|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **USN** |  |  |  |  |  |  |  |  |  |  | | | | | | |  | | | | |
| **Internal Assessment Test I – March. 2018** | | | | | | | | | | |
| **Sub:** | Advanced Java Programming | | | | | **Sub Code:** | 16MCA41 | **Branch:** | MCA | |
| **Date:** | 14/03/2018 | **Duration**: | 90 min’s | **Max Marks**: | 50 | **Sem / Sec:** | IV | | |  |

**1.Define Servlet. Explain the basic servlet structure and its life cycle methods**

Java Servlets are programs that run on a Web or Application server

 Act as a middle layer between a request coming from a Web browser or other HTTP

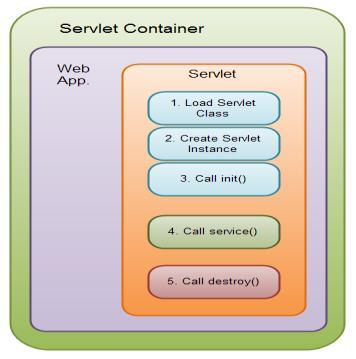
client and databases or applications on the HTTP server.

 Using Servlets, you can collect input from users through web page forms, present records

from a database or another source, and create web pages dynamically.

 Servlets are server side components that provide a powerful mechanism for developing

web applications. A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet



 The servlet is initialized by calling the init () method.

 The servlet calls service() method to process a client's request.

 The servlet is terminated by calling the destroy() method.

 Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

 The init method is designed to be called only once.

 It is called when the servlet is first created, and not called again for each user request. So, it is

used for one-time initializations, just as with the init method of applets.

 The servlet is normally created when a user first invokes a URL corresponding to the servlet,

but you can also specify that the servlet be loaded when the server is first started.

 The init() method simply creates or loads some data that will be used throughout the life of

the servlet.

The init method definition looks like this:

public void init() throws ServletException {

// Initialization code...

}

The service() method :

 The service() method is the main method to perform the actual task.

 The servlet container (i.e. web server) calls the service() method to handle requests coming

from the client( browsers) and to write the formatted response back to the client.

 Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE,

etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Signature of service method:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

{

}

 The service () method is called by the container and service method invokes doGe, doPost,

doPut, doDelete, etc.methods as appropriate.

 So you have nothing to do with service() method but you override either doGet() or doPost()

depending on what type of request you receive from the client.

 The doGet() and doPost() are most frequently used methods with in each service request.

Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no

METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

// Servlet code

}

The destroy() method :

 The destroy() method is called only once at the end of the life cycle of a servlet.

 This method gives your servlet a chance to close database connections, halt background

threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

 After the destroy() method is called, the servlet object is marked for garbage collection.

The destroy method definition looks like this:

public void destroy() {

// Finalization code...

}

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ServletTemplate extends HttpServlet {

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

// Use "request" to read incoming HTTP headers

// (e.g., cookies) and query data from HTML forms.

// Use "response" to specify the HTTP response status

// code and headers (e.g., the content type, cookies).

PrintWriter out = response.getWriter();

// Use "out" to send content to browser.

}

}

Servlets typically extend HttpServlet and override doGet or doPost, depending on whether the data is being sent by GET or by POST. If we want a servlet to take the same action for both GET and POST requests, simply have doGet call doPost, or vice versa. Both doGet and doPost take two arguments: an HttpServletRequest and an HttpServletResponse. The HttpServletRequest lets us get at all of the incoming data; the class has methods by which we can find out about information such as form (query) data, HTTP request headers, and the client’s hostname. The HttpServletResponse lets us specify outgoing information such as HTTP status codes (200, 404, etc.) and response headers (Content-Type, Set-Cookie, etc.). Most importantly, HttpServletResponse lets us obtain a PrintWriter that we use to send document content back to the client. For simple servlets, most of the effort is spent in println statements that generate the desired page.

**2. List and explain the core classes and interfaces that are provided in javax.servlet package.**

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

|  |  |  |  |
| --- | --- | --- | --- |
| **Interface Summary** | | | |
| [**Filter**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Filter.html) | A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both. | | |
| [**FilterChain**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/FilterChain.html) | A FilterChain is an object provided by the servlet container to the developer giving a view into the invocation chain of a filtered request for a resource. | | |
| [**FilterConfig**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/FilterConfig.html) | A filter configuration object used by a servlet container to pass information to a filter during initialization. | | |
| [**RequestDispatcher**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/RequestDispatcher.html) | Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. | | |
| [**Servlet**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Servlet.html) | Defines methods that all servlets must implement. | | |
| [**ServletConfig**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletConfig.html) | A servlet configuration object used by a servlet container to pass information to a servlet during initialization. | | |
| [**ServletContext**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContext.html) | Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file. | | |
| [**ServletContextAttributeListener**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContextAttributeListener.html) | Implementations of this interface receive notifications of changes to the attribute list on the servlet context of a web application. | | |
| [**ServletContextListener**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContextListener.html) | Implementations of this interface receive notifications about changes to the servlet context of the web application they are part of. | | |
| [**ServletRequest**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequest.html) | Defines an object to provide client request information to a servlet. | | |
| [**ServletRequestAttributeListener**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequestAttributeListener.html) | A ServletRequestAttributeListener can be implemented by the developer interested in being notified of request attribute changes. | | |
| [**ServletRequestListener**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequestListener.html) | A ServletRequestListener can be implemented by the developer interested in being notified of requests coming in and out of scope in a web component. | | |
| [**ServletResponse**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletResponse.html) | Defines an object to assist a servlet in sending a response to the client. | | |
| [**SingleThreadModel**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/SingleThreadModel.html) | **Deprecated.** *As of Java Servlet API 2.4, with no direct replacement.* | | |
| **Class Summary** | | | |
| [**GenericServlet**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/GenericServlet.html) | | | Defines a generic, protocol-independent servlet. |
| [**ServletContextAttributeEvent**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContextAttributeEvent.html) | | | This is the event class for notifications about changes to the attributes of the servlet context of a web application. |
| [**ServletContextEvent**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContextEvent.html) | | | This is the event class for notifications about changes to the servlet context of a web application. |
| [**ServletInputStream**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletInputStream.html) | | | Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time. |
| [**ServletOutputStream**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletOutputStream.html) | | | Provides an output stream for sending binary data to the client. |
| [**ServletRequestAttributeEvent**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequestAttributeEvent.html) | | | This is the event class for notifications of changes to the attributes of the servlet request in an application. |
| [**ServletRequestEvent**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequestEvent.html) | | | Events of this kind indicate lifecycle events for a ServletRequest. |
| [**ServletRequestWrapper**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletRequestWrapper.html) | | | Provides a convenient implementation of the ServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet. |
| [**ServletResponseWrapper**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletResponseWrapper.html) | | | Provides a convenient implementation of the ServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet. |
| **Exception Summary** | | | |
| [**ServletException**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletException.html) | | Defines a general exception a servlet can throw when it encounters difficulty. | |
| [**UnavailableException**](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/UnavailableException.html) | | Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **3** | | **a)Briefly explain the different HTTP 1.1 request header** | |
|  | | **3. a)Briefly explain the different HTTP 1.1 request header** | |

When a browser requests for a web page, it sends lot of information to the web server which can not be read directly because this information travel as a part of header of HTTP request. You can check HTTP Protocol for more information on this.

|  |  |
| --- | --- |
|  |  |
| Header | Description |
|  |  |
|  |  |
| Accept | This header specifies the MIME types that the browser or other clients |
|  | can handle. Values of **image/png** or**image/jpeg** are the two most |
|  | common possibilities. |
|  |  |
| Accept-Charset | This header specifies the character sets the browser can use to display |
|  | the information. For example ISO-8859-1. |
|  |  |
| Accept-Encoding | This header specifies the types of encodings that the browser knows |
|  | how to handle. Values of **gzip** or**compress** are the two most common |
|  | possibilities. |
|  |  |
| Accept-Language | This header specifies the client's preferred languages in case the servlet |
|  | can produce results in more than one language. For example en, en-us, |
|  | ru, etc. |
|  |  |
| Authorization | This header is used by clients to identify themselves when accessing |
|  | password-protected Web pages. |
|  |  |
| Connection | This header indicates whether the client can handle persistent HTTP |
|  | connections. Persistent connections permit the client or other browser |
|  | to retrieve multiple files with a single request. A value of **Keep-** |
|  | **Alive** means that persistent connections should be used |
|  |  |
| Content-Length | This header is applicable only to POST requests and gives the size of |
|  | the POST data in bytes. |
|  |  |
| Cookie | This header returns cookies to servers that previously sent them to the |
|  | browser. |
| Host | This header specifies the host and port as given in the original URL. |
| If-Modified-Since | This header indicates that the client wants the page only if it has been |
|  | changed after the specified date. The server sends a code, 304 which |
|  | means Not Modifiedheader if no newer result is available. |
| If-Unmodified- | This header is the reverse of If-Modified-Since; it specifies that the |
| Since | operation should succeed only if the document is older than the |
|  | specified date. |
| Referer | This header indicates the URL of the referring Web page. For example, |
|  | if you are at Web page 1 and click on a link to Web page 2, the URL of |
|  | Web page 1 is included in the Referer header when the browser |
|  | requests Web page 2. |
| User-Agent | This header identifies the browser or other client making the request |
|  | and can be used to return different content to different types of |
|  | browsers. |

**b)Write the differences between JSP and servlets.**

|  |  |
| --- | --- |
| JSP | Servlets |
| [JSP is a webpage scripting language](http://www.withoutbook.com/Technology.php?tech=2&subject=JSP%20Interview%20Questions%20and%20Answers) that can generate dynamic content. | [Servlets are Java programs](http://www.withoutbook.com/Technology.php?tech=3&subject=Servlets%20Interview%20Questions%20and%20Answers) that are already compiled which also creates dynamic web content. |
| JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets. | Servlets run faster compared to JSP. |
| It’s easier to code in JSP than in Java Servlets. | Its little much code to write here. |
| In MVC, jsp act as a view. | In MVC, servlet act as a controller. |
| JSP are generally preferred when there is not much processing of data required. | servlets are best for use when there is more processing and manipulation involved. |
| The advantage of JSP programming over servlets is that we can build [custom tags](http://www.withoutbook.com/InterviewQuestionAnswer.php?tech=2&quesId=458&subject=JSP%20Interview%20Questions%20and%20Answers) which can directly call [Java beans](http://www.withoutbook.com/Technology.php?tech=35&subject=Java%20Beans%20Interview%20Questions%20and%20Answers). | There is no such custom tag facility in servlets. |
| We can achieve functionality of JSP at client side by running [JavaScript](http://www.withoutbook.com/Technology.php?tech=8&subject=JavaScript%20Interview%20Questions%20and%20Answers) at client side. | There are no such methods for servlets. |

|  |  |
| --- | --- |
| 4 | a) Explain any five attributes of page directive with an example. |

**4. a) Explain any five attributes of page directive with an example.**

i) import

The import attribute of the page directive lets us specify the packages that should be imported by the servlet into which the JSP page gets translated.

By default, the servlet imports java.lang.\*, javax.servlet.\*, javax.servlet.jsp.\*, javax.servlet.http.\*, and possibly some number of server-specific entries. Never write JSP code that relies on any server-specific classes being imported automatically; doing so makes your code nonportable. Use of the import attribute takes one of the following form

<% @ page import =”package.classname” %>

Ex: <% @ page import=”java.util.\*” %>

ii) errorPage and isErrorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

//index.jsp

<html>

<body>

<%@ page errorPage="myerrorpage.jsp" %>

<%= 100/0 %>

</body>

</html>

The isErrorPage attribute is used to declare that the current page is the error page.

<html>

<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>

The exception is: <%= exception %>

 </body>

</html>

iii) Content Type

The contentType attribute sets the Content-Type response header, indicating the MIME type of the document being sent to the client.

Ex:

Capture.PNG

iv) buffer and autoflush

The buffer attribute specifies the size of the buffer used by the out variable, which is of type JspWriter. Use of this attribute takes one of two forms:

Capture.PNG

The autoFlush attribute controls whether the output buffer should be automatically flushed when it is full (the default) or whether an exception should be raised when the buffer overflows (autoFlush="false"). Use of this attribute takes one of the following two forms.

Capture.PNG

**4.b)Write a java servlet program to auto page refresh**

**package** j2ee.prg2;

**import** java.io.\*;

**import** java.util.Date;

**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** java.util.\*;

/\*\*

\* Servlet implementation class program2

\*/

@WebServlet("/program2")

**public** **class** program2 **extends** HttpServlet {

private static final long *serialVersionUID* = 1L;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** program2() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// **TODO** Auto-generated method stub

performTask(request,response);

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

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

// **TODO** Auto-generated method stub

performTask(request,response);

}

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

{

// Setting the HTTP Content-Type response header to text/html

response.setContentType("text/html");

//Adds a response header to refresh the webpage in every one second.

response.addHeader("Refresh","1");

// Returns a PrintWriter object that can send character text to the client.

PrintWriter out=response.getWriter();

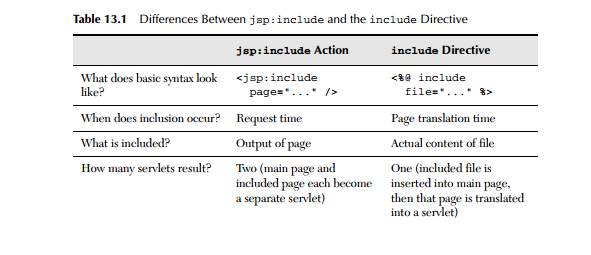
//writing the output in the html format

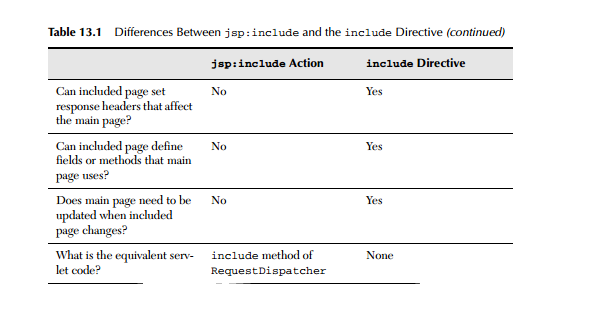
out.println("Text servlet says hi at "+**new** Date());

}

}

**5.a)Write the differences between <jsp:include> action tag and include directive**





**b) Explain the different types of JSP tags with an example.**

1) Expression Tag: ( <%= … %> )

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java

Language Specification but you cannot use a semicolon to end an expression.

Syntax two forms:

<%= expr %>

<jsp:expression> expr </jsp:expression> (XML form)

2) Scriptlet Tag ( <% … %> )

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Embeds Java code in the JSP document that will be executed each time the JSP page is processed.

Code is inserted in the service() method of the generated Servlet

Syntax two forms:

<% any java code %>

<jsp:scriptlet> ... </jsp:scriptlet>. (XML form)

Example

– <% if (Math.random() < 0.5) { %>

Have a <B>nice</B> day! <% } else { %>

Have a <B>lousy</B> day! <% } %>

• Representative result

– if (Math.random() < 0.5) { out.println("Have a <B>nice</B> day!"); } else {

out.println("Have a <B>lousy</B> day!");

}

3) Declaration Tag ( <%! … %> )

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Code is inserted in the body of the servlet class, outside the service method.

o May declare instance variables.

o May declare (private) member functions.

Syntax two forms:

<%! declaration %>

<jsp:declaration> declaration(s)</jsp:declaration>

Example for declaration of Instance Variable:

<html>

<body>

<%! private int accessCount = 0; %>

<p> Accesses to page since server reboot:

<%= ++accessCount %> </p>

</body></html>

4) Directive Tag ( <%@ … %> )

Directives are used to convey special processing information about the page to the JSP container.

The Directive tag commands the JSP virtual engine to perform a specific task, such as importing a Java package required by objects and methods.

Directive Description

<%@ page ... %> Defines page-dependent attributes, such as

scripting language, error page, and buffering

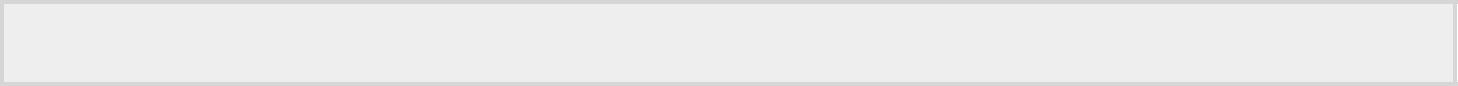
requirements.

<%@ include ... %> Includes a file during the translation phase.

<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

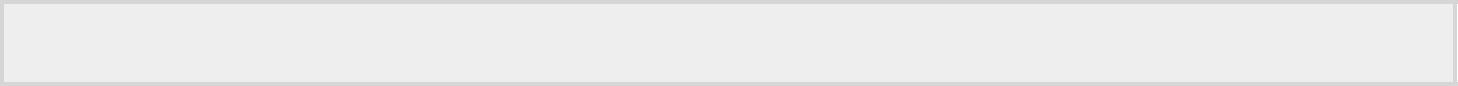
The page directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of page directive:



<%@ page attribute="value" %>

You can write XML equivalent of the above syntax as follows:



<jsp:directive.page attribute="value" />

**6.a) Explain different types of JDBC Drivers**

JDBC driver specification classifies JDBC drivers into four groups.

They are…

**Type 1: JDBC-to-ODBC Driver**

 Microsoft created ODBC (Open Database Connection), which is the basis from which

Sun created JDBC. Both have similar driver specifications and an API.

 The JDBC-to-ODBC driver, also called the JDBC/ODBC Bridge, is used to translate

DBMS calls between the JDBC specification and the ODBC specification.

 MS Access and SQL Server contains ODBC driver written in C language using pointers,

but java does not support the mechanism to handle pointers.

 So JDBC-ODBC Driver is created as a bridge between the two so that JDBC-ODBC

bridge driver translates the JDBC API to the ODBC API.

 Type-1 ODBC Driver for MS Access and SQL Server

Drawbacks of Type-I Driver:

o ODBC binary code must be loaded on each client.

o Transaction overhead between JDBC and ODBC.

o It doesn‟t support all features of Java.

o It works only under Microsoft, SUN operating systems.

**Type 2: Java/Native Code Driver or Native-API Partly Java Driver**

 It converts JDBC calls into calls on client API for DBMS.

 The driver directly communicates with database servers and therefore some database

client software must be loaded on each client machine and limiting its usefulness for

internet

 The Java/Native Code driver uses Java classes to generate platform- specific code that is

code only understood by a specific DBMS.

Ex: Driver for DB2, Informix, Intersoly, Oracle Driver, WebLogic drivers

Drawbacks of Type-I Driver:

o Some database client software must be loaded on each client machine

o Loss of some portability of code.

o Limited functionality

o The API classes for the Java/Native Code driver probably won‟t work with

another manufacturer‟s DBMS.

**Type 3: Net-Protocol All-Java Driver**

 It is completely implemented in java, hence it is called pure java driver. It translates the

JDBC calls into vendor‟s specific protocol which is translated into DBMS protocol by a

middleware server

 Also referred to as the Java Protocol, most commonly used JDBC driver.

 The Type 3 JDBC driver converts SQL queries into JDBC- formatted statements, in-turn

they are translated into the format required by the DBMS.

Ex: Symantec DB

Drawbacks:

 It does not support all network protocols.

 Every time the net driver is based on other network protocols.

**Type 4: Native-Protocol All-Java Driver or Pure Java Driver**

 Type 4 JDBC driver is also known as the Type 4 database protocol.

 The driver is similar to Type 3 JDBC driver except SQL queries are translated into the

format required by the DBMS.

 SQL queries do not need to be converted to JDBC-formatted systems.

 This is the fastest way to communicated SQL queries to the DBMS.

 Here the driver uses network protocol this protocol is already built-into the database

engine; here the driver talks directly to the database using java sockets. This driver is

better than all other drivers, because this driver supports all network protocols.

 Use Java networking libraries to talk directly to database engines

Ex: Oracle, MYSQL

**Only disadvantage:** need to download a new driver for each database engine

**b) How servlets are better in comparison with CGI?**

**CGI (Commmon Gateway Interface) :**

1. CGI (Common Gateway Interface) is used to provide dynamic content to the user.

2. CGI is used to execute a program that resides in the server to process data or access databases to produce the relevant dynamic content.

3. Programs that resides in server can be written in native operating system such as C++.

**Diagrammatic Representation :**

****

We have listed some problems in CGI technology –

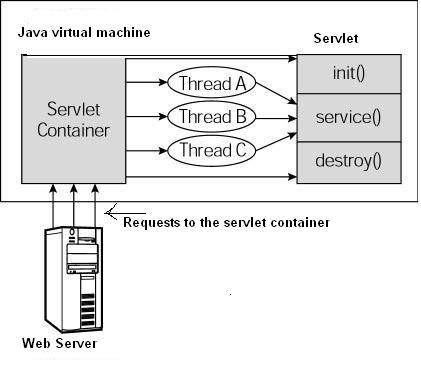
**Disadvantages of CGI :**

1. For each request CGI Server receives, It creates new Operating System Process.

2. If the number of requests from the client increases then more time it will take to respond to the request.

3. As programs executed by CGI Script are written in the native languages such as C, C++, perl which areplatform independent.

**Servlet :**



**CGI:-**

1.It is a “process based” that is for every request a separate process will be created and that is responsible to generate the response.

2.Creation and destruction of new process for every request is costly. If the no. of requests increases, the performance of the system goes down. Hence CGI technology fails to deliver Scalable Applications.

3.Two processes never share the common address space. Hence concurrency never comes in CGI technology.

4.CGI programming can be written in multiple languages. Mostly common used languages PERL

5.Most of the CGI languages are not object oriented. Hence we may miss the benefit of the OOPS.

6.Non-persistant CGI programs are removed from the memory once the request is processed and each time a new process is initiated whenever new request arrives.

7.CGI technology is platform dependent.

**Servlet:-**

1.It is “Thread based” that is for every client request a separate thread will be created and that is responsible for generation of dynamic response.

2.Creation and destruction of a new thread for every request is not Costly. Hence even though no. of requests increases there is no change in the response time and performance. Hence Servlet Technology succeeds to deliver Scalable Applications

3.All the Threads share the same common address space. Hence concurrency problems may rise.

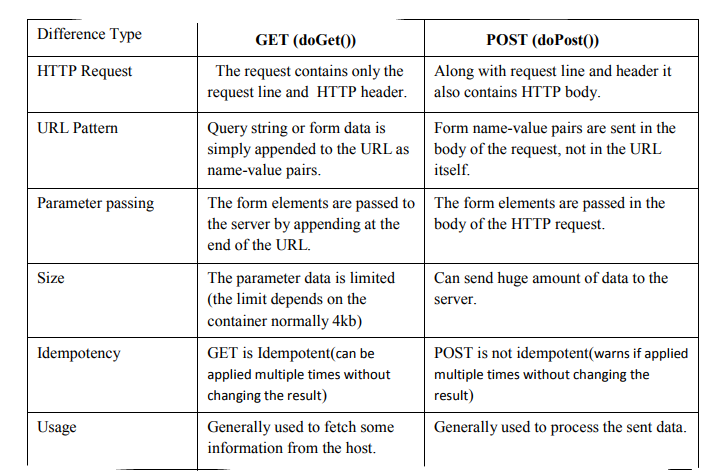
4.Servlet should be written in Java only.

5.Java languages itself object oriented. Hence we can get all benefit of the OOPS.

6. Persistant - Servlets remain in memory until explicitly destroyed.

7.Servlet technology is platform independent.

**7.a)Write the differences between get and post**.



**b) Write a Java servlet program to implement get() and post() methods.**

**package** prg3.j2ee;

**import** java.io.\*;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class prg3

\*/

@WebServlet("/prg3")

**public** **class** prg3 **extends** HttpServlet {

private static final long *serialVersionUID* = 1L;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** prg3() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// Setting the HTTP Content-Type response header to text/html

response.setContentType("text/html");

// Returns a PrintWriter object out that can send character text to the client.

PrintWriter out=response.getWriter();

// To retrieve the optional values (color) from HTML page and store in the string color

String col = request.getParameter("color");

out.println("<html><body bgcolor="+col+">");

out.println("You have selected "+col);

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

out.close();

}

}

index.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<!-- send the form data to url mapping “prg3“ and the get method is used -->

<form method =*"post"* action=*"prg3"*>

<!--Display 3 Colors RED, BLUE, GREEN in the dropdown Box -->

<select name=*"color"* size=*"1"*>

<Option value=*"red"*>RED</Option>

<Option value=*"green"*>GREEN</Option>

<Option value=*"blue"*>BLUE</Option>

</select>

<input type=*"Submit"* value=*"Enter"*>

</form>

</body>

</html>

**8.What is Cookie? Explain the working of Cookie with an example program**

**Cookies**

Cookies are small bits of textual information that a web server sends to a browser and that the browser later returns unchanged when visiting the same web site or domain

Sending cookies to the client:

1.Creating a cookie object

Cookie():constructs a cookie.

Cookie(String name, String value)constructs a cookie with a specified name and value.

EX:

Cookie ck=new Cookie("user",”mca");

2.Setting the maximum age

setMaxAge() is used to specify how long (in seconds) the cookie should be valid.

Ex:cookie.setMaxAge(60\*60\*24);

3.Placing the cookie into the HTTP response headers.

We use **response.addCookie** to add cookies in the HTTP response header as follows:

response.addCookie(cookie);

Reading cookies from the client:

Call request.getCookies(). This yields an array of cookie objects.

Loop down the array, calling getName on each one until you find the cookie of interest.

Ex:

String cookieName=“userID”;

Cookie[] cookies=request.getCookies();

If(cookies!=null)

{

for(int i=0;i<cookies.length;i++){

Cookie cookie=cookies[i];

if(cookieName.equals(cookie.getName())){

doSomethingwith(cookie.getValue());

}}}

EX:

Servlet1.java

**package** j2ee.prg4;

**import** java.io.\*;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class store

\*/

@WebServlet("/store")

**public** **class** store **extends** HttpServlet {

private static final long *serialVersionUID* = 1L;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** store() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

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

// Setting the HTTP Content-Type response header to text/html

response.setContentType("text/html;charset=UTF-8");

// Returns a PrintWriter object that can send character text to the client.

PrintWriter out=response.getWriter();

try

{

//Requesting input color from html page and storing in String variable s1

String s1=request.getParameter("color");

//Checking the color either RED or Green or Blue

**if** (s1.equals("RED")||s1.equals("BLUE")||s1.equals("GREEN"))

{

// Creating cookie object ck1 and storing the selected color

Cookie ck1=**new** Cookie("color",s1);

//adding the cookie to the response

response.addCookie(ck1);

//writing the output in the html format

out.println("<html>");

out.println("<body>");

out.println("You selected: "+s1);

out.println("<form action='retrieve' method='post'>");

out.println("<input type='Submit' value='submit'/>");

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

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

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

}

}

finally

{

//Closing the output object

out.close();

}

}

}

retrieve.java

**package** j2ee.prg4;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class retrieve

\*/

@WebServlet("/retrieve")

**public** **class** retrieve **extends** HttpServlet {

private static final long *serialVersionUID* = 1L;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** retrieve() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

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

// Setting the HTTP Content-Type response header to text/html

response.setContentType("text/html;charset=UTF-8");

// Returns a PrintWriter object that can send character text to the client.

PrintWriter out=response.getWriter();

try

{

//Requesting all the cookies and stored in cookie array ck[]

Cookie ck[]=request.getCookies();

out.println("<html>");

out.println("<head>");

out.println("<title>servlet</title>");

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

// Getting the value from cookie and setting the HTML form background color

out.println("<body bgcolor="+ck[0].getValue()+">");

//Getting the value from cookie and displaying the color name in HTML form

out.println("You selected color is: "+ck[0].getValue()+"</h1>");

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

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

}

finally

{

//closing the printwriter object out

out.close();

}

}

}

Index.jsp

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<!-- send the form data to the url store and the post method is used -->

<form action=*"store"* method=*"post"*>

<!-- Display the Radio button with three option -->

RED:<input type=*"radio"* name=*"color"* value=*"RED"*/><br>

GREEN:<input type=*"radio"* name=*"color"* value=*"GREEN"*/><br>

BLUE:<input type=*"radio"* name=*"color"* value=*"BLUE"*/><br>

<input type=*"submit"* value=*"submit"*/>

</form>

</body>

</html>

**9.With the code snippets Write the steps involved in creating a database.**

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

Register the Driver

Create a Connection

Create SQL Statement

Execute SQL Statement

Closing the connection

Register the Driver

Class.forName() is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Create a Connection

getConnection() method of **DriverManager** class is used to create a connection.

Syntax

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

Example establish connection with Oracle Driver

Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:XE","username","password");

Create SQL Statement

createStatement() method is invoked on current **Connection** object to create a SQL Statement.

Syntax

public Statement createStatement() throws SQLException

Example to create a SQL statement

Statement s=con.createStatement();

Execute SQL Statement

executeQuery() method of **Statement** interface is used to execute SQL statements.

Syntax

public ResultSet executeQuery(String query) throws SQLException

Example to execute a SQL statement

ResultSet rs=s.executeQuery("select \* from user");

while(rs.next())

{

System.out.println(rs.getString(1)+" "+rs.getString(2));

}

Closing the connection

After executing SQL statement you need to close the connection and release the session. The close() method of **Connection** interface is used to close the connection.

Syntax

public void close() throws SQLException

Example of closing a connection

con.close();

**import** java.sql.\*;

**class** OracleCon{

**public** **static** **void** main(String args[]){

**try**{

//step1 load the driver class

Class.forName("oracle.jdbc.driver.OracleDriver");

  //step2 create  the connection object

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object

Statement stmt=con.createStatement();

//step4 execute query

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object

con.close();

  }**catch**(Exception e){ System.out.println(e);}

}

}

**10. Write a Java JSP program to get Student information through a HTML and create a Java Bean class, populate bean and display the information through another JSP**

|  |
| --- |
| student.java  **package** program8;  public class stud  {  **public** String sname;  **public** String rno;  //Set method for Student name  **public** **void** setsname(String name)  {  sname=name;  }  //Get method for Student name  **public** String getsname()  {  **return** sname;  }  //Set method for roll no  **public** **void** setrno(String no)  {  rno=no;  }  //Get method for roll no  **public** String getrno()  {  **return** rno;  }  }  display.jsp  <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- Using the studb bean -->  <jsp:useBean id =*"studb"* scope = *"request"* class = *"program8.stud"*></jsp:useBean>  Student Name : <jsp:getProperty name=*"studb"* property=*"sname"*/><br/>  Roll No. : <jsp:getProperty name=*"studb"* property=*"rno"*/><br/>  </body>  </html>  first.jsp  <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- Create the bean studb and set the property -->  <jsp:useBean id=*"studb"* scope=*"request"* class=*"program8.stud"*></jsp:useBean>  <jsp:setProperty name=*"studb"* property=*'\*'*/>  <jsp:forward page=*"display.jsp"*></jsp:forward>  </body>  </html>  index.html  <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <!-- send the form data to first.jsp -->  <form action=*"first.jsp"*>  Student Name : <input type=*"text"* name = *"sname"*>  Student Roll no : <input type=*"text"* name = *"rno"*>  <input type = *"submit"* value=*"Submit"*/>  </form>  </body>  </html> |