2>

<p>This is <i>italic</i> text.</p>

<p>This is <em>emphasized</em> text.</p>

<h2>Underlined Text</h2>

<p>This is <u>underlined</u> text.</p>

<h2>Strikethrough Text</h2>

<p>This is <s>strikethrough</s> text.</p>

<p>This is <del>deleted</del> text.</p>

<h2>Inserted Text</h2>

<p>This is <ins>inserted</ins> text.</p>

<h2>Subscript and Superscript</h2>

<p>This is <sub>subscript</sub> text.</p>

<p>This is <sup>superscript</sup> text.</p>

<h2>Small Text</h2>

<p>This is <small>small</small> text.</p>

<h2>Code and Preformatted Text</h2>

<p>This is a <code>code</code> snippet.</p>

<pre>

function sayHello() {

console.log("Hello, world!");

}

</pre>

<h2>Blockquote</h2>

<blockquote>

"The only limit to our realization of tomorrow is our doubts of today."

- Franklin D. Roosevelt

</blockquote>

<h2>Text with Mark</h2>

<p>This is <mark>highlighted</mark> text.</p>

<h2>Keyboard Input</h2>

<p>Press <kbd>Ctrl</kbd> + <kbd>C</kbd> to copy.</p>

<h2>Variable</h2>

<p>The value of <var>x</var> is 5.</p>

<h2>Sample Output</h2>

<p>The computer said <samp>Hello, world!</samp></p>

</body>

</html>

List some features of jQuery.

Certainly! Here are the key features of jQuery:

1. DOM Manipulation: Simplifies finding, selecting, and manipulating HTML elements.

2. Event Handling: Makes it easier to attach event handlers to elements and supports event delegation.

3. Ajax Support: Simplifies making asynchronous HTTP requests and handling responses.

4. Animations and Effects: Provides built-in animations and effects, and supports custom animations.

5. Cross-Browser Compatibility: Ensures consistent behavior across different browsers.

6. CSS Manipulation: Allows easy manipulation of CSS properties and classes.

7. HTML/DOM Element Creation and Insertion: Provides methods to create and insert elements into the DOM.

8. Utility Functions: Offers utility functions for common tasks like iteration, extending objects, and more.

9. Plugins: Supports an extensive plugin architecture for extending jQuery's functionality.

10. Chaining: Allows method chaining to improve code readability and conciseness.

11. Selectors: Provides a powerful set of selectors to find elements within the DOM using CSS-like syntax.

12. Deferred and Promises: Implements deferred objects and promises for handling asynchronous operations more effectively.

Discover the benefits of using JavaScript

JavaScript is a versatile and powerful programming language that is essential for modern web development. Here are some of the key benefits of using JavaScript:

1. Client-Side Execution:

- Fast Execution: JavaScript code runs directly in the browser, reducing the need for server round-trips and improving application performance.

- Immediate Feedback: Users get immediate feedback from their interactions without waiting for a server response.

2. Interactivity and Rich User Interfaces:

- Dynamic Content: JavaScript enables dynamic content updates without refreshing the page, enhancing user experience.

- Interactive Elements: It allows the creation of interactive elements like sliders, drag-and-drop features, and interactive forms.

3. Cross-Browser Compatibility:

- Standardized: Modern JavaScript is standardized by ECMAScript (ES), ensuring compatibility across different browsers.

- Polyfills and Libraries: Tools like polyfills and libraries (e.g., jQuery) help maintain compatibility across older and newer browsers.

4. Extensive Frameworks and Libraries:

- Rich Ecosystem: There is a vast ecosystem of frameworks (e.g., React, Angular, Vue.js) and libraries (e.g., jQuery, Lodash) that simplify development.

- Reusable Components: Frameworks allow the creation of reusable components, streamlining development and maintenance.

5. Versatility:

- Full-Stack Development: With environments like Node.js, JavaScript can be used for both client-side and server-side development, enabling full-stack development.

- Mobile and Desktop Apps: JavaScript frameworks (e.g., React Native, Electron) enable the development of mobile and desktop applications.

6. Event-Driven Programming:

- Asynchronous Operations: JavaScript's event-driven nature and support for asynchronous operations (via callbacks, promises, async/await) make it well-suited for handling concurrent tasks.

- User Interaction: It efficiently handles user interactions and events (e.g., clicks, form submissions, mouse movements).

7. Ease of Learning and Use:

- Widely Adopted: JavaScript is widely taught and used, making it easier to find learning resources and community support.

- Syntax: Its syntax is relatively simple and easy to understand for beginners.

8. Integration Capabilities:

- APIs and Services: JavaScript can interact with a wide range of APIs and services, facilitating the integration of third-party functionalities (e.g., Google Maps, payment gateways).

- HTML and CSS: It works seamlessly with HTML and CSS to create robust web applications.

9. Performance Improvements:

- JIT Compilation: Modern JavaScript engines (e.g., V8 in Chrome) use Just-In-Time (JIT) compilation to improve execution speed.

- Optimizations: Continuous optimizations in JavaScript engines lead to better performance over time.

10. Community and Ecosystem:

- Large Community: A large and active community contributes to a wealth of resources, libraries, frameworks, and tools.

- Regular Updates: Regular updates and new features are added through ECMAScript specifications and community contributions.

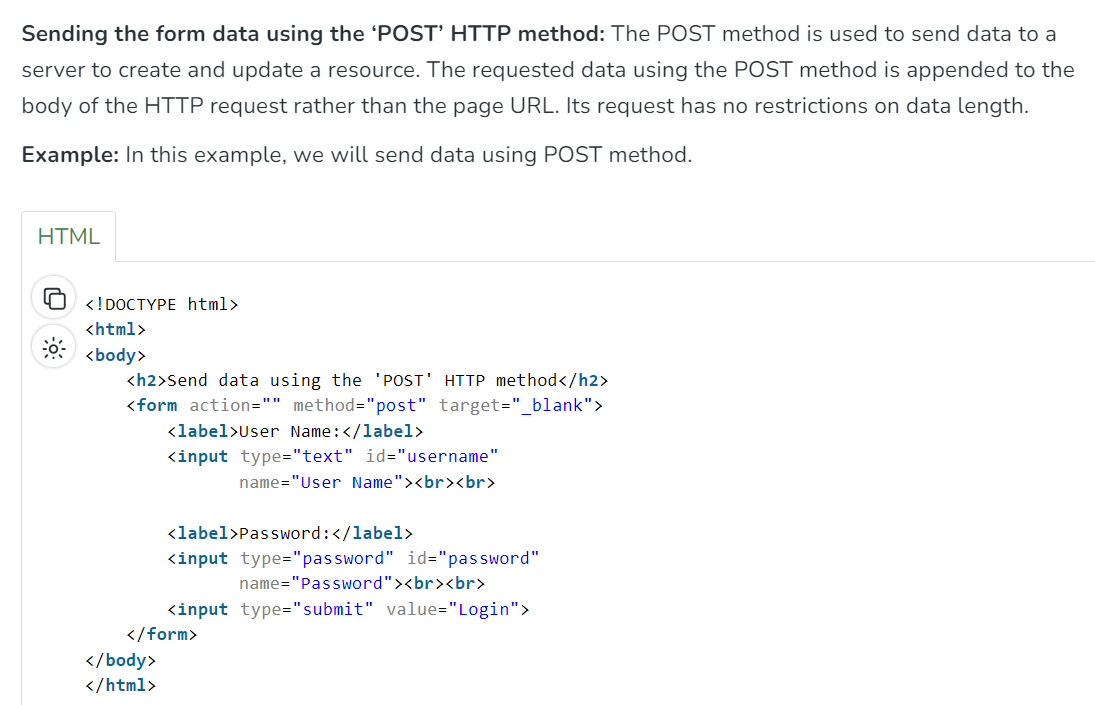
By leveraging these benefits, developers can create highly responsive, interactive, and efficient web applications that provide a seamless user experience.

Outline the various http methods used to send an html form’s data to server in details.Explain with the help of a program.

**Sending the form data using the ‘GET’ HTTP method:** The GET method is used to request data from specified resources. It sends an empty body to the server and asks to get resources. If the form data is sent using the GET method the data sent by the server is appended to the URL of the page. Its requests have some length restrictions. It is not used for modification.









What is the utility of Spring boot in web programming? Explain its working

Utility of Spring Boot in Web Programming

Spring Boot is a powerful, open-source Java-based framework used to create stand-alone, production-grade Spring applications with minimal configuration. It is designed to simplify the development of new Spring applications by providing defaults for code and annotation configuration to reduce boilerplate code.

Key Utilities and Benefits

1. Auto-Configuration:

- Automatically configures your Spring application based on the dependencies you have added, which helps in reducing the need for extensive XML configuration or Java configuration classes.

2. Standalone Applications:

- Spring Boot applications can run independently, which means they can be started by simply running the `main` method. It embeds servers like Tomcat, Jetty, or Undertow directly, which removes the need to deploy WAR files.

3. Production-Ready Features:

- Includes several features to help you monitor and manage your application in production, such as health checks, metrics, application info, and externalized configuration.

4. Microservices Support:

- Spring Boot is designed to support the development of microservices architectures, providing a robust ecosystem to build, deploy, and manage scalable applications.

5. Developer Productivity:

- Includes a range of tools and frameworks to speed up development and testing, such as Spring Initializr for project generation, and Spring DevTools for live reload and hot swapping.

6. Opinionated Defaults:

- Provides sensible defaults to get started quickly but allows full customization when required. This means it makes it easy to start a project but doesn't limit you when you need to customize it to your needs.

How Spring Boot Works

Spring Boot works by leveraging the core Spring Framework and its features, and layering on top of them to provide an easier, more productive way to develop Spring-based applications. Here’s a step-by-step overview of its working:

1. Spring Boot Starter Projects:

- Spring Boot provides a range of starter POMs (Project Object Models) to simplify dependency management. For example, `spring-boot-starter-web` includes all necessary dependencies to create a web application.

2. Auto-Configuration:

- When a Spring Boot application starts, it scans the classpath and the configured beans to auto-configure the application context based on the presence of certain classes and properties.

3. Embedded Servers:

- Spring Boot applications are packaged as JAR files by default and include an embedded server (like Tomcat) so that you can run the application directly without deploying it to an external server.

4. Spring Boot Application Class:

- The entry point of a Spring Boot application is the main class annotated with `@SpringBootApplication`. This annotation enables auto-configuration, component scanning, and configuration properties support.

5. Externalized Configuration:

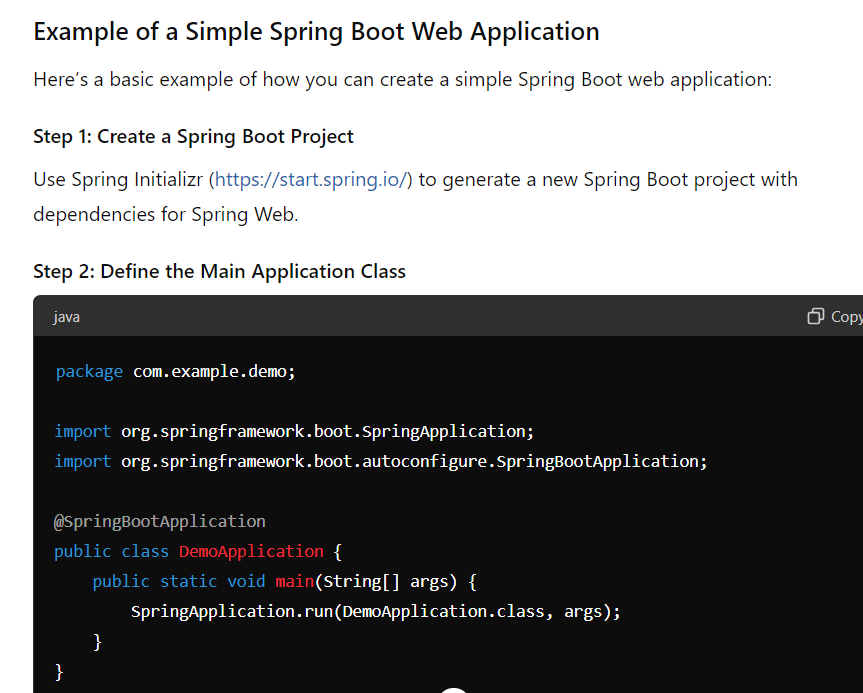
- Spring Boot allows you to externalize your configuration so that you can work with the same application code in different environments. Configuration properties can be defined in `application.properties` or `application.yml` files.

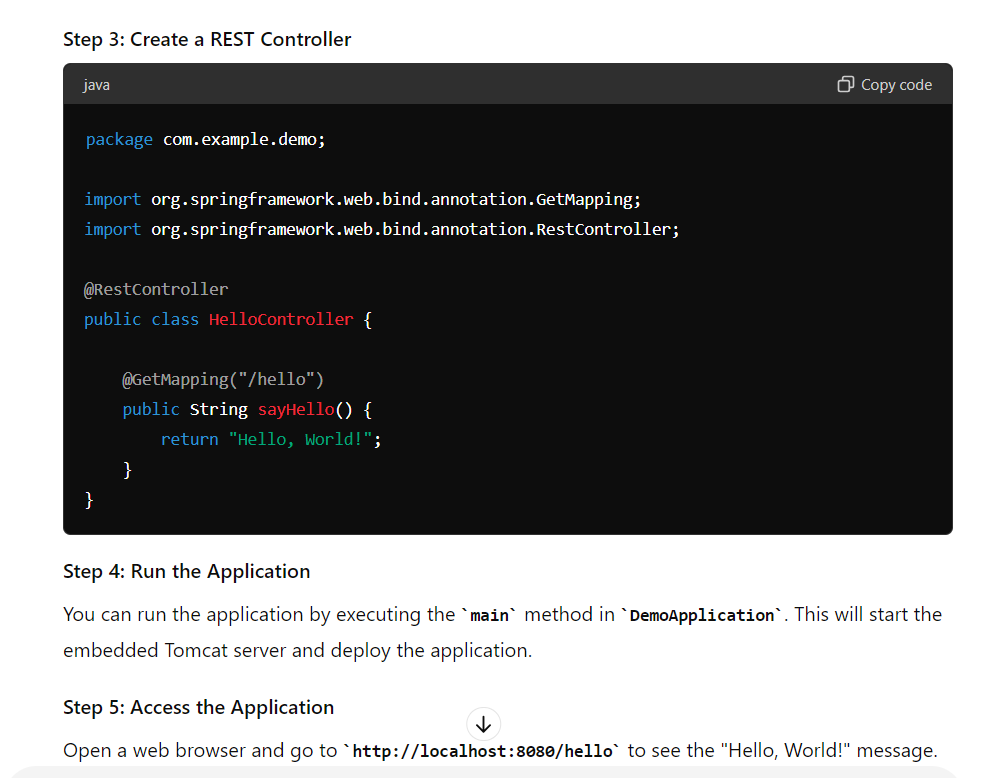
6. Actuator:

- Spring Boot Actuator provides production-ready features like health checks, metrics, information about the application, and more. This is crucial for monitoring and managing the application in a production environment.

7. Spring Boot DevTools:

- Spring Boot DevTools provides additional development-time features, such as automatic restart, live reload, and configurations aimed at improving developer productivity.

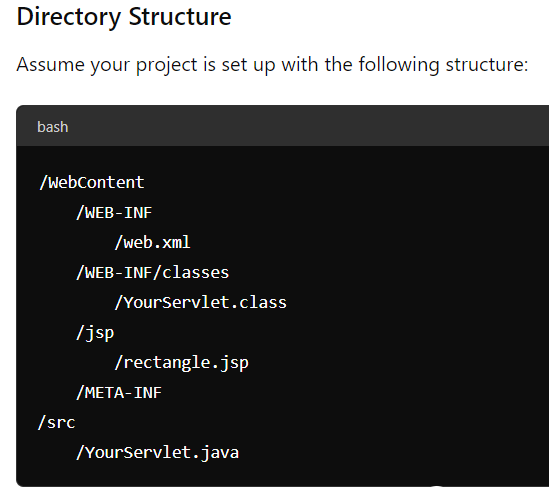


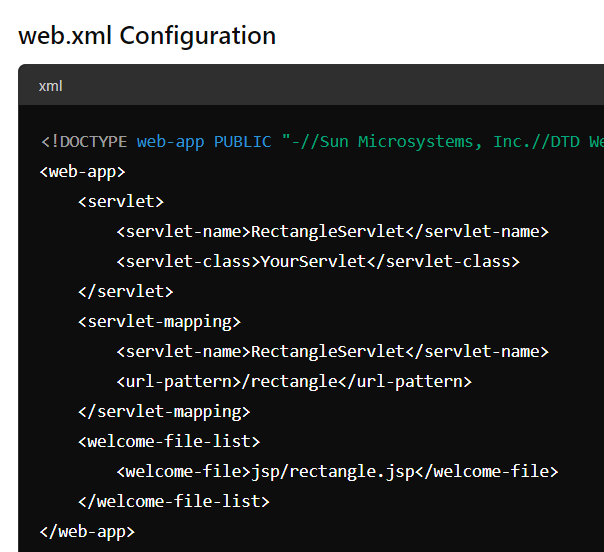


Conclusion

Spring Boot significantly simplifies the development of Spring applications by providing a range of tools and features that enhance productivity and reduce configuration overhead. Its embedded servers, auto-configuration, starter dependencies, and production-ready features make it a valuable framework for building robust and scalable web applications

Write an Servlet / Java Server Pages (JSP) program to calculate Perimeter and area using post & get method





import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/rectangle")

public class YourServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

private void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

Write a Servlet code snippet using PreparedStatement to insert data into a table in a MySQL/ Oracle database.

What is the use of Spring and Spring Boot in web application. Explain with a suitable example

Below is an example of a Servlet/JSP program to validate user login credentials using data stored in a database.

**Directory Structure**

**Assume your project is set up with the following structure:**

```

/WebContent

/WEB-INF

/web.xml

/jsp

/login.jsp

/META-INF

/src

/YourServlet.java

```

**web.xml Configuration**

**```xml**

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>YourServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>jsp/login.jsp</welcome-file>

</welcome-file-list>

</web-app>

```

**YourServlet.java**

**```java**

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/login")

public class YourServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String username = request.getParameter("username");

String password = request.getParameter("password");

// Database connection parameters

String url = "jdbc:mysql://localhost:3306/your\_database"; // MySQL database URL

// For Oracle, the URL format would be something like: "jdbc:oracle:thin:@localhost:1521:your\_sid"

String dbUsername = "your\_username";

String dbPassword = "your\_password";

// SQL query to validate user login

String sql = "SELECT FROM users WHERE username = ? AND password = ?";

try (

// Establishing database connection

Connection conn = DriverManager.getConnection(url, dbUsername, dbPassword);

// Creating prepared statement with SQL query

PreparedStatement pstmt = conn.prepareStatement(sql);

) {

// Setting parameters for prepared statement

pstmt.setString(1, username);

pstmt.setString(2, password);

// Executing the SQL query to validate user login

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

// User login successful

response.sendRedirect("success.jsp"); // Redirect to success page

} else {

// User login failed

response.sendRedirect("failure.jsp"); // Redirect to failure page

}

} catch (SQLException e) {

e.printStackTrace();

response.getWriter().println("SQL Exception: " + e.getMessage());

}

}

}

```

**login.jsp**

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login Page</title>

</head>

<body>

<h1>Login</h1>

<form action="login" method="post">

<label for="username">Username:</label>

<input type="text" id="username" name="username"><br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password"><br><br>

<input type="submit" value="Login">

</form>

</body>

</html>

```

**success.jsp**

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login Success</title>

</head>

<body>

<h1>Login Successful</h1>

<p>Welcome, user!</p>

</body>

</html>

```

**failure.jsp**

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login Failure</title>

</head>

<body>

<h1>Login Failed</h1>

<p>Invalid username or password. Please try again.</p>

</body>

</html>

```

This Servlet/JSP program validates user login credentials by checking against data stored in the database. The `login.jsp` page contains a login form where users can enter their username and password. Upon form submission, the `YourServlet` servlet is invoked to validate the credentials. If the credentials are valid, the user is redirected to the `success.jsp` page; otherwise, they are redirected to the `failure.jsp` page.

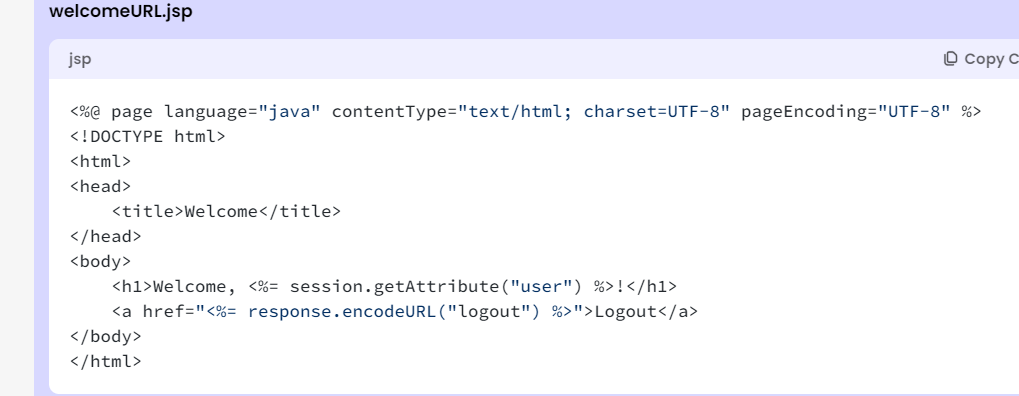
Write a Servlet/ Java Server Pages (JSP) program to maintain session using session object, Hidden Form Field and URL Rewriting











Use of `<span>` in HTML

The `<span>` tag in HTML is an inline container used to mark up a part of a text or a part of a document. It doesn't inherently carry any styling or formatting, but it's often used in combination with CSS and JavaScript to style and manipulate specific parts of content within a block-level element.

Characteristics:

- Inline Element: `<span>` is an inline element, which means it doesn't start on a new line and only takes up as much width as necessary.

- No Semantic Meaning: `<span>` does not convey any particular meaning about the content. It’s used purely for styling and scripting purposes.

- Versatile: It can be used anywhere within other text-containing elements, such as paragraphs, lists, or headers.

**Example Usage of `<span>`**

**Here's an example that demonstrates various uses of the `<span>` element:**

**```html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Example of Span Usage</title>

<style>

.highlight {

background-color: yellow;

}

.bold {

font-weight: bold;

}

.italic {

font-style: italic;

}

</style>

</head>

<body>

<h1>Understanding the <span class="italic">span</span> Element</h1>

<p>The <span class="bold">span</span> element is an inline container that can be used to apply styles or scripts to a part of the text within a block-level element. For example, you can <span class="highlight">highlight text</span> or make it <span class="bold">bold</span> and <span class="italic">italic</span>.</p>

<p>Another use case is for JavaScript manipulation. Click the button below to change the color of the highlighted text:</p>

<button onclick="changeColor()">Change Color</button>

<script>

function changeColor() {

document.querySelector('.highlight').style.color = 'red';

}

</script>

</body>

</html>

```

Explanation:

1. CSS Styling:

- The CSS classes `.highlight`, `.bold`, and `.italic` are used to style specific portions of text within the paragraph.

- The `<span>` elements within the paragraph are assigned these classes to apply the respective styles.

2. JavaScript Manipulation:

- A button is provided to demonstrate JavaScript manipulation. When clicked, it changes the color of the text within the `<span>` with the `highlight` class.

- The `changeColor` function uses `querySelector` to find the first element with the `highlight` class and changes its `color` style property to `red`.

3. Text Highlighting:

- The `<span class="highlight">highlight text</span>` part in the paragraph highlights the text with a yellow background.

4. Text Styling:

- `<span class="bold">bold</span>` makes the text bold.

- `<span class="italic">italic</span>` makes the text italic.

The `<span>` element is very useful for applying styles or handling specific portions of content within a larger block of text, making it an essential tool in HTML for fine-grained control over text presentation and interaction.

What is the utility of Spring in web programming? Explain

Utility of Spring in Web Programming

Spring is a comprehensive framework used extensively in Java-based web development. It offers a multitude of features and utilities that simplify and enhance the development of web applications. Here’s an overview of its key utilities and features:

1. Dependency Injection (DI) and Inversion of Control (IoC)

- Dependency Injection: Spring’s core feature allows developers to manage and inject dependencies between objects, promoting loose coupling and better modularization.

- IoC Container: Manages the lifecycle and configuration of application objects. It helps in creating objects, wiring them together, configuring them, and managing their lifecycle.

2. Spring MVC (Model-View-Controller)

- Separation of Concerns: Organizes web applications into three main layers: Model, View, and Controller. This clear separation improves code maintainability and readability.

- Request Handling: Provides powerful request handling features such as URL mapping, form handling, and validation.

- View Resolution: Integrates easily with various view technologies like JSP, Thymeleaf, and FreeMarker.

3. Spring Boot

- Simplified Configuration: Reduces the need for extensive XML configuration through a convention-over-configuration approach.

- Embedded Servers: Includes embedded servers (like Tomcat, Jetty) which simplify the deployment process by allowing the application to run independently.

- Auto-Configuration: Automatically configures Spring and third-party libraries based on the dependencies in the project.

4. Transaction Management

- Provides a consistent programming model for transaction management. It supports declarative transaction management using annotations and XML configurations.

5. Security

- Spring Security: A comprehensive security framework that handles authentication, authorization, and other security concerns. It integrates with various authentication mechanisms like LDAP, OAuth, and custom security systems.

6. Integration with Other Technologies

- Data Access: Seamlessly integrates with JDBC, JPA, Hibernate, and other ORM tools for data access.

- Messaging: Supports asynchronous messaging with integration for JMS, RabbitMQ, and Kafka.

- RESTful Web Services: Simplifies the creation of RESTful web services with annotations such as `@RestController` and `@RequestMapping`.

7. Aspect-Oriented Programming (AOP)

- Allows separating cross-cutting concerns such as logging, security, and transaction management from business logic, resulting in cleaner and more modular code.

8. Internationalization

- Provides comprehensive support for internationalization, enabling applications to be easily adapted for different languages and regions.

9. Batch Processing

- Spring Batch provides a robust framework for developing batch processing applications, including features like transaction management, job processing statistics, job restart, and job status management.

10. Testing

- Spring offers robust support for testing, including unit testing with JUnit and TestNG, and integration testing. The Spring TestContext Framework provides annotations and utility classes to simplify testing.

Summary

The Spring framework is widely adopted in the Java ecosystem due to its robust features, flexibility, and ease of use. It enhances web development by providing a comprehensive infrastructure that addresses a wide range of concerns, from dependency management and transaction handling to security and RESTful service creation. This makes Spring a powerful tool for building scalable, maintainable, and secure web applications.

Nested Tables

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Nested Tables Example</title>

</head>

<body>

<h1>Example of Nested Tables</h1>

<table border="1">

<tr>

<th>Main Table Header 1</th>

<th>Main Table Header 2</th>

</tr>

<tr>

<td>Main Table Row 1, Cell 1</td>

<td>

<table border="1">

<tr>

<th>Nested Table Header 1</th>

<th>Nested Table Header 2</th>

</tr>

<tr>

<td>Nested Row 1, Cell 1</td>

<td>Nested Row 1, Cell 2</td>

</tr>

<tr>

<td>Nested Row 2, Cell 1</td>

<td>Nested Row 2, Cell 2</td>

</tr>

</table>

</td>

</tr>

<tr>

<td>Main Table Row 2, Cell 1</td>

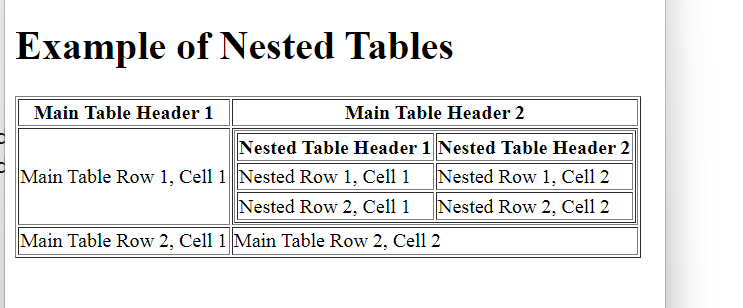
<td>Main Table Row 2, Cell 2</td>

</tr>

</table>

</body>

</html>



Create an HTML form in which accept marks of 5 subjects and submit marks to a servlet. Calculate the division of the students based upon following criterion Marks >= 60 First Division Marks = 45 Second Division Marks 33 Third Division Marks < 33 Fail

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Student Division Classification</title>

</head>

<body>

<h1>Student Division Classification</h1>

<form id="marksForm">

<label for="marks">Enter Marks:</label>

<input type="number" id="marks" name="marks" required>

<button type="button" onclick="classifyStudent()">Submit</button>

</form>

<p id="result"></p>

<script>

function classifyStudent() {

var marks = document.getElementById("marks").value;

var result = '';

if (marks >= 60) {

result = 'First Division';

} else if (marks >= 45 && marks < 60) {

result = 'Second Division';

} else if (marks > 33 && marks < 45) {

result = 'Third Division';

} else if (marks <= 33) {

result = 'Fail';

} else {

result = 'Invalid marks';

}

document.getElementById("result").innerText = "Division: " + result;

}

</script>

</body>

</html>

State with an example the various way to add JavaScript and CSS to HTML.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>JavaScript and CSS Inclusion Example</title>

<!-- External CSS -->

<link rel="stylesheet" href="styles.css">

<style>

/ Internal CSS /

h1 {

color: blue;

}

</style>

</head>

<body>

<h1>JavaScript and CSS Inclusion Example</h1>

<!-- Inline JavaScript -->

<button onclick="alert('Hello, world!')">Click me</button>

<!-- Internal JavaScript -->

<script>

function greet() {

alert('Hello, world!');

}

</script>

<!-- External JavaScript -->

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

<!-- Inline CSS -->

<p style="color: green;">This paragraph has inline CSS.</p>

</body>

</html>

Write a Servlet/ Java Server Pages (JSP) program showing Page Navigation using Send redirect & RequestDispatcher

Below is an example of a Servlet program demonstrating page navigation using both `sendRedirect` and `RequestDispatcher` methods:

**Servlet Example (PageNavigationServlet.java)**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/PageNavigationServlet")

public class PageNavigationServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String destination = request.getParameter("destination");

// Using sendRedirect

if ("sendRedirect".equals(destination)) {

response.sendRedirect("destination1.jsp");

}

// Using RequestDispatcher

if ("requestDispatcher".equals(destination)) {

request.getRequestDispatcher("destination2.jsp").forward(request, response);

}

}

}

```

**JSP Example (destination1.jsp)**

**```jsp**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Destination Page 1</title>

</head>

<body>

<h2>This is Destination Page 1</h2>

<p>Page navigated using sendRedirect.</p>

</body>

</html>

```

**JSP Example (destination2.jsp)**

**```jsp**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Destination Page 2</title>

</head>

<body>

<h2>This is Destination Page 2</h2>

<p>Page navigated using RequestDispatcher.</p>

</body>

</html>

```

Explanation:

- In the servlet (`PageNavigationServlet`), we handle a POST request and retrieve the destination parameter from the request.

- Depending on the value of the destination parameter, we use either `sendRedirect` or `RequestDispatcher` to navigate to the respective JSP page.

- `sendRedirect` method redirects the client to the specified URL (destination1.jsp), and the browser makes a new request to that URL.

- `RequestDispatcher` forward method sends the request to another resource (destination2.jsp) on the server. The client is unaware of the redirection.

- Both JSP pages (destination1.jsp and destination2.jsp) display a message indicating the page navigation method used.

Usage:

- Deploy the servlet and JSP files to a servlet container like Apache Tomcat.

- Access the servlet URL (e.g., `http://localhost:8080/yourapp/PageNavigationServlet`) from a web browser.

- Provide the destination parameter value as "sendRedirect" or "requestDispatcher" in the form, and submit.

- You'll be navigated to the respective JSP page using the specified navigation method.

Develop an application using AJAX and Java Server Pages (JSP) that contains html page that accept email and check and display a message email is present in the database or not. Create an user and store data in database

How do we use custom tag in Java Server Pages (JSP)? Formulate steps to create a custom tag.

Write a program of Servlet/ JSP in which pass three input using form element Name, Age and Marks. Use cookie / session objects to maintain session and send data to third page and print all.

Below is an example of a Servlet/JSP program that takes input from a form (Name, Age, and Marks), stores the data in session attributes, and displays the data on a third page:

**HTML Form Page (inputForm.html)**

**```html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Input Form</title>

</head>

<body>

<h2>Enter Details</h2>

<form action="StoreDataServlet" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br><br>

<label for="age">Age:</label>

<input type="number" id="age" name="age" required><br><br>

<label for="marks">Marks:</label>

<input type="number" id="marks" name="marks" required><br><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

```

**Servlet to Store Data (StoreDataServlet.java)**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/StoreDataServlet")

public class StoreDataServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String name = request.getParameter("name");

int age = Integer.parseInt(request.getParameter("age"));

int marks = Integer.parseInt(request.getParameter("marks"));

HttpSession session = request.getSession();

session.setAttribute("name", name);

session.setAttribute("age", age);

session.setAttribute("marks", marks);

response.sendRedirect("DisplayData.jsp");

}

}

```

**JSP to Display Data (DisplayData.jsp)**

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Display Data</title>

</head>

<body>

<h2>Details</h2>

<p>Name: <%= session.getAttribute("name") %></p>

<p>Age: <%= session.getAttribute("age") %></p>

<p>Marks: <%= session.getAttribute("marks") %></p>

</body>

</html>

```

Explanation:

- inputForm.html: This HTML page contains a form where users can input their name, age, and marks.

- StoreDataServlet.java: This servlet receives the form data, stores it in session attributes, and redirects to the DisplayData.jsp page.

- DisplayData.jsp: This JSP page retrieves the data from session attributes and displays it on the web page.

Usage:

1. Deploy the servlet and JSP files to a servlet container like Apache Tomcat.

2. Access the inputForm.html page from a web browser.

3. Enter the details in the form and submit.

4. The servlet will store the data in session attributes and redirect to the DisplayData.jsp page.

5. The DisplayData.jsp page will display the stored data from session attributes.

This example demonstrates how to use session objects to maintain session data across multiple requests and display it on subsequent pages.

Write a program of Servlet/ JSP in which pass three input using form element and find the greatest on in main.jsp page. If the greatest number is even then send control to Even.jsp otherwise odd.jsp

Here's an example of a Servlet/JSP program that takes three inputs using a form element, finds the greatest number, and redirects to either Even.jsp or Odd.jsp based on whether the greatest number is even or odd:

**HTML Form Page (inputForm.html)**

**```html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Input Form</title>

</head>

<body>

<h2>Enter Three Numbers</h2>

<form action="FindGreatestServlet" method="post">

<label for="num1">Number 1:</label>

<input type="number" id="num1" name="num1" required><br><br>

<label for="num2">Number 2:</label>

<input type="number" id="num2" name="num2" required><br><br>

<label for="num3">Number 3:</label>

<input type="number" id="num3" name="num3" required><br><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

```

**Servlet to Find Greatest Number (FindGreatestServlet.java)**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/FindGreatestServlet")

public class FindGreatestServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

int num1 = Integer.parseInt(request.getParameter("num1"));

int num2 = Integer.parseInt(request.getParameter("num2"));

int num3 = Integer.parseInt(request.getParameter("num3"));

int greatest = Math.max(num1, Math.max(num2, num3));

if (greatest % 2 == 0) {

response.sendRedirect("Even.jsp");

} else {

response.sendRedirect("Odd.jsp");

}

}

}

```

**JSP Pages (Even.jsp and Odd.jsp)**

**```jsp**

<!-- Even.jsp -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Even Number</title>

</head>

<body>

<h2>The greatest number is Even</h2>

</body>

</html>

```

**```jsp**

**<!-- Odd.jsp -->**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Odd Number</title>

</head>

<body>

<h2>The greatest number is Odd</h2>

</body>

</html>

```

Explanation:

- inputForm.html: This HTML page contains a form where users can input three numbers.

- FindGreatestServlet.java: This servlet receives the form data, finds the greatest number among the three inputs, and redirects to either Even.jsp or Odd.jsp based on whether the greatest number is even or odd.

- Even.jsp and Odd.jsp: These JSP pages display a message based on whether the greatest number is even or odd.

Usage:

1. Deploy the servlet, HTML, and JSP files to a servlet container like Apache Tomcat.

2. Access the inputForm.html page from a web browser.

3. Enter three numbers in the form and submit.

4. The servlet will find the greatest number and redirect to either Even.jsp or Odd.jsp based on whether the greatest number is even or odd.

5. The respective JSP page will display a message indicating whether the greatest number is even or odd.

Design a shopping cart for an apparel store with session tracking API.

To design a shopping cart for an apparel store with session tracking, we can use a combination of HTML for the front-end, Java Servlets for server-side processing, and session tracking to maintain the user's cart across multiple requests. Below is an outline of how to implement this:

1. Front-end (HTML/CSS/JavaScript):

Product Listing Page (index.html):

- Display the list of apparel products with details like name, price, and an "Add to Cart" button.

- Each "Add to Cart" button should trigger JavaScript to send a request to the server to add the selected product to the cart.

Shopping Cart Page (cart.html):

- Display the items added to the cart with details like product name, price, quantity, and a total.

- Provide options to update quantity or remove items from the cart.

- Include a "Checkout" button to proceed to the checkout process.

2. Server-side (Java Servlets):

AddToCartServlet:

- Handles requests to add products to the cart.

- Retrieves product details from the request parameters.

- Adds the product to the user's cart stored in the session.

UpdateCartServlet:

- Handles requests to update the quantity of items in the cart.

- Retrieves product ID and new quantity from the request parameters.

- Updates the quantity of the corresponding item in the user's cart stored in the session.

RemoveFromCartServlet:

- Handles requests to remove items from the cart.

- Retrieves product ID from the request parameters.

- Removes the corresponding item from the user's cart stored in the session.

3. Session Tracking:

Session Management:

- Use session tracking to maintain the user's cart across multiple requests.

- Create a session when the user adds the first item to the cart.

- Store the cart as a session attribute and update it as the user adds/removes items.

Example Workflow:

1. User visits the product listing page (index.html).

2. User selects apparel products and clicks "Add to Cart".

3. JavaScript sends an AJAX request to the server (AddToCartServlet) to add the product to the cart.

4. AddToCartServlet retrieves product details and adds them to the user's cart stored in the session.

5. User navigates to the shopping cart page (cart.html) to view added items.

6. User can update quantity or remove items from the cart.

7. User proceeds to checkout.

Additional Considerations:

- Implement validation to ensure only valid product IDs are added to the cart.

- Handle edge cases like out-of-stock products, invalid requests, etc.

- Implement security measures to prevent common vulnerabilities like session fixation, session hijacking, etc.

- Enhance user experience with features like persistent carts (using cookies), pagination for large product lists, etc.

By following these steps and considerations, you can design a shopping cart for an apparel store with session tracking capabilities.

Create a webpage that includes various form elements such as Text area, select, text box, checkboxes, and radio buttons. Implement functionality using JavaScript to display the data entered by the user.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Form Elements Demo</title>

<style>

.output {

margin-top: 20px;

padding: 10px;

border: 1px solid #ccc;

}

</style>

</head>

<body>

<h2>Form Elements Demo</h2>

<form id="myForm">

<label for="name">Name:</label><br>

<input type="text" id="name" name="name"><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email"><br>

<label for="message">Message:</label><br>

<textarea id="message" name="message" rows="4" cols="30"></textarea><br>

<label for="gender">Gender:</label><br>

<input type="radio" id="male" name="gender" value="male">

<label for="male">Male</label>

<input type="radio" id="female" name="gender" value="female">

<label for="female">Female</label><br>

<label for="country">Country:</label><br>

<select id="country" name="country">

<option value="USA">USA</option>

<option value="Canada">Canada</option>

<option value="UK">UK</option>

<option value="Australia">Australia</option>

</select><br>

<label for="subscribe">Subscribe:</label><br>

<input type="checkbox" id="subscribe" name="subscribe" value="yes">

<label for="subscribe">Yes, I want to subscribe to updates</label><br>

<input type="submit" value="Submit">

</form>

<div class="output" id="output">

<!-- Output will be displayed here -->

</div>

<script>

document.getElementById("myForm").addEventListener("submit", function(event) {

event.preventDefault();

displayFormData();

});

function displayFormData() {

var formData = new FormData(document.getElementById("myForm"));

var output = "<h3>Form Data:</h3>";

formData.forEach(function(value, key) {

output += "<p><strong>" + key + ":</strong> " + value + "</p>";

});

document.getElementById("output").innerHTML = output;

}

</script>

</body>

</html>

Design a table with the following fields: name, password, email-id, and phone number. Write a program using Servlets or JavaServer Pages (JSP) to connect to the database, retrieve data from the table, and display it. Additionally, implement functionality to insert user details into the table when a new user registers by clicking the submit button on the registration page.

Here's an example of how you can implement the described functionality using Java Servlets:

**### 1. Create a MySQL Database Table:**

**```sql**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

password VARCHAR(100),

email VARCHAR(100),

phone\_number VARCHAR(15)

);

```

**### 2. HTML Registration Form (registration.html):**

**```html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Registration</title>

</head>

<body>

<h2>User Registration Form</h2>

<form action="RegisterServlet" method="post">

<label>Name:</label><br>

<input type="text" name="name"><br>

<label>Password:</label><br>

<input type="password" name="password"><br>

<label>Email:</label><br>

<input type="email" name="email"><br>

<label>Phone Number:</label><br>

<input type="tel" name="phone\_number"><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

```

**### 3. Servlet to Handle Registration (RegisterServlet.java):**

**```java**

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/RegisterServlet")

public class RegisterServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String name = request.getParameter("name");

String password = request.getParameter("password");

String email = request.getParameter("email");

String phoneNumber = request.getParameter("phone\_number");

// JDBC Connection Parameters

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String username = "your\_username";

String dbPassword = "your\_password";

try (Connection conn = DriverManager.getConnection(url, username, dbPassword)) {

String sql = "INSERT INTO users (name, password, email, phone\_number) VALUES (?, ?, ?, ?)";

PreparedStatement statement = conn.prepareStatement(sql);

statement.setString(1, name);

statement.setString(2, password);

statement.setString(3, email);

statement.setString(4, phoneNumber);

statement.executeUpdate();

response.sendRedirect("registration.html"); // Redirect back to registration page after successful registration

} catch (SQLException e) {

e.printStackTrace();

}

}

}

```

**### 4. Servlet to Retrieve Data (RetrieveDataServlet.java):**

**```java**

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/RetrieveDataServlet")

public class RetrieveDataServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// JDBC Connection Parameters

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String username = "your\_username";

String dbPassword = "your\_password";

try (Connection conn = DriverManager.getConnection(url, username, dbPassword)) {

String sql = "SELECT \* FROM users";

PreparedStatement statement = conn.prepareStatement(sql);

ResultSet resultSet = statement.executeQuery();

// Display data on the webpage

response.setContentType("text/html");

response.getWriter().println("<html><body><h2>User Details</h2><table border='1'><tr><th>Name</th><th>Email</th><th>Phone Number</th></tr>");

while (resultSet.next()) {

String name = resultSet.getString("name");

String email = resultSet.getString("email");

String phoneNumber = resultSet.getString("phone\_number");

response.getWriter().println("<tr><td>" + name + "</td><td>" + email + "</td><td>" + phoneNumber + "</td></tr>");

}

response.getWriter().println("</table></body></html>");

} catch (SQLException e) {

e.printStackTrace();

}

}

}

```

### 5. Access the Registration Form and Display User Data:

- Access the registration form by navigating to `http://localhost:8080/your\_web\_app/registration.html`.

- Users can register by filling out the form and clicking the submit button.

- User data can be retrieved and displayed by accessing the servlet mapping defined in `RetrieveDataServlet.java`.

### Note:

- Replace `"your\_database\_name"`, `"your\_username"`, and `"your\_password"` with your actual database credentials.

- Ensure that you have the MySQL JDBC driver (`mysql-connector-java.jar`) in your project's classpath.

- Don't forget to handle exceptions and perform proper error handling in your application.

Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet/ JSP for doing the following. Create a Cookie and add these four user id’s and passwords to this Cookie. 2. Read the user id and passwords entered in the Login form and authenticate with the values available in the cookies.

Here's an example of how you can implement the described functionality using Java Servlets:

**### 1. Servlet to Create and Store Cookies (CookieServlet.java):**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/CookieServlet")

public class CookieServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Create cookies for user ids and passwords

Cookie user1Cookie = new Cookie("user1", "pwd1");

Cookie user2Cookie = new Cookie("user2", "pwd2");

Cookie user3Cookie = new Cookie("user3", "pwd3");

Cookie user4Cookie = new Cookie("user4", "pwd4");

// Add cookies to the response

response.addCookie(user1Cookie);

response.addCookie(user2Cookie);

response.addCookie(user3Cookie);

response.addCookie(user4Cookie);

response.getWriter().println("Cookies created successfully.");

}

}

```

**### 2. HTML Login Form (login.html):**

**```html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

</head>

<body>

<h2>Login Form</h2>

<form action="LoginServlet" method="post">

<label>User ID:</label><br>

<input type="text" name="userId"><br>

<label>Password:</label><br>

<input type="password" name="password"><br>

<input type="submit" value="Login">

</form>

</body>

</html>

```

**### 3. Servlet to Authenticate User (LoginServlet.java):**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String userId = request.getParameter("userId");

String password = request.getParameter("password");

// Retrieve cookies

Cookie[] cookies = request.getCookies();

boolean isAuthenticated = false;

if (cookies != null) {

for (Cookie cookie : cookies) {

if (cookie.getName().equals(userId) && cookie.getValue().equals(password)) {

isAuthenticated = true;

break;

}

}

}

if (isAuthenticated) {

response.getWriter().println("Login successful.");

} else {

response.getWriter().println("Invalid user ID or password.");

}

}

}

```

### Usage:

1. Access the login form by navigating to `http://localhost:8080/your\_web\_app/login.html`.

2. Enter the user ID and password.

3. Click the login button.

4. The servlet will authenticate the user based on the cookies created earlier in `CookieServlet.java`.

### Note:

- This is a basic example. In a real-world scenario, you should never store passwords directly in cookies. Always hash them securely.

- Ensure proper error handling and validation in your application.

Write a JSP/Servlet program using preparedStatement in entering data in MySQL/ Oracle table. Also display the result

Below is a simple example of a JSP program that maintains session using the session object:

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Session Example</title>

</head>

<body>

<h2>Session Example</h2>

<%-- Check if a session already exists --%>

<% if (session.getAttribute("username") == null) { %>

<form action="LoginServlet" method="post">

<label>Username:</label><br>

<input type="text" name="username"><br>

<input type="submit" value="Login">

</form>

<% } else { %>

<%-- Display welcome message if session exists --%>

<p>Welcome, <%= session.getAttribute("username") %>!</p>

<form action="LogoutServlet" method="post">

<input type="submit" value="Logout">

</form>

<% } %>

</body>

</html>

```

This JSP program checks if a session exists. If a session does not exist (i.e., the user is not logged in), it displays a login form. If a session exists (i.e., the user is logged in), it displays a welcome message along with a logout button.

To complement this JSP program, you will also need to implement the Servlets to handle login and logout functionality:

**### LoginServlet.java**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String username = request.getParameter("username");

// Create a session and store the username

HttpSession session = request.getSession();

session.setAttribute("username", username);

// Redirect to the JSP page

response.sendRedirect("session-example.jsp");

}

}

```

### LogoutServlet.java

```java

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/LogoutServlet")

public class LogoutServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Invalidate the session

HttpSession session = request.getSession();

session.invalidate();

// Redirect to the JSP page

response.sendRedirect("session-example.jsp");

}

}

```

In this example, when the user logs in, the username is stored in the session. When the user logs out, the session is invalidated. The JSP page checks for the existence of the session and displays the appropriate content accordingly.

Write a JSP/Servlet program to generate the exponential of a user entered value.

Below is an example of a JSP/Servlet program to generate the exponential of a user-entered value:

**### ExponentialForm.jsp:**

**```jsp**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Exponential Calculator</title>

</head>

<body>

<h2>Exponential Calculator</h2>

<form action="ExponentialServlet" method="post">

<label>Enter a number:</label><br>

<input type="number" name="number"><br>

<input type="submit" value="Calculate Exponential">

</form>

</body>

</html>

```

**### ExponentialServlet.java:**

**```java**

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/ExponentialServlet")

public class ExponentialServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Retrieve the user-entered number

double number = Double.parseDouble(request.getParameter("number"));

// Calculate the exponential

double exponential = Math.exp(number);

// Display the result

response.setContentType("text/html");

response.getWriter().println("<html><body><h2>Exponential Result</h2>");

response.getWriter().println("Exponential of " + number + " is: " + exponential);

response.getWriter().println("</body></html>");

}

}

```

In this example, the user enters a number in the form generated by ExponentialForm.jsp. When the form is submitted, the value is sent to the ExponentialServlet, where the exponential of the entered number is calculated using the Math.exp() method. Finally, the result is displayed back to the user.