

In the previous Block you have gone through the concepts related to web programming languages such as markup languages like HTML5, XML and scripting languages like JavaScript etc. One common point about these languages is that these languages are interpreted and executed by the browser which results in display of web pages. When you access a web page the client side script files are transferred from the web site which is hosted on a web server to the client side through hypertext transfer protocol. But are these files stored as similar files at the web server or they are generated with the help of a programming language? Simply, if a web site displays standard HTML pages to a client browser, the web server will either have those web pages as standard HTML pages or these HTML pages will be created using a programming language. The web sites where all the pages are in standard HTML are sometimes referred to as static web sites. However, in practice static web sites has limited uses and Web 2.0 web sites are more dynamic and interactive. This means that the pages that may be created for a client may change as per the need of the client, thus, you need some dynamism at the web server level too. The languages that are used to create client pages at the server, based on clients requirements may be categorized as the Server Side Scripting Languages. Some of the common server side scripting languages are PHP, JSP, SERVLET, ASP.NET, PYTHON and many more. In this Block, we have used JSP as the server side scripting language.

This Block discusses about the server side scripting. It consists of four Units: Unit 1 discusses the basic concepts relating to server side scripting. It explains various enterprise level architectures, HTTP methods and the concept of web container.

Unit 2 explains the basic directives and elements of JSP. It also explains the concepts of expressions, and some basic JSP objects.

Unit 3 focuses on exception handling using JSP and use of cookies and sessions in the content of JSP. It also introduces the concept of managing emails using JSP.

Unit 4 presents an example of JSP that uses JDBC and database drivers. It also explains how you can connect to databases and develop applications involving databases.

Unit Continues from Next Page

UNIT 1: THE SERVER SIDE SCRIPTING

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 Server side scripting and its need	
1.3 Web Application Architectures	
1.3.1 N-Tier Architecture	
1.3.2 MVC Architecture	
1.4 Tools for server scripting	
1.5 HTTP Primitives	
1.5.1 Request and Response	
1.5.2 HTTP Methods	
1.6 Web container	
1.7 Summary	
1.9 Answers to Check Your Progress	
1.9 Further Readings	

1.0 INTRODUCTION

In the last Block you have gone through the concepts of client side scripting. A client side script is executed by a browser and displayed as per the stated format. Such code however, in general is static in nature. In order to create dynamic web sites you need to use server side scripting.

This unit explains the concepts of server side scripting. It first defines the need of server side scripting. It explains various types of client server architectural model that are being used to develop flexible dynamic applications. This Unit also describes the role of various HTTP methods which help in transfer of data from the client side to the server side. It also explains the concept of request and response primitives that may be needed in a client server system. Finally the unit explains the concept of a web container with the help of an example.

This Unit thus, provides the basic information about the server side scripting. The remaining three Units of this Block are devoted to one server side scripting language JSP. Please remember that this block only introduces you to the server side scripting. This is one area where tools and technologies are changing at a very rapid rate, therefore, you must keep learning about web programming through suggested further readings on the WWW.

1.1 OBJECTIVES

After going through this Unit you should be able to:

- Define the need and the purpose of server side scripting
- Explain the purpose of tiers of architecture
- Identify several languages for server side scripting

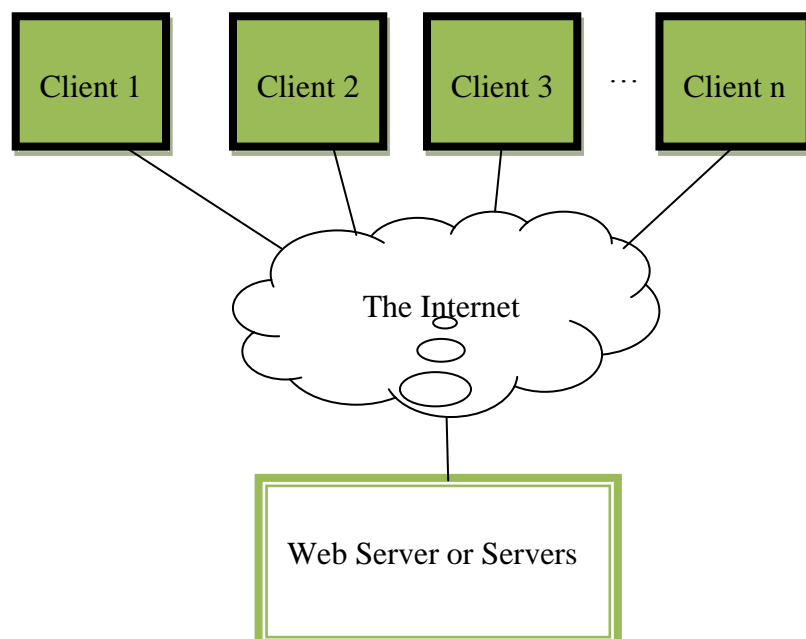
- Work with some of the HTTP methods.
- define request and response primitives and their uses.
- List the features of a web container

1.2 SERVER SIDE SCRIPTING AND ITS NEED

Internet and specially World Wide Web is a major phenomenon of the last two decades. It was first proposed in a paper written by Sir Tim Berner-Lee in the year 1989. The WWW proposal by him was aimed at better communication and was based on the concept of hypertext – a text that can be linked (called hyperlink) to other text document. What should the basic requirements of WWW?

You must have visited WWW and would have visited many web sites. So just try to identify what are the very basic requirements that might be needed for WWW. The first and the foremost is the web address which is a unique address of a resource at a website on WWW. This resource address is also called the Uniform Resource Locator (URL). The second in this context would be a set of rules that allows transfer of the resources (mostly files) from a website to the user. This set of rules was documented as HyperText Transfer Protocol (HTTP). Thirdly, you must agree on some common but simple computer language that allows consistent publishing of the contents (may be in any spoken language). This common computer language is HyperText Markup Language (HTML) which uses standard tags. Now, the question is how does a user reach to a website and where does the website reside?

You as a user of WWW you need an Internet connection which should be running the HTTP and TCP/IP protocols (recollect HTTP is the application layer protocol of TCP/IP). You on your computer use a browser, type a website address which happens to be a URL of a starting page(may be the default index page index.html) of the website. The website responds by sending this page over the Internet and browser interprets and displays this page for you. In the whole process, the browser at your computer acts as a *Client*. The website obviously has to take the responsibility to make sure that client gets the requested page, thus, serves the client. Thus, websites are hosted on a *Server*. Since this server is of WWW commonly called web, the server is called the Web Server. In effect, the basic architecture is the *Client-Server* architecture. Figure 1 shows this architecture. Please note in the figure that there may be many clients for a website.



The implication of this architecture is that you can clearly identify three basic components viz. the Client, the Internet cloud and the Server. The technologies of all these three components can be enhanced separately as long as they follow the three basic tenets, those are, the resources should be named using a standard (URL), a standard communication protocol is used (HTTP) and the contents are transferred to the client using a standard tagging language (HTML). Even all these standards over the period of time can be enhanced, but for this purposes newer client and server software needs to be created. You can relate these facts to development of newer secure protocols like HTTPS, creation and enhancement of client side scripting languages and CSS, and server side scripting technologies. The tremendous growth of WWW in terms of technologies can, thus, be attributed to the simple, flexible architecture and basic technologies.

Static Web pages: As discussed earlier a web page is accessed by the browser of a client computer from the web server using the Internet. For the discussion here let us ignore other client side scripting languages and just assume that web page has just HTML and CSS. The question here is: Does the server stores the web page seen by you exactly that is as HTML and CSS page? If the answer is YES, then the web site designed by you is Static. Please note that a static web site in addition to HTML, CSS can include images, audio, video content etc. You cannot interact with these static web sites as the content is fixed (static). Every user of the web site will see the same information on the web site. The web sites that were created in early stages of WWW were Static. These web sites were useful in showing brochure like fixed content about an organisation. Any changes or corrections in the content were made by the web administrator that too occasionally. For editing, the web designer/administrator may edit the static HTML pages or change images and their links on the HTML pages and reload them on the web server. Some of early tools used for such editing included Notepad which you can find in the accessories Menu of windows. You also needed tools for editing images, audio and video etc. Some of the major task of these designers was creating a web site layout, page linkages and creation of content and presentation format for each web page. Even today some of the pages of your web sites may be static in nature and may be designed using just HTML and CSS. A elementary example of a static web page may be – a static result page that displays the result of all the student in one page.

Dynamic Web pages: With the popularity of WWW, it was felt that the static web sites were not flexible and lacked user interaction. User wanted that a web page should display information as per his requirements. For example, you just need to be shown only your result. This would require that a web page is created that asks for your enrolment number and a password and displays your result to your browser only. There can be many other forms of user interactions that were possible (please refer to section on Web 2.0 in Block 1 Unit 1 of this Course). Thus, your web pages may consist of JavaScript or Java Applet at the client side, as well as they may be created at the server as per your interaction using rich set of server side languages and database tools. However, all these programming activities must be completed in quick time, such that you as a client do not have to unduly wait for information.

Today WWW consists of dynamic and interactive websites. A website can be made dynamic using both the client-side and server-side scripting. On the client side many technologies, like HTML5, CSS3, JavaScript, JSON, AJAX and many others, have been developed. Some of these technologies have been studied by you in Block 1. You must keep studying about these technologies form the WWW and keep updating yourself. The client side scripting can be used to perform activities at the client site that does not require information from the server. Some such activities may include creating effects such as animations, simple games, changing display, and even checking of information in forms that does not require checking at the server etc. The

client-side scripting not only helps in creating some visual effect on the client but also reduces unnecessary communication between the client and server.

However, it is the Server-Side Scripting that has lead to major advances in website design. The basic tenet of server side scripting is that to send a HTML file to a client (browser), you need not store these files on the server as HTML files. Rather, information can be collected from many sources including databases and HTML document can be created and sent to the browser. This makes the websites very dynamic as they can collect information and display formats as per the choices of the clients. The input to server side script is the input obtained from the client. The server-side script is executed at server using the resources of the servers. The final output is created for a client and sent to it for display on the browser. Please note that server-side scripting may generate different output for different browsers, if required. It makes the process of creating web pages simple and programmable. Server side scripting had helped in creating big e-commerce based portals simpler to create and deploy. Can you think of some basic usage of server-side scripting?

Some basic uses of server-side scripting include applications that require users to login to system using a password. However, for such applications you also face problems due to stateless nature of HTTP protocol. What does this stateless protocol means?

It simply means that your current request to a server cannot be related to previous request as in HTTP server is not required to retain information about your previous requests. Therefore, if you are performing a transaction that requires multiple requests, you need to make sure that the information of previous requests is duly recorded. Thus, you need to make special provisions such as creating cookies or sessions that help in performing such tasks. Unit 3 of this Block explains these concepts in more details. However, you should know that the concept of stateless protocol is extremely useful for Internet as storing states take space and time, and given the size of internet and its users it would have made Internet very slow.

Server-side scripting is also very useful in extracting information from the user, may be with the help of a form or otherwise, and processing the information for some useful purpose. Suppose, you are interested in opening a web based email account, then you may fill up an account opening request form of an e-mail service provider like gmail, yahoo mail, rediffmail, etc. When you are filling up the form and commit some catchable mistake, client side JavaScript caches the error. However, once you submit the form and still an error is encountered, like duplicate user name, it can only be caught by the server-side script. As another example, consider that you are buying some product on an e-commerce based website, the client side is expected to keep track of your login information and present selections and sever need to maintain detailed list of your selections during a session. Please note that in general, client side scripts are less secure and are browser dependent. For secure applications, you should use server-side scripting languages.

Server-side scripting is also useful when you want to use a database as a backend to a web based application. Databases are normally stored on a database server which can be directly connected to the web server. Such kind of application has been discussed in the Unit 4 of this Block.

Finally, just some basic advice on when to use client-side scripting and when to use server-side scripting? The answer is very simple – if a user interaction element of a page can be processed without the need of any information from the web server, then you use client-side scripts. It reduces the load on network traffic, but the client hardware should be sufficiently fast to execute the client side script.

1.3 WEB APPLICATION ARCHITECTURES

In the previous section, you have gone through the concept of client-server model of computing. The model discussed in Figure 1, defines two simple portion where computing is performed. The first one is the client, which performs the tasks such as displaying a web page of an HTML document using CSS, running JavaScript, performing data conversion on the client side, dealing with client side Graphical User Interface display, communicating the data of the clients over the Internet to the server. While on the server side data may be stored, updated, or retrieved from database, the data may also be processed and a response may be sent to the client. Thus, the model clearly identifies a Client and a Server.

The objective of such a model is to clearly the division of work providing a flexible and scalable model of computing. This model provides flexibility in the sense that a new client can be added to whole system without much effort. The client only needs to know what server to access through the network and how. Now, consider a client-server environment in which you initially had only single digit number of clients, therefore, one single server was able to serve all the clients. However, the application became popular and hundreds of clients joined. In order to serve these clients you may need to deploy more servers. The important point here is that you can scale the number of servers based on the requirements. Thus, client-server model is a scalable model of computing.

1.3.1 N-Tier Architecture

In order to develop client-server based, you need an application development architecture. One such architecture that can be considered is 2-tier architecture. Obviously, in the 2-tier architecture, the first tier belonging to the client, which may be called a client tier, client tier or frontend, and the second tier may be called the server tier or backend. A developer needs to design the basic interactivity for the application, rules of data access and business rules in the client tier. In present day applications interactivity is provided through GUIs. Thus, all the information pages, forms, etc. designed by you are part of the client tier. Thus, web pages of a web application accessed by you using your computer system and browser are part of the client tier of the web application. The following is an illustration of use of 2-tier architecture.

You (client Computer) using a Browser	Web Server running a Web Application	
	Client Tier/Layer	Server Tier/Layer
Requests index page of a web site by typing the address of the web site	The presentation tier will use server tier files to create the web page HTML+CSS+JavaScript for communication to client computer.	Receives the request and processes it to collect relevant files.
The index page is displayed on the browser. Suppose the web site requests you to login. You will enter the user Id and password and press	The failure message will be converted to the	The server will receive the data. It will connect to database to determine if

Submit.	required web page in HTML format along with CSS and the page will be communicated to the client computer. Please note this layer may also check the number of login attempts.	the user ID and Password are correct. Suppose it is not then it will create the failure message.
The message page will be displayed. You can try again and the process continues...		

Figure 2: An Illustration of 2-Tier Architecture

As a client, you use the client tier to submit your request for a web page or some data. The client tier also checks, if you have permission to access the requested data. In case, user has the access permission, the requested information details are passed to the server, where the server-tier programs services the request sent by you (as a client). Once the requested data is available, then it is sent back to the client. Please note that display of data on your client computer is processed by the client tier of the web application.

The 2-tier client-server applications are useful for distribution of work, however, they require complex client implementations and number of communications between the client and servers. With the popularity of Internet new web application architecture was developed called n-Tier architecture. One of the most common n-tier architecture is 3-tier architecture. Figure 3 shows 3-tier architecture.

The layers and their basic functions are:

Presentation Layer/Tier: The presentation tier of the three tier architecture interfaces with the client. The presentation tier is responsible for displaying the information to the client as well as extracting information from the client. Technically, for a web based application the presentation tier resides on the web server on request the presentation layer displays interactive web pages in the browser of client computers.

Application Logic Layer/ Tier: This is also called the middle ware. This layer is primarily responsible to provide business rules, sharable components of a web application, access control etc. This layer shields the data layer from direct use of the clients. This layer provides an interface between the presentation and data layer.

Data Layer/Tier: The data for a web application may be hosted on a database management system or a file system. The data layer controls the integrity of data residing on some data storage system. The application logic sends queries to data layer which sends back the query results to the application logic layer.

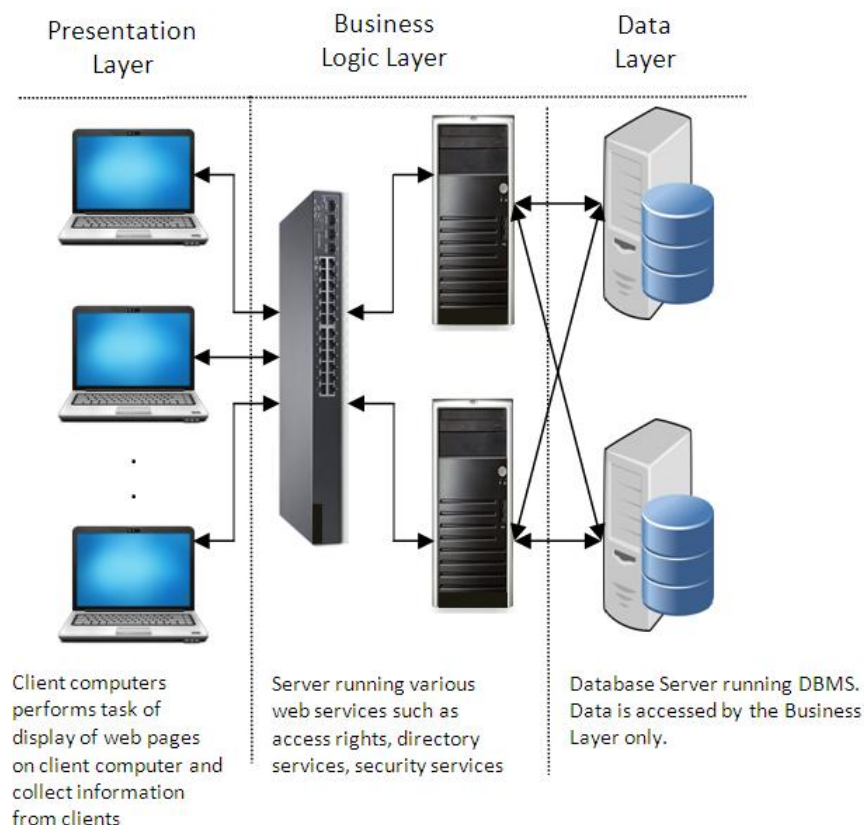


Figure 3: An n-Tier Architecture

Figure 4 shows an illustration of 3-tier Architecture.

Layer / Executed at	Example
Presentation Layer/client computer and web server	User logs in and on Successful login requests the list of students for BCA programme using a GUI based interface created by the web server.
Business Layer/Web Server	On user request the business layer verifies the access constraints of the user, suppose he is only authorized to see a group of students of BCA, then the query to database is suitably modified.
Data Layer/Web Server and Database Server	The query is executed using a connection with the database, and the results are returned to the web server.
Business Layer/Web Server	The web server checks if the returned data set is empty or not. If not passes the data set to the Presentation Layer, else returns an error message.
Presentation Layer/web server and client computer	The presentation layer converts the data into a HTML table or error message to an error page. This page then is sent to client computer and displayed in the browser.

Figure 4: An illustration of 3-Tier Architecture

An n-tier architecture in addition to the above three layers may have many more application layers such as client presentation layer, entity class layer, persistence layer etc. The more are the number of layers the more complex the system is, however, more layers may bring better application flexibility.

In general, n-tier architectures result in more scalable, more secure (database access is hidden) and better integrity based applications. However, they are more complex in nature.

1.3.2 MVC Architecture

In the previous sub-section, you have gone through the concept of n-tier architecture. In this section, we discuss an important architecture that is used by Java for web application development. This architecture is known as Model-View-Controller (MVC). This architecture can also be used for the development of a web application. Figure 6 shows the component of web applications as per this architecture.

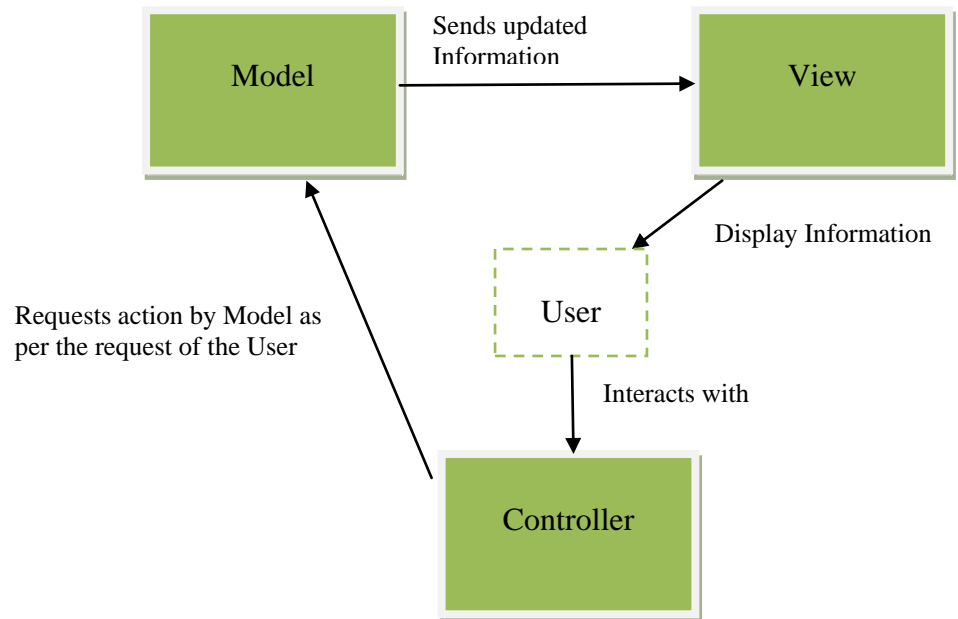


Figure 6: MVC Architecture

The MVC architecture has three functional components of a web application that communicate through a user as well as among them. The following are the basic components of MVC architecture:

Model: In the context of MVC a Model defines the data model and its access controls. The main role of this component is to represent data and perform updates on it as per the defined rules. The main responsibility of this component is to accept user requests and the data entered by the user and perform the necessary data-related function.

View: The function of the view will be to accept the data from the Model and convert it to a form that can be seen by a user in a user-friendly way. Thus, the view is responsible for displaying web pages for the user. Any change in the model should also change the view. This can be achieved using two processes -

Push processes allow views to register with a model and changes in the data of the model result in pushing the current data to the view.

Pull process requires the view to get the current data from the model when needed.

Controller: A user may be allowed to interact with the web pages created by the View component. This interaction may be in the form of selection of options of Menu, pull down lists, check boxes etc., filling up data in text boxes etc. It is the controller which accepts this data from the user and initiates suitable actions that should be carried out by the models.

Let us explain the MVC with the help of an illustration as shown in figure 7.

Model	You decided to create a website for storing information of registered students for various courses on an e-learning website. The model may be the student registration database.
View	As a first time user, you want to register into the website, one of the view would be the registration page. Another view may be the list of courses on offer. Please note that in the view pages the course list will be created from the model. In case a new course is added, this list needs to be updated. How? Not manually, but by a program. You will learn about this in the subsequent Units.
Controller	You may register to a course by submitting the online registration form, the data of this form will be accepted by the controller and sent to model for necessary actions.

Figure 7: An Illustration of MVC

Check Your Progress 1

- 1) Differentiate between Static and Dynamic Web pages?
.....
.....
- 2) What is the need of Server side scripting? How is it different to client side Scripting?
.....
.....
.....
- 3) What are the various components of 3-Tier architecture? Why is it needed?
.....
.....
- 4) Compare 3-Tier Architecture with MVC?
.....
.....

1.4 TOOLS FOR SERVER SIDE SCRIPTING

In this section we will explain about the basic features needed for the server side scripting languages. As discussed in section 1.2, purpose of server-side scripting is program the response of a server to the client side requests. Therefore, the input to server side scripting would be the interactive data input sent by the client using a web browser, and the output is the displayable web page which are displayed at the client's browser.

The question that you need an answer is: What should be the elements of a server side programming language? In order to answer this question, you need to first look at the functions that may be performed by the server side scripting. These are:

- Some mechanism for accessing the data that has been transferred from the client.
- Creation of dynamic web pages in which server side scripting may change the contents of a specific portion of a webpage
- Processing of form data obtained from the client side, this data sometimes may be used to store, update, and retrieve information from the database. All these activities are performed by the server-side scripting
- Handling errors that may occur due to several reasons, such as database access errors, data related errors, or other events.
- Enforcing the data security and integrity and handling constraints.
- Handling creation of sessions between client and server despite HTTP is stateless protocol.
- Creating output pages in a HTML or other client side language form.

Thus, a server side scripting language support following constructs:

- The basic programming constructs like variables, data types, assignment statements, if statement, looping statements, arrays, etc.
- Most of the web programming languages are object oriented programming languages, therefore, they use classes, objects, inheritance hierarchy, containers, collections of classes, error handling etc.
- To communicate with the clients, they support some predefined methods, primitives and protocols. This ensures that data entered by a user is duly sent to a program that handles it. Most of the server side programming languages call these data items as parameters and have various ways to refer to these parameters.
- A good website requires access to a database system to store, update and retrieve data based on user's request. A database is maintained under the control of a DBMS. You are required to use a database driver that makes possible the exchange between the web server and the database server. These drivers must be connected through some connection protocols (some of these protocols are Open DataBase Connectivity (ODBC) and Java DataBase Connectivity (JDBC)). Thus, a server side script must have classes for driver specification and connection establishment. In addition, server-side scripting support constructs to create and execute SQL commands through these connections and obtain the resulting data tables. But how to access individual data records and attributes? Thus, every server-side scripting must support tools to access data records or rows and attributes or columns.
- Since HTTP is a stateless protocol, therefore, you need to establish sessions or create cookies (These will be discussed in Unit 3 of this Block). Thus, server side scripting must have classes/mechanisms to deal with sessions and cookies.
- Server side script produces output for the browser, therefore, such classes or commands must exist in such languages.

Thus, server-side scripting requires extensive features on part of its tools. Some of the most popular server-side scripting languages include – ASP.NET, PHP, JSP and SERVLETS, Perl and many more. There are a number of tools that integrate complete web development environment including web servers. Some such IDEs are Eclipse, NetBeans, and Visual Studio Express etc. A detailed discussion on these tools is beyond the scope of this Unit. You will be learning more about JSP in this Block. However, you may learn and use other languages also as many of the concepts are of similar nature.

1.5 HTTP PRIMITIVES

In the previous sections, you have been explained about the client-server architectures and the features of the language required for web application development. In these sections, you have been told that data from the client is passed to the server using a protocol. In this section, we define some of the basic methods that make this communication possible.

HyperText Transfer Protocol (HTTP) is the protocol that is used for WWW. HTTP is an application level protocol. What is an application level protocol? Please find out answer to this question from BCS041: Fundamentals of Computer Networks. Present version that is in use is HTTP/1.1.

An interesting related work that has helped WWW immensely is the concept of MIME (Multipurpose Internet Mail Extensions). MIME is a standard that was designed by an Internet Engineering Task Force Working Group. It was designed so that you can include or attach multimedia contents such as pictures, audio, video etc. and non-ASCII texts in your emails. The standard defines the way to represent and encode such contents. MIME standard became very popular and now it is used to describe content type for the Internet also. In the context of HTTP it defines the format of data for transmission. Internet Media Types, also called MIME type, defines the content type that is being transmitted over the Internet using HTTP or some other protocols. Some of the examples of these Internet Media Types are:

File Type	Description
audio/mp4	The audio file containing audio data as per mp4 standard
image/png	Image file containing picture in Portable Network Graphics format.
text/css	Human readable text data containing style sheet content
...	...

There is long list of all such types. You can refer to further readings for more details.

1.5.1 Request and Response

HTTP uses a request-response method of communication. As a client, you type in the URL of a website, for example, <http://www.ignou.ac.in/student/results.jsp>, you are initiating a request for the web server. This process would involve TCP to set up a connection between the client and the host computer hosting www.ignou.ac.in at port 80 which is the default port for HTTP. But, how will it identify to which web server this request has been initiated, so that the connection can be established? First nearest Domain Name System (DNS) server to the client tries to translate the www.ignou.ac.in portion of the URL into the equivalent IP. If it fails it takes the help of other name servers to do so. Thus, if a web site exists, its IP address is returned. Please note that the IP address is of the web server hosting www.ignou.ac.in. However, please note that the URL also has path and name of the file which you wish to access. In this case the path is `/student/` and the file name that you wish to access is `results.jsp`. Please note that due to security issues path may be a virtual path. Please

refer to further readings for more details on path. In any case the web server must be told about the path and file (resource) that a client needs to access.

Now, the client computer knows the IP address of the web server, the path and the resource name. It initiates the request using the HTTP methods such as GET (this is explained in the next subsection). A request is initiated with the help of a message. This message contains a head and a body. The format of the message is defined in the RFC 2616. A detailed discussion on RFC 2616 is beyond the scope of this Unit. You can read more about this document from one of the websites of RFC 2616 - <http://www.hjp.at/doc/rfc/rfc2616.html>.

For example, for the requested IGNOU page, the request may look like the following for the host www.ignou.ac.in:

```
GET /student/results.jsp HTTP/1.1
```

The message means that the request method is GET, the resource is results.jsp and the path for that resource is student folder of the root directory. HTTP/1.1 informs the host about the version of protocol.

Please note that a request for a resource should specify the host name, path, resource name and the protocol version.

As far as response of the sent request is concerned the response will include several messages that are sent in a sequence:

First the status code of the failure or success of the request along with the protocol version is sent. For example, the response line that indicates success of the request is: HTTP/1.1 200 ok. There are a number of status codes (with small description). For more details on response code, you may refer to RFC 2616.

As a second step a number of *HTTP headers* are sent to the clients. The purpose of HTTP headers is to inform the client about the type of document being sent such as text, graphics, etc. For more details, please refer to the RFC 2616 document.

Finally the resources requested by the client are sent. Please note that even a single request may result in sending multiple resources to the client.

The process of request-response is shown in Figure 7.

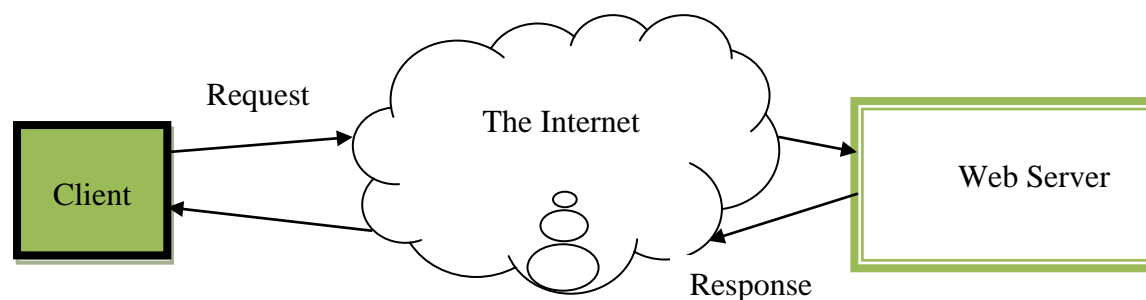


Figure 7: Request and response primitives

1.5.2 HTTP Methods

In the previous subsection, you read the term method in the context of request message. There are several methods defined in the RFC 2616, however, we will discuss the ones that you use most often for getting specific response from

the server. These methods are - GET, POST, HEAD, etc. But before we discuss, about the specific methods let us identify some basic characteristics of these methods.

Safe Methods: The methods that are only meant to retrieve information and not make any change in the server of any kind are considered as safe methods. GET and HEAD methods are safe methods.

GET method: The purpose of GET method is to retrieve the required sources from the server. GET can also be used to communicate information to the server as part of URL. For example, the in the practical lab manual of course BCSL057, we have used the GET method to access the list of the students of a course through a form. The following is the display of web page:

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/DatabaseApplication/CourseStudentList.jsp'. The page has a green header with the text 'Indira Gandhi National Open University'. Below the header, there is a purple sidebar with links: 'Home', 'Today's Date', 'Your Visitor No.', 'Enter Student Information', and 'Student List for a Course'. The main content area is white and titled 'Welcome to Lab Sessions'. It contains a 'Select Programme' dropdown menu with a list of programs: 'Bachelor of Computer Applications', 'Certificate in Information Technology', 'Master of Business Administration', and 'Master of Computer Applications'. A 'Submit' button is located below the dropdown menu.

Figure 8: Display of form that initiates a GET request

Please note that the form is stored in the webpage

<http://localhost:8080/DatabaseApplication/CourseStudentList.jsp>

Here, the localhost:8080 indicates that you are referring to the local web server on this machine at a port number 8080. The webpage uses the path /DatabaseApplication/ and access the resource CourseStudentList.jsp. This results in display as shown in Figure 8. This jsp file contains the HTML code line:

```
<form name="InputProgramme" action="CourseStudentListProcess.jsp" method="GET">
```

Thus, when you select Bachelor of Computer Application and press Submit button on the form displayed in Figure 8, the action will be taken by the resource file CourseStudentListProcess.jsp. The method that will be used will be GET method as stated in the HTMLcode (...method="GET">). This will result in display of the web page that displays the student list. From the point of view of our discussion, the actual display of the student list is not important but the address line which is:

<http://localhost:8080/DatabaseApplication/CourseStudentListProcess.jsp?ProgCode=BCA>

Please notice that after the resource name there is text *?ProgCode=BCA*. This text is passing additional information to the server that about a parameter called ProgCode having the value BCA for Bachelor of Computer Applications. Thus, GET not only passes the resource name but also several parameters that can be used by server to refine information. For example, in the case above server may use the jsp file and ProgCode information to find the information of BCA students only and send it back to the client.

HEAD method: HEAD method is like the GET method only, the only difference being that the only the headers are sent by the response and not the requested resources.

POST method: POST is another important method used for passing data to the server. However, in POST method data is passed inside a message and not as part of the URL being sent. Consider the following filled in HTML form from the URL <http://localhost:8080/DatabaseApplication/StudentInformationForm.jsp> and displayed in Figure 9.

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/DatabaseApplication/StudentInformationForm.jsp'. The page has a green header with the text 'Indira Gandhi National Open University'. On the left, there is a purple sidebar with links: 'Home', 'Today's Date', 'Your Visitor No.', 'Enter Student Information', and 'Student List for a Course'. The main content area is white and titled 'Welcome to Lab Sessions'. It contains a form with the following fields: 'Student ID' (text input with value '9'), 'Student Name' (text input with value 'Rajeev Saini'), 'Student Phone' (text input with value '9987887777'), and 'Select Programme' (dropdown menu with 'Master of Business Administration' selected). A 'Submit Information' button is located at the bottom of the form.

Figure 9: An HTML form for Student Information.

The form contains the following HTML code:

```
<form name="InputStudentData" action="StudentDataInput.jsp"
method="POST">
```

When you submit the form by pressing Submit Information Button, the information must be sent to the server. In this case parameters like student ID, name, phone, programme etc will be put inside the request message as the method used is POST. Thus, the URL of the page processing this information will be <http://localhost:8080/DatabaseApplication/StudentDataInput.jsp>

There are several other request Methods, however, a detailed discussion on those methods are beyond the scope of this Unit. You may refer about them in the further readings.

Mostly you would be using GET or POST methods while doing simple web programming. You must be wondering which of the two would be better? GET has an advantage that it is simple and safe, however, the maximum length of the string that you can pass through a URL is fixed. POST on the other hand has an advantage that it hides information as it is not displayed. Sometimes several hidden parameters are also passed along with basic parameters, in such case you may like to use POST method. In general, whenever you want that data should not be visible on URL you use POST method. GET method is very useful when you are testing your applications, as you clearly know the values of different parameters.

Check Your Progress 2

- 1) Differentiate between the client side and server side scripting languages. Name at least four server side scripting languages

.....

.....

- 2) What is a request and response in the context of HTTP? What are its methods in the context of a request?

.....

.....

.....

3) How is GET different to POST?

.....
.....
.....

1.6 WEB CONTAINERS

In the last section we have discussed about the examples of GET and POST methods of HTTP request. A HTTP request is for specific web resource form a web server, generally, the HTML page, which should be sent as part of the response to the client. Notice that in the example of GET and POST we have asked for a web resource which have .jsp as a file extension which is a JSP file and not HTML file. How, does this JSP file along with the parameters is converted to a desired dynamic response from the web server to the client? This is the responsibility of the web container part of the web server.

A web container is part of Java Enterprise Architecture. We will discuss only generic information about Web Container. You can refer to more details from WWW or any reference book on JSP and Servlets. A web container is part of a web server. It relates a client request to a servlet that is to be executed in order to give a response to a request. A Servlet is a Java Class that is executed on the server. Figure 10 shows the basic functioning of a web container.

Client	Web Server	
	HTTP Server	Web Container
Requests for a JSP Page	Accepts the parameters from the client and checks if the request is for a jsp file or servlet. If so pass the parameter and requested resource information to the Web Container	Accepts the request and checks if the latest version of JSP file is translated to the Servlet, if not then <ol style="list-style-type: none">1. Translates JSP file to servlet2. Uses Java Compiler to convert the Servlet to Java (.class) file. Once the .class file is available, it is executed using Java Runtime Environment at the Server. The output may be in the form of HTML. This is sent back to HTTP server.
The Requested Page is displayed.	Checks for error conditions, if everything is ok, HTML code is sent back to the client or else error message is sent.	

Figure 10: Role of Web Container

Some of the most common Web containers are Apache Tomcat, GlassFish, Eclipse Virgo, JBOSS, WebLogic, WebSphere and so on.

Check Your Progress 3

1) Explain the role of web container.

.....

.....

2) Name any three web containers

.....

1.7 SUMMARY

This Unit introduces you to the concepts of server side scripting. The unit first defines the server-side scripting and explains the basic terms used in this context. Client-server architecture specifically n-tier architectures were explained. MVC architecture is an important architecture for web application development. Model-View-Controller defines the three important aspects of server side programming. The model defines the data model, view defines the user view, and controller establishes the necessary interaction between the two. In order to perform web programming you need to use a language. This Unit list some of the basic features required in the server side scripting languages. You may find the trends in server side scripting technologies from the WWW or further readings. HTTP is the protocol that is used for WWW. The Unit also discusses the request-response primitive of HTTP. It explains GET and POST methods of client to server communication. Finally, the Unit defines the web container and explains its need.

1.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1:

1

<i>Static Web pages</i>	<i>Dynamic Web pages</i>
The web pages on server use HTML and CSS, they can include images, audio, video etc.	Web pages use a server side language and/or database. They can also include images, audio, video etc.
You cannot interact with static web pages	You can interact with these pages
Every user of the web site will see the same information	User may see the information of his/her choice
web sites were useful in showing brochure like fixed content	Web based applications such as Web 2.0 based applications
Changes or corrections in the content were made by the web administrator that too occasionally	Changes can be made in program, if needed.
Editing would require reload of content to web site.	Changed program may be reloaded
Designing and maintenance of web site was very tedious	Designing and maintenance and security of web site is better

- 2 Server side scripting is required for dynamic web page environment. This script can help you in designing flexible, maintainable and dynamic websites involving databases. The client-side scripting is used when an interaction can be replied to without any server interaction. Client side scripting is less secure than server side scripting. Client side scripting cannot connect to a server database
- 3 Three-tier web software architecture involves Presentation layer which is displayed on the client computer, Business Logic layer which controls access of data which is stored in the Data layer. This architecture helps in making scalable, secure, flexible web applications.
- 4 In MVC architecture the view can directly be updated by the Model, however, in the 3-tier architecture, the Data layer never interacts directly with the Presentation Layer. The information is passed through the Application Logic layer.

Check Your Progress 2:

- 1 The client side scripting languages are used for implementing functions that can provide basic interactivity at client side. These functions do not need any support of the server. For example, checking whether information in a form has been filled up, format of data entered is correct, while server side scripting is used to provide features from the server, for example, client side scripting cannot check the availability of a user name, it has to be checked using server side scripting. Server side scripting is safer and very useful when dynamic web pages are to be designed using database. Client-side scripting is browser dependent whereas server scripts are run at web server and are generally independent of the browser. Server side scripting is very useful in making maintainable websites. Server side scripting languages include PHP, ASP.NET, Ruby on Rails, Java Server Pages etc.
- 2 Request and response are messages send in HTTP protocol. Request primarily is used by client to ask for resource specified using the URL. Response is given by the web server on receiving a request. Response may include either the resource requested or the error messages. The methods are used as part of request to include additional information or parameters for the server.
- 3 GET passes the parameters by appending them with the URL, whereas, POST puts them in the message body. POST is preferred over GET, when you want that information should not be seen along URL. GET is preferred when testing an application.

Check Your Progress 3:

- 1 A web container is used for translation, if needed, of the server side script like JSP into a servlet. In addition, the web container also decides which servlets are to be invoked in order to respond to user request passed to it.
- 2 Some of the web containers are Apache Tomcat, GlassFish, Eclipse Virgo, JBOSS, WebLogic, WebSphere etc.

1.9 FURTHER READINGS

<http://www.wikipedia.org>
<http://www.w3Schools.com>
<http://www.oracle.com>

<http://www.microsoft.com>

<http://www.eclipse.org>

<http://www.java2s.com>

"MIME: Multimedia Internet Mail Extensions" written by "Nathaniel S. Borenstein"

UNIT 2 JSP - Basics

Structure

- 2.0 Introduction
- 2.1 Objective
- 2.2 JSP: An Introduction
- 2.3 JSP Life Cycle
- 2.4 Elements of JSP
 - 2.4.1 Directives
 - 2.4.2 Scripting
 - 2.4.3 Action Elements
- 2.5 Comments and Template Data
- 2.6 JSP Implicit Object
 - 2.6.1 request
 - 2.6.2 response
 - 2.6.3 session
 - 2.6.4 application
 - 2.6.5 page
 - 2.6.6 pageContext
 - 2.6.7 out
 - 2.6.8 config
 - 2.6.9 exception
- 2.7 Complete Example
- 2.8 Summary
- 2.9 Solutions/Answers
- 2.10 Further Readings

2.0 Introduction

In the previous block, you have learned that how to create HTML web pages. The simple HTML pages are static pages. Java Server Pages are simple but powerful technology used to generate dynamic web pages. Dynamic web pages are different from static web pages in that web server will create a web page when it is requested by a client or user. For example, your online results on IGNOU website, the page for every student instead IGNOU web server dynamically creates a page depending on your enrolment number.

JSP was released in 1999 by SUN Microsystems. JSP is a technology for developing web pages. JSP is similar to PHP and ASP, but it uses the Java programming language. It follows the characteristics of Java 'write once and run anywhere'. JSP enables you to add dynamically generated content with static html. In addition to html, JSP page are built using different components viz., directives, scripting elements, standard actions and implicit objects. This unit covers how to create a JSP page. It also provides a basic understanding of Java Bean, custom tag and life cycle of Java Server Page.

2.1 Objectives

After going through this Unit, you will be able to

- use JSP to create simple dynamic web pages
- define the JSP page life cycle
- use directives, scripting tags and JSP action elements in a JSP page
- access Java Bean within JSP page
- create a custom tag
- forward request from JSP page to other resource
- use JSP implicit objects


2.2 JSP: An Introduction

Java Server Pages (JSP) is a web technology that helps software developers to create dynamic content based web pages. Unlike a plain HTML page, which contains static content that always remains the same but in JSP; you can change content dynamically with the help of Java Bean and JSP elements. JSP is an extension of Java Servlet because it provides more functionality than servlet.

Servlet are server side components that services requests from a web server. It is basically a Java class that runs on a server. Servlet technology is used to create web application. It uses the Java language. It is runs inside the Java Virtual Machine. Two packages such as javax.servlet and javax.servlet.http are required for writing a servlet. These two packages make up the servlet architecture. The javax.servlet package contains the classes and interfaces that are implemented and extended by all the servlets. Other package javax.servlet.http contains classes that are required when creating a HTTP specific servlet. All servlets must implement the Servlet interface, which defines life-cycle methods. The Servlet interface defines the three most important methods such as init(), service() and destroy() method. Servlet also possesses all the Java features such as platform independence, high portability, security and Java database connectivity.

Servlet is very useful for writing server side code but it suffer from some disadvantages. In particular, writing HTML code with plenty of out.println() statements (println() is a method of system.out object to display string which is passed to it), it is very tedious and error prone and also software developers has to take on dual roles of developing application logic and designing web pages. JSP is designed to address these disadvantages.

A Java Server Page contains HTML tags as well as JSP elements. The JSP elements are basic building blocks of the page. The JSP elements are easier to maintain than the servlet. A JSP page contains a very simple structure that makes it easy for developers to write JSP code and also easy for servlet engine to translate the page into a corresponding servlet. In addition to html tags, a JSP page consists of directives, scripting elements, scriptlets and action elements. Each of these elements can use either JSP syntax or they can be expressed in XML syntax but you cannot intermix the two. For this problem, you can use the include mechanism to insert file that may use different syntax.



Web Servers are computers that using client/server model and connected to internet for serving web pages.

The Java Server Pages has more advantages over the servlet which are as follows:

- It allows programmers to insert the Java code directly into the JSP file that makes the development process easier.
- JSP support element based dynamic content that allows programmers to develop custom tags libraries to satisfy application needs.
- Content and display logic are separated
- JSP pages can be used in conjunction with servlet that handle business logic.

The file extension for the source file of a JSP page is .jsp. The following code contains a simple example of a JSP page:

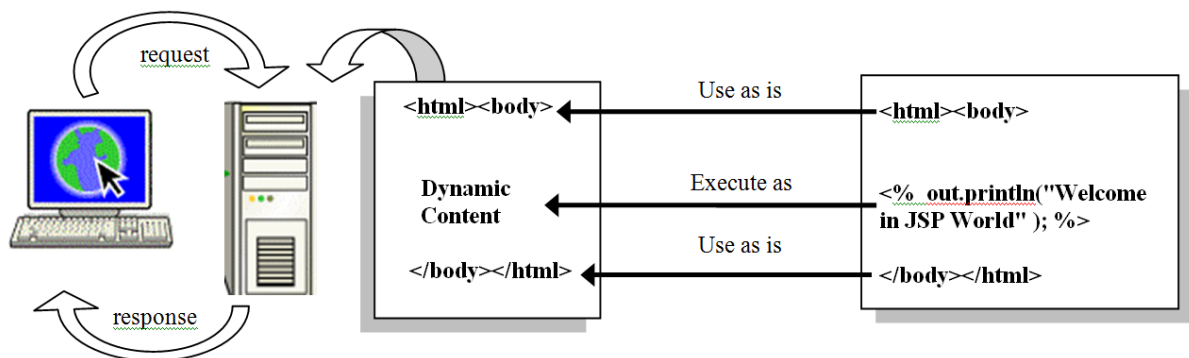


Figure 1: Generating dynamic content with JSP page.

The output of the above program is as follows:

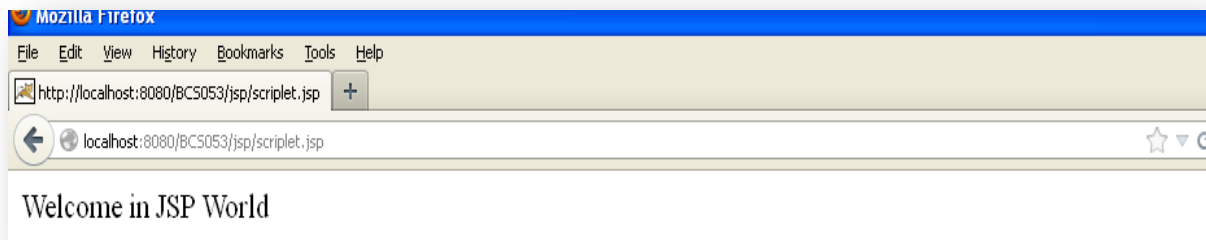


Figure 2: JSP Page

You can see the above program which looks like any other HTML page with some added JSP elements that allow the server to insert dynamic content in the page. When client send a request for a JSP page, the server executes a JSP page elements merges with static contents and sends the dynamically page back to the client browser, as illustrated in Figure-1.

The JSP technology is based on JSP API (Application Programming Interface) that consists of two packages i.e. javax.servlet.jsp and javax.servlet.jsp.tagext packages. In addition to these two packages, JSP also needs two packages of servlet such as javax.servlet and javax.servlet.http. Apart from these interfaces and classes, the two exception classes: JspException and JspError are also defined in JSP API. The javax.servlet.jsp package has two interfaces such as HttpJspPage and JspPage and four classes: JspEngineInfo, JspFactory, JspWriter and PageContext. The jspInit() and jspDestroy() methods are defined in

JspPage interface and `_jspService()` method is in `HttpJspPage` interface. The `javax.servlet.jsp.tagext` contains classes and interfaces for the definition of Java Server Pages Tag Libraries.

2.3 JSP Life Cycle

In this section, you will go through the life cycle of JSP and see how a JSP page is displayed. When the JSP is first accessed, it is translated into corresponding servlet (i.e. java class) and compiled, then JSP page services request as a servlet. The translation of JSP page is done by the JSP engine of the underlying web container/servlet container (e.g. Tomcat). Figure-3 shows that how a JSP page is processed.

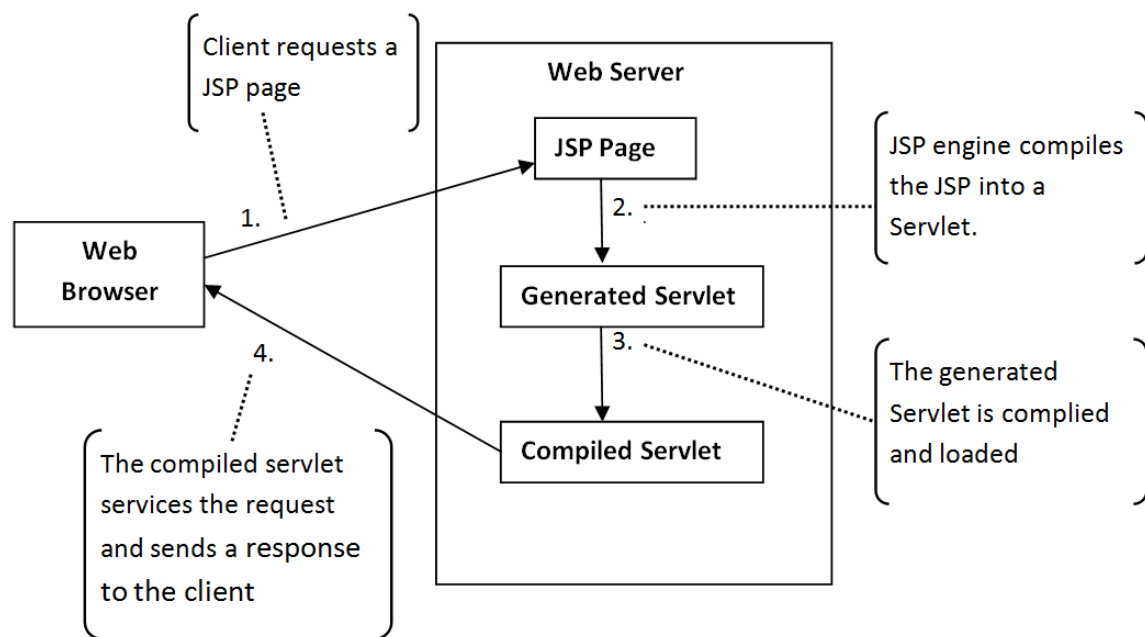


Figure 3: The steps of a JSP Page processing.

The life cycle of JSP page is controlled by three methods i.e. `jspInit()`, `_jspService()` and `jspDestroy()`.

jspInit() - The `jspInit()` method is called only once during life cycle of a JSP, similarly, servlet also have an `init()` method whose purpose is same as that of `jspInit()`. `jspInit()` method is used to initialize objects and variables that are used throughout the life cycle of JSP. This method is defined in `JspPage` interface. This method is invoked when the JSP page is initialized. It has no parameters, return no value and thrown no exceptions. The signature of the method is as follows:

```
public void jspInit() { // Initialization code }
```

_jspService() - `_jspService()` is the method which is called every time the JSP page is requested to serve a request. This method is defined in the `javax.servlet.jsp.HttpJspPage`

interface. This method takes `HttpServletRequest` and `HttpServletResponse` objects as arguments. The `_jspService()` method corresponds to the body of the JSP page. It is defined automatically by the processor and should never be redefined by you. It returns no value. The underscore ('_') signifies that you cannot override this method. The signature of the method is as follows:

```
public void _jspService(  
    javax.servlet.http.HttpServletRequest request,  
    javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException, java.io.IOException  
{  
    // services handling code  
}
```

jspDestroy()- The `jspDestroy()` is invoked when the JSP page is to be terminated. It is synonymous with the `destroy()` method of a servlet. It has no parameters, return no value and thrown no exceptions. Override `jspDestroy()` when you need to perform any cleanup, such as releasing database connections or closing open files. The signature of the method is as follows:

```
public void jspDestroy(){ // cleanup code }
```

The following Figure-4 shows the life cycle of JSP page.

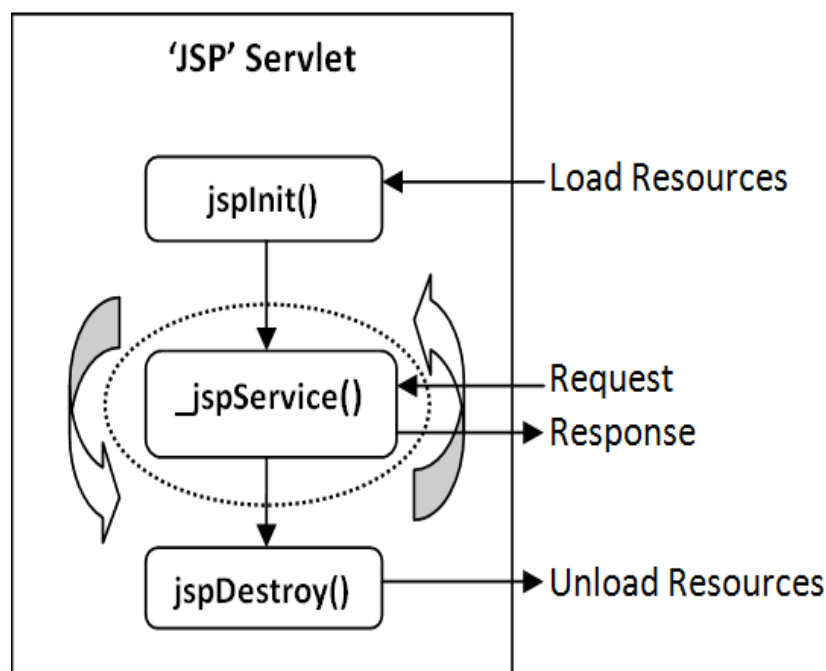


Figure 4: Life Cycle of JSP Page

Check Your Progress 1

1. Explain the term JSP.

2. What are the advantages of JSP over Servlet?

3. Describe the `_jspService()` method.

4. Write the basic steps for processing JSP request.

5. Explain the life cycle of JSP.

2.4 Elements of JSP

In this section, we are going to introduce the elements of JSP that make up a JSP page. There are three types of JSP components such as directives, expressions and scriptlets. A directive affects the overall structure of the JSP page. They are discussed in detail in the following sections:

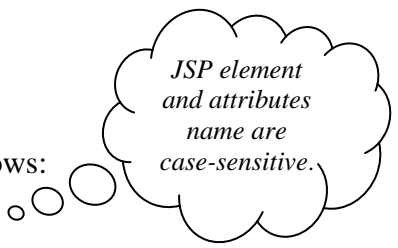
2.4.1 Directives

Directives are used as guiding the JSP container for translating and compilation of JSP page. It appears at the top of the page. Using directives, the container translate a JSP page into corresponding servlet. They do not directly produce any output. A directive contains one or more attribute name/value pairs. Directives are defined by using `<%@` and `%>` tags. Directives have the following syntax:

`<%@ directive attribute = "value" %>`

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

`<jsp:directive. directivename attribute = "value" />`



*JSP element
and attributes
name are
case-sensitive.*

There are three types of directives currently used in JSP document: *page*, *include* and *taglib*. Each one of these directives and their attributes are defined in following sections:

page Directive

The page Directive is used to specify the attributes of JSP page such as declaration of other resources i.e. classes and packages, page buffering requirements and name of page that should be used to report run time errors, if any. The page Directive is a JSP element that

provides global information about an entire JSP page. This information will directly affect the compilation of the JSP document. Following is the syntax of page directive:

```
<%@ page attribute = "value" %>
```

The page directive contains many attributes, you can set these attributes. The attribute may be one or more of the following:

Attribute	Description
language="scripting language"	This attribute define the language that will be used to compile the JSP document. By default, it is java language.
import="import list"	This attribute defines the names of packages.
session="true false"	It specifies whether or not the JSP document participate in HTTP session. The default is <i>true</i> .
extends="classname"	This attribute define the name of parent class that will be inherited by generated servlet. It is rarely used.
buffer="none size in kb"	The default value is 8kb. It specifies the size of <i>out</i> buffer.
autoFlush="true false"	The default is <i>true</i> . It means that the <i>out</i> buffer will be automatically flushed when full. If it is <i>false</i> , it will be raised an exception when buffer is full.
Info="text"	If this attribute is used, the servlets will override the <code>getServletInfo()</code> method.
errorPage="error_page_url"	This attribute defined the relative URL to JSP document that will handle exception.
isErrorPage="true false"	This attribute indicates whether the current page can act as an error page for another JSP page. The default is <i>false</i> .
isThreadSafe="true false"	The default value is <i>true</i> . It indicates that the page can service more than request at a time. When it is <i>false</i> , the <code>SingleThreadModel</code> is used.

An example of the use of page directive is as follows:

```
<%@ page import="java.io.*, java.util.Date" buffer="16k" autoFlush="false" %>
```

This page directive instructs the web container to import `java.io` package and `java.util.Date` class. It also instructs the web container to set buffer size to 16k and turn off autoflushing.

You can specify multiple page directives in your JSP document, such as the following:

```
<%@ page import="java.util.Date" info="Example Page" %>
<%@ page errorPage="MyErrorPage.jsp" buffer="16kb"%>
```

In the above examples of page directives, the first directive statement tells the web container to import java.util.Date class and to set information about the JSP page which is retrieved by getServletInfo() method. The value of the info attribute will be a text string. The second page directive statement define a name of error page which is used for handling errors and set the buffer size in 16 kb.

include Directive

Include directive is an important JSP directive. This is used to insert text and code in the form of file such as html, JSP into a current JSP document at the translation time. It means that it enables you to import the content of another static file into a current JSP page. This directive can appear anywhere in a JSP document. The syntax of include directive is as follows:

```
<%@ include file = "relative URL" %>
```

***Relative URL** only specifies the filename or resource name; while **Absolute URL** specifies the protocol, host, path, and name of the resource name.*

For example: The following example will demonstrate the physical inclusion of header file. In this way, you can create a single header file only once and use it many times for your website.

Source code for main.html:

```
<html><body>
<h2>Example of include directive</h2>
<%@ include file='header.html' %>
</body></html>
```

Source code for header.html:

```
<html><body>
<h2>This is from Header File</h2>
</body></html>
```

The output of the above program is as follows:

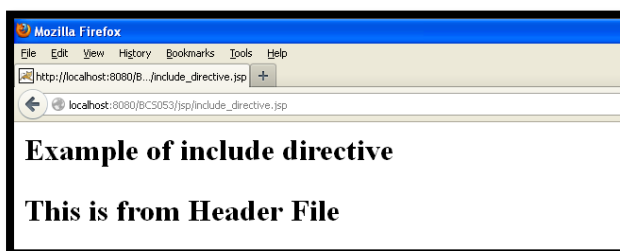


Figure 5: Example of include directive

Uniform Resource Identifier (URI) identifies a resource either by location or name. For e.g.
<http://www.ignou.ac.in/ignou/aboutignou>

taglib Directive

As yet, you have learned the basic elements of JSP. In this section, you will learn about the creation of custom tag libraries in Java Server Pages.

A custom tag is user defined tag. It is a reusable code in a JSP page and tag library is a collection of custom tags.

Custom Tag Syntax -

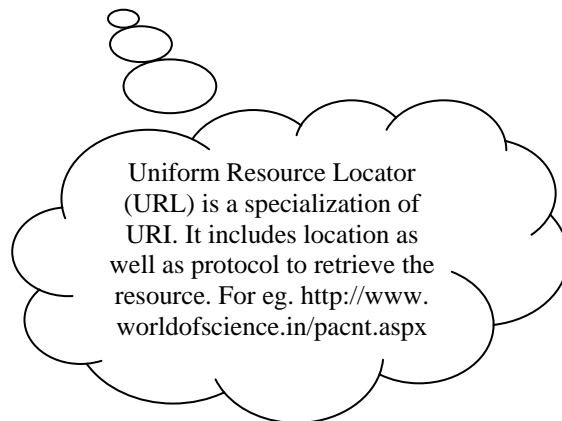
To use the customs tags in a JSP page, JSP technology provide a taglib directive to make use of the tag. The taglib directive has the following syntax:

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

The uri attribute defines an absolute or relative uri of tag library descriptor (TLD) file and the prefix attribute defines the string that will identify a custom tag instance.

To use customs tags in a JSP page, you need to know four components that make use of the custom tags. These are:

- 1) Tag handler class
- 2) Tag library descriptor file
- 3) JSP file
- 4) Deployment descriptor file (web.xml)



Tag Handler Class -

It is a java class that defines the behaviour of the tags. This class must implement the javax.servlet.jsp.tagext package.

Tag Library Descriptor (TLD) file -

A tag library descriptor file is an xml document. It defines a tag library and its tags. The file extension of this file is .tld. It contains one <taglib> root element and <tlibversion>, <jspversion>, <shortname> tag are sub elements of the taglib element. The tag is the most important element in TLD file because it specifies the name of the tag and class name of the tag. You can define more than one tag element in the same TLD file.

Deployment Descriptor file (web.xml) -

The deployment descriptor is an xml file that specifies the configuration details of the tag. The most important element for custom tag in web.xml file is <taglib-location>. Using the web.xml, the JSP container can find the name and location of the TLD file.

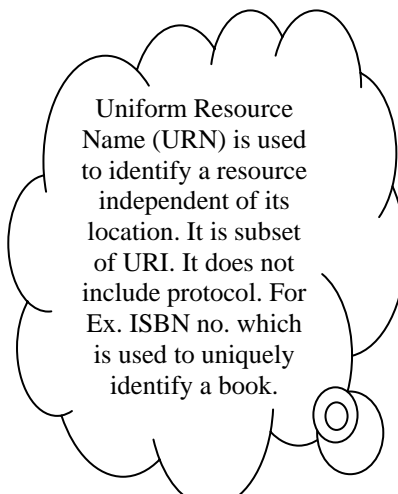
A JSP file -

Once, you have created tag handler java class, a tag descriptor file and define configuration details in deployment descriptor file and then you have to write a JSP file that makes use of the custom tag.

Here are some steps for generating a simple custom tag. Now, follow the following five easy steps:

Step-1: write and compile a java class called MyCustomTag.java which is given in following program source. The class file must be placed in the directory say 'customTag' under the WEB-INF/classes directory.

```
package customTag;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class MyCustomTag extends TagSupport {
public int doEndTag() throws JspException {
JspWriter out = pageContext.getOut();
try { out.println("Hello from custom tag"); }
catch(Exception e) {}
return super.doEndTag();
} //doEndTag()
} //main class
```



Uniform Resource Name (URN) is used to identify a resource independent of its location. It is subset of URI. It does not include protocol. For Ex. ISBN no. which is used to uniquely identify a book.

Step-2: Create a TLD file named taglib.tld as shown in following program source and save it in WEB-INF directory.

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN" "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
<taglib><tlib-version>1.0</tlib-version><jsp-version>1.2</jsp-version>
<short-name></short-name> <tag> <name>myTag</name>
<tagclass>customTag.MyCustomTag</tagclass></tag> </taglib>
```

Step-3: Create a JSP file named CustomTag.jsp that contains the following code:

```
<html><body><% @ taglib uri="/myTLD" prefix="easy" %>
<easy:myTag /></body></html>
```

Step-4: Place the following code in web.xml under the <web-app> root element.

```
<taglib> <taglib-uri> /myTLD </taglib-uri>
<taglib-location>/WEB-INF/taglib.tld </taglib-location>
</taglib>
```

Step-5: Start server say Tomcat (if you are using this). Open web browser and run the JSP page. The following screen comes as an output for a simple custom tag.

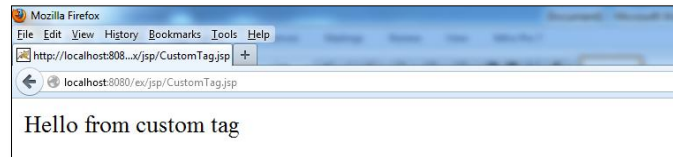


Figure 6: A simple JSP Custom Tag page

When the user requests the JSP page, the JSP container first sees the taglib directive and gets the taglib uri. On the basis of this, JSP container looks into the web.xml file to find taglib location and continues the processing and get the name of tag. After getting the name and location of TLD file, it obtains the java class. Now, the JSP container loads the class for custom tag and continues the processing.

2.4.2 Scripting Elements

Scripting elements allow you to insert java code fragments directly into an HTML page. You can specify three ways to include java code into your JSP document, such as declarations, expression and scriptlets. Each of these scripting elements has an appropriate location in the generated servlet. These are discussed in more detail in following sections:

Declarations

As its name implies, declarations are used to declare the variables and methods that can be used in the JSP document. The declaration part is initialized when the JSP document is initialized. After the initialization, they are available to other expressions, declaration and scriptlets. A declaration is start with a `<%!` and end with a `%>`. The syntax of the declaration is as follows:

```
<%! declaration %>
```

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

```
<jsp:declaration> code fragment </jsp:declaration>
```

For example: simple declaration of variables

```
<%! int i = 0; %>    // i is an integer type variable
<%! int a, b, c; %>  // a, b, c is an integer type variable
<%! Square a = new Square (4.0); %>
```

Following is a variable and method declaration:

```
<%! String name = new String("SOCIS"); %>
<%! public String getName() { return name; } %>
```

When the above code is automatically compiled and converted from JSP code to servlet, the declaration part of JSP page is included in the declaration section of the generated servlet. The first declaration statement is used to defined a variable as string type with initial value 'SOCIS' and second statement is defined a string type method named 'getName()'.

Expressions

An Expression is an instruction to the web container to execute the code within the expression and to display the resulting data at the expression's referenced position in the JSP document. Expressions are evaluated at request time i.e. run time. The syntax of the expression is as follows:

```
<%= expression %>
```

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

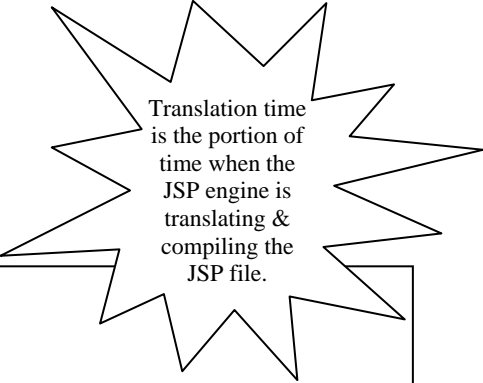
```
<jsp: expression > code fragment </jsp: expression >
```

For example, to show the current date and time:

```
<%= new java.util.Date() %>
```

Consider another example:

```
<html><body>
<%! String name = new String("SOCIS"); %>
<%! public String getName() { return name; } %>
Hello <b><%= getName()%></b><br><%= new java.util.Date() %>
</body></html>
```



Translation time is the portion of time when the JSP engine is translating & compiling the JSP file.

The output of the above program is as follows:

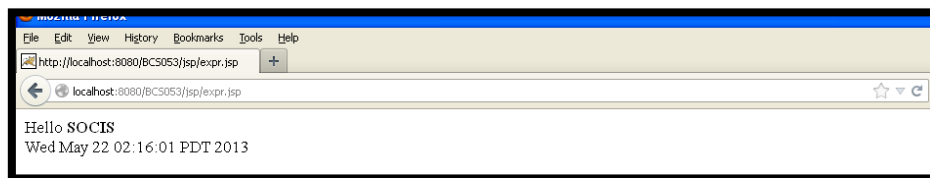


Figure 7: Expression example

When the above code is compiled and converted to servlet code and then expression part is placed in its referenced position of the generated servlets' `_jspService()` method.

Scriptlets

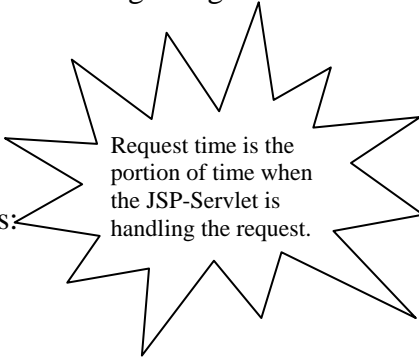
In scriptlets, all the scripting elements are brought together. It is executed at the request time and makes use of declaration, expressions and JavaBeans. You can write scriptlets anywhere in a page. It contains a valid java statements within the `<%` and `%>` tag and get inserted into the `_jspService()` method of generated servlet.

The syntax of the scriptlet is as follows:

```
<% scriptlet code %>
```

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

```
<jsp:scriptlet > code fragment </jsp:scriptlet >
```



Request time is the portion of time when the JSP-Servlet is handling the request.

For example: The following example will demonstrate the number series from 1 to 3.

```
<html><body><p>Counting to 1- 3 :</p>
<% for (int i=1; i<=3; i++) { %><p>This number is <%= i %>.</p>
<% } %></body></html>
```

Output of the above program is as follows:

```
Counting to 1- 3 :
This number is 1.
This number is 2.
This number is 3.
```

Figure 8: scriptlets Example

2.4.3 Actions Elements

Action elements (or standard actions) are tags that can be embedded in a JSP document. At the compile time, they are replaced by java code.

Before going to start learning about how you can add Java Bean in JSP page, you must take a look at what a bean is. A Java Bean is nothing more than a java class. It is reusable component that work on any Java Virtual Machine. For the creation of Java Bean, you must create a java class that implements `java.io.Serializable` interface and uses public `get/set` methods to show its properties.

<jsp:useBean>

The first JSP standard action (identified by *jsp* prefix) is `<jsp:useBean>`. This action is used to include an instance of java bean within java server pages. The syntax of the `<jsp:useBean>` action is as follows:

```
<jsp:useBean id="name"
             scope="page | request | session | application"
             typeSpecification>
    body
</jsp:useBean>
```

typeSpecification can be represented in the following syntax:

```
typeSpecification ::= class="className" |
                     class="className" type="typeName" |
                     type="typeName" beanName="bean_name" |
                     type="typeName"
```

Following table contains the attributes of the `<jsp:useBean>` action:

Attribute	Description
<i>id</i>	It represents name of the object. This attribute is required.
<i>scope</i>	It defined the scope of the object. It may be <i>page</i> , <i>request</i> , <i>session</i> or <i>application</i> . This attribute is optional. By default, it is <i>page</i> .
<i>Class</i>	This attribute represents the fully qualified class name of the object. The class name is case sensitive.
<i>type</i>	It specifies the type of object. The value of this attribute is equal to <i>class</i> , a super class of <i>class</i> or an interface implemented by the <i>class</i> . If it is not specified then same as class attribute.
<i>beanName</i>	It is the name of bean.

The **scope** attribute of Java Bean means how long the object is available and if it is available than only for a single user or all application users JSP provides different scope for sharing data between web pages. These are:

- **Page** - ‘page’ scope means, the JSP object can be accessed only from within the same page where it is created. By default, it is page. JSP implicit objects out, exception, response, pageContext, config and page have ‘page’ scope.
- **Request** – The object is available to current JSP and to any JSP or Servlet that control is forward to and included from. Only Implicit object request has the ‘request’ scope.
- **Session** – JSP object is accessible from any JSP within the same session. Implicit object session has the ‘session’ scope.
- **Application** - JSP object is accessible from any JSP within the same web application. Implicit object application has the ‘application’ scope.

<jsp:setProperty>

This action is used to sets the Java Beans property value. The syntax for this action is as follows:

```
<jsp:setProperty name="beanName" property_expression />
```

property_expression can be represented in the following syntax:

```
property="*" |
property="propertyName" |
property="propertyName" param="parameterName" |
property="propertyName" value="propertyValue"
```

Following table contains the attributes of the <jsp: setProperty > action:

Attribute	Description
<i>name</i>	It represents name of the bean instance defined in <jsp:useBean> action.
<i>property</i>	It specifies the name of property being set. This attribute is required. If it is an asterisk (*) then it is used to set all properties of bean. You can also specify the specific bean property.
<i>param</i>	This attribute represents the name of parameter.
<i>value</i>	It specifies the value assigned to the named bean’s property.

<jsp:getProperty>

Once you have defined a bean and given it a name using the <jsp:useBean> action. Now, you can get the beans property values with another standard action <jsp:getProperty>. This action is used to get the property value from a Java Beans and add to the response body. The syntax for this action is as follows:

```
<jsp:getProperty name="beanName" property="propertyName" />
```

Following table contains the attributes of the <jsp: getProperty > action:

Attribute	Description
name	It represents name of the bean instance defined in <jsp:useBean> action.
property	This attributes represents the specific property within bean.

Following is the simple Java Bean example that store student name.

Let's create a Java Bean named student.java and place class file under WEB-INF/classes/bean1 directory.

```
package bean1;  
public class student {  
    private String name;  
    public String getName() { return name;}  
    public void setName(String name) { this.name=name;}}
```

Now, you can access the properties of the Java Bean from a JSP.

```
<html><body>  
<jsp:useBean id="myBean" class="bean1.student" scope="session" />  
<jsp:setProperty name="myBean" property="name" value="Poonam" />  
Welcome to this Unit, <jsp:getProperty name="myBean" property="name" />  
, Please visit egyankosh.ignou.ac.in to get more.  
</body></html>
```

The following output screen is as follows for the above program.

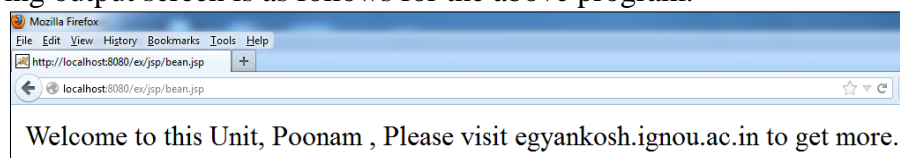


Figure 9: A simple Java Bean Example

You have learned the standard actions specific to a Java Bean. The remaining actions are defined in the following sections. Using the following standard actions, you can dynamically insert a resource into a current JSP page, forward the current request to another page or add bean/applet in a page.

<jsp:param>

The <jsp:param> action tag defines the name/value pair of parameter to be passed to an included or forwarded JSP document or a page that uses the <jsp:plugin> tag.

Following is the syntax of the above action tag:

```
<jsp:param name="paramName" value="paramValue" />
```

In the above syntax, name attribute defines the name of parameter being referenced and value represents the value of named parameter.

<jsp:include>

JSP supports two types of include: static and dynamic. In the previous section, you have already been studied the static include i.e. include directive. In static include, the content of included JSP code is inserted into the including JSP (or current JSP Document) at the translation time. After the content of included JSP is processed, the content included in JSP file does not change until the included file is changed and server is restarted. While in dynamic include, the content is included at the request time. This means that it is a mechanism for including static as well as dynamic content in the current JSP document such as static HTML, JSP or Servlet. This is done by <jsp:include> action element.

Dynamic includes are performed using the following syntax:

```
<jsp:include page="relativeURLSpecification" flush="true" />  
or  
<jsp:include page="relativeURLSpecification" flush="true" >  
    <jsp:param ...../>  
</jsp:include>
```

In the above syntax, page attribute defines the path of the resource to be included and flush attribute indicates whether the buffer is flushed. It is an optional parameter. The first syntax is used when <jsp:include> does not have a parameter name/value pair. If you want to pass the parameter to the included resource, use the second syntax.

For example:

```
<html><head><title>Student Information</title></head><body>  
<h2>JSP Include Example</h2><table><tr><td>  
<jsp:include page="head.jsp" flush="true">  
<jsp:param name="student" value="Poonam"/>  
<jsp:param name="prg" value="BCA"/>  
</jsp:include>  
</td></tr><tr><td>Course Name:BCS-053</td></tr></table>  
</body></html>
```

Source code for **head.jsp** file:

```
<% // Get the Student Name from the request
    out.println("<b>Student Name:</b>" + request.getParameter("student"));
    out.println("<br>");
    // Get the Course Name from the request
    out.println("<b>Programme Name:</b>" + request.getParameter("prg")); %>
```

Output screen for the above program:

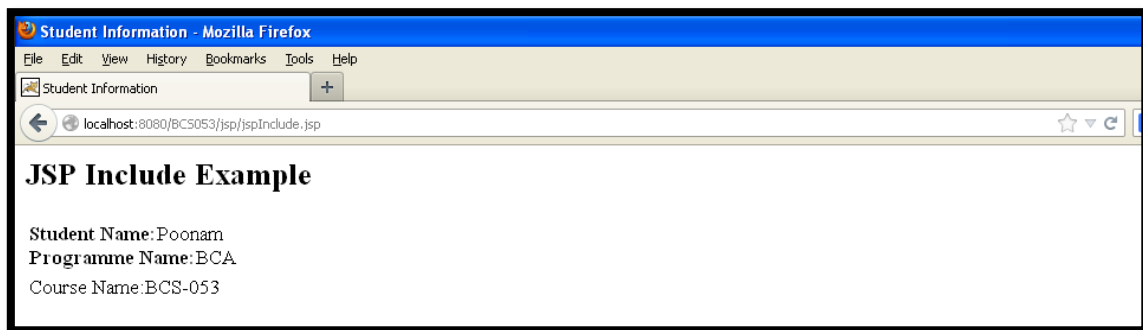


Figure 10: JSP Include Example

<jsp:forward>

The <jsp:forward> action is used to terminates the current execution of JSP page and transfer the control to the another JSP page within the same application. It may be static or dynamic page or resource. It can be used with <jsp:param> tag. The <jsp:param> is used for providing the values for parameters in the request to be used for forwarding.

The syntax for <jsp:forward> action tag is as follows:

```
<jsp: forward page="relativeurlSpecification" flush="true" />
or
<jsp: forward page="relativeurlSpecification" flush="true" >
    <jsp:param ...../>
</jsp: forward >
```

For example:

```
<html><head><title>Example: JSP Forward</title></head><body>

<% if((request.getParameter("city")).equals("delhi")) { %>
    <jsp:forward page="head.jsp" >
    <jsp:param name="student" value="Poonam"/>
    <jsp:param name="prg" value="BCA"/></jsp:forward>

<% } else { out.println("Your value is incorrect"); } %>

</body></html>
```

Source code for head.jsp file is same as in the above <jsp:include> example. Enter the following URL into your browser to see the result.

http://localhost:8080/BCS053/jsp/jspForward.jsp?city=delhi

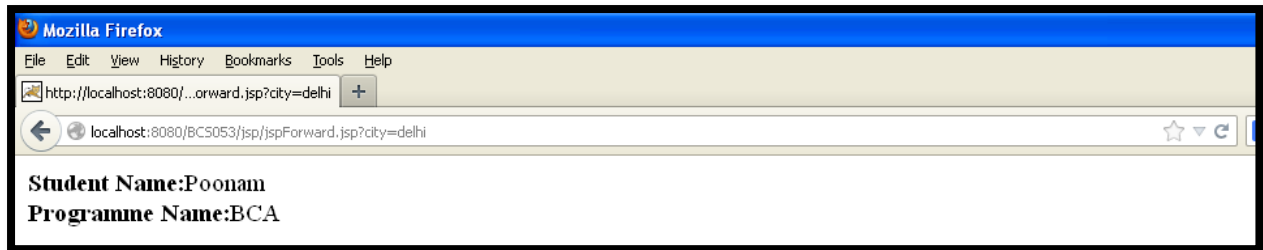


Figure 11: JSP forward example

<jsp:plugin>

The <jsp:plugin> action element is used to insert java component such as Applets and JavaBean in JSP page. The <jsp:param> is also used with action element to send parameters to Applet or Bean. Following is the syntax of <jsp:plugin> action:

```
<jsp:plugin type="pluginType" code="classFile" codebase="relativeURLpath">
<jsp:param ...../>
</jsp: plugin >
```

In the above syntax, type attribute indicates the type of plugin to include in JSP page, code attribute represent the name of class file and codebase attribute is the path of where the code attribute can be found.

For example: An Applet program named JavaApplet.java

```
import java.applet.*;
import java.awt.*;

public class JavaApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("Welcome in Java Applet.",40,20);
    }
}
```

Plugin source code:

```
<html><body><p>Plugin Example :</p>
<jsp:plugin type="applet" code="JavaApplet.class" width = "200" height = "200">
    <jsp:fallback>
        <p>Unable to load applet</p>
    </jsp:fallback>
</jsp:plugin>
</body></html>
```

The output of the above plugin program is as follows:

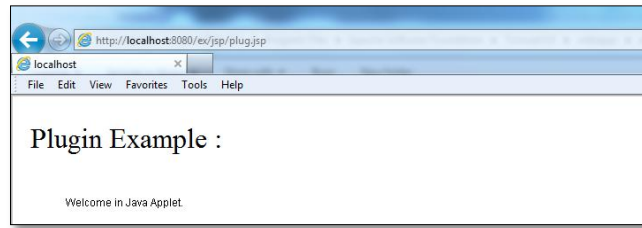


Figure 12:plugin example

<jsp:fallback>

The <jsp:fallback> action is used only with the <jsp:plugin> action element. It provides alternative text to the browser.

For example: <jsp:fallback> Unable to see plugin </jsp:fallback>

Check Your Progress 2

1. Explain the directive element of JSP with their syntax.

2. What are Standard actions in JSP? Explain the standard action specific to Java Bean.

3. What is difference between JSP include and include directive?

4. What is declaration scripting element? Explain with example.

5. Write a program which will demonstrate the use of <jsp:include> with <jsp:param> action.

2.5 Comments and Template Data

In this section, you will learn about the comments which is very necessary for developers to understand the flow of program for further reference. It is explanatory information about the function of code. Comments are very useful for any other developers also to maintaining or enhancing the code. You will also learn how to use template data within your JSP page.

JSP Comments

Commenting is a part of good programming practice. Comments help in understanding what is actually code doing. You can write comments in JSP like the following way:

JSP comment syntax:

<%--jsp comment text --%>

This comments tag is used within a Java Server Page for documentation purpose. It tells the JSP container to ignore the comment part from compilation, so it does not appear in the resulting java code.

`<!-- HTML comment -->` This HTML comment is treated by JSP container as just like another html tag. It is not a JSP comment. It can be viewed through the web browser's "view source" option.

`/* java comment */` or `//` used inside the scriptlets tag is also not a JSP comment. This is a java comment. These comments are not translated by JSP container and therefore do not appear on the client side web document.

For example:

```
<html><body>
<%-- This is JSP comment – not visible in the page source --%>
<!-- This HTML Comment - visible only in the page source
<% out.println("Welcome in JSP World" ); %> -->
<% //This is single line comment under the scriptlet %>
<% /* out.println("This comment is used for multiple lines."); */ %>
```

Program output:

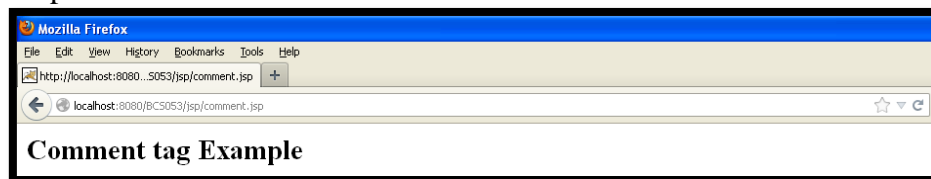


Figure 13: JSP comment example

Program output is visible only in page source code of web browser.



Figure 14: HTML comment's view in web browser

JSP XML Text Element

The `<jsp:text>` is an XML tag. It is used to enclose template data in JSP document. Template(static) data consists of any text that is not interpreted by the JSP translator. You cannot intermix JSP data with XML data in single file even you can include directive. The XML syntax for text element is as follows:

```
<jsp:text> text </jsp:text>
```

2.6 JSP Implicit Object

JSP Implicit objects are the java objects that are available for the use in JSP document to developers and they can call them directly without being declared first. These objects are also called predefined variables. These variables are parsed by JSP engine and inserted into generated servlet as if you defined them yourself. These objects are used within the scriptlets and expressions of the JSP page. These implicit objects are defined in the servlet specification's javax.servlet.http package, two are part of the JSP javax.servlet.jsp package and some is in Java core API.

JSP supports implicit objects which are listed below:

Implicit Object	Type	Scope
request	javax.servlet.HttpServletRequest	Request
response	javax.servlet.HttpServletResponse	Page
session	javax.servlet.http.HttpSession	Session
application	javax.servlet.ServletContext	Application
page	javax.servlet.jsp.HttpJspPage	Page
pageContext	javax.servlet.jsp.pageContext	Page
out	javax.servlet.jsp.JspWriter	Page
config	javax.servlet.http.servletConfig	Page
exception	java.lang.throwable	Page

2.6.1 request Object

The request object is an instance of HttpServletRequest interface. This object is used to retrieves the values that the client browser passed to the server during HTTP request such as cookies, headers or parameters associated with the request. The most common use of request object is to access query string values. You can do this by calling the getParameter() method. You have already seen the example of request.getParamater() method in <jsp:include> and <jsp:forward>.

For example:

```
<html><body><h2>Example of request Object</h2>
<% // Get the Programme Name from the request Query
    String name=request.getQueryString();
    out.println("Hello: "+name); %>
</body></html>
```

When you run this program, it looks for your name in the query and returns the value, if it is found. Enter the following URL into your browser to see the result.

<http://localhost:8080/BCS053/jsp/reqObject.jsp?poonam>

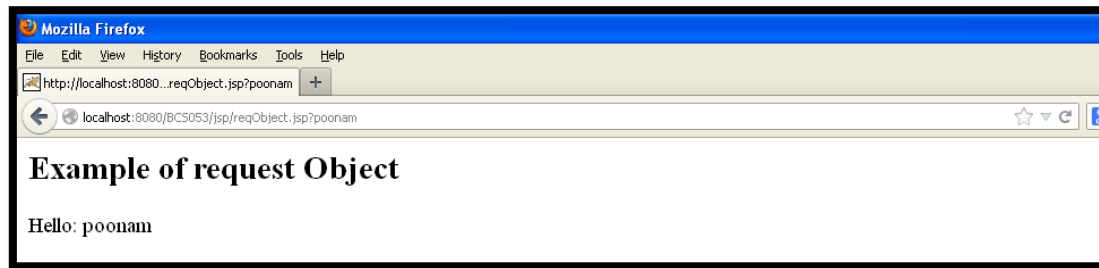


Figure 15: Example of Request Object

In the above program, `getQueryString()` method of request object is used to return the query string from the request.

2.6.2 response Object

As you know, response is a process to responding against it request. Using response object, reply is sent back to the client browser. Through this object, response parameter can be modified or set. This object is used to handles the output of client. The response object is an instance of *HttpServletResponse* interface.

```
<% response.sendRedirect("http://www.ignou.ac.in");%>
```

When the above code is run in Tomcat server, the *sendRedirect()* method of the *javax.servlet.HttpServletResponse* to redirect the user to a different URL. In this case, control transfer to IGNOU website.

2.6.3 session Object

The session object is represented by the *javax.servlet.http.HttpSession* interface. This object is behaves in same way as under the java servlet. For each user, servlet can create an *HttpSession* object that is associated with a particular single user. This object is used to track the user information in a same session. Session tracking allows servlet to maintain information about a series of request from the same user. This can be done by URL rewriting, cookies, hidden form fields and session.

2.6.4 application Object

The application object is an instance of *javax.servlet.ServletContext*. This object has application scope which means that it is available to all JSP pages until the JSP engine shut down. The *ServletContext* is the environment where the servlet run. The servlet container creates a *ServletContext* object that we can use to access the servlet environment. There is one *ServletContext* object for each web application per Java Virtual Machine.

2.6.5 page Object

It represents the *javax.servlet.jsp.HttpJspPage* interface. This object is reference to the current instance of the JSP page. The page object is a synonym for *this* object and is not useful for programming language.

2.6.6 pageContext Object

The `pageContext` object is an instance of `javax.servlet.jsp.PageContext`. It is used to represent the entire JSP page. The `pageContext` object is used to set, get and remove attribute of the JSP page.

It provides a single point of access to many of the page attribute such as directives information, buffering information, `errorPageURL` and page scope. It also provides a convenient place to store shared data. This object stores references to the request and response objects for each request. The `PageContext` class defines several fields, including `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE`, and `APPLICATION_SCOPE`, which identify the four scopes.

2.6.7 out Object

`Out` is a very simple and frequently used implicit object in scriptlet. JSP expression is automatically placed in output stream. So, `out` object is rarely needed to refer in expression. It is an instance of `javax.servlet.jsp.JspWriter` class which extends `java.io.writer`. You call either `print()` or `println()` method to send output to the client.

For example: `<% out.println(" Hello! Out Object"); %>`

2.6.8 config Object

The `config` object is an instance of `javax.servlet.http.servletConfig`. This object is used to get the configuration information of the particular JSP page.

2.6.9 exception Object

This object is available only on pages that are marked as an error page using the page directive `isErrorPage` attribute. This object is assigned to the `Throwable` class which is the super class of all errors and exceptions in the java language. It contains reference to uncaught exception that caused the error page to be invoked.

Check Your Progress 3

1. What are the implicit objects?

2. What is request object? Explain with example.

3. Which implicit object allows the storage of values in an associated hash table?

4. Write Java code using JSP component for generating odd numbers.

5. Write a JSP program for displaying a Remote address using scriptlets.

2.7 Complete Example

The following program displays a JSP login page which accepted login information from the user and counts the number of times user visit the page.

Source code for header.html

```
<html><head><title>BCA:BCS-053</title>
<style>h1 { text-align:center }</style>
</head><body><table><tr><td></td><td>
<h1>Indira Gandhi National Open
University</h1></td></tr></table></body></html>
```

Source code for visitCount.java

```
package bean1;
public class visitCount {
int count=0;
public visitCount(){ }
public int getCount(){ count++;
return this.count;}
public void setCount(int count)
{ this.count=count; } }
```

Source code for footer.jsp

```
<html><head><title>BCA:BCS-053</title></head><body>
<% @ page language="java" %>
<jsp:useBean id="counter" scope="session" class="bean1.visitCount" />
<jsp:setProperty name="counter" property="count" param="count" />
<footer><hr> &copy; 2013 IGNOU</footer>
Visitor Count:
<jsp:getProperty name="counter" property="count" />
</body></html>
```

Source code for main.jsp

```
<html><head><title>BCA:BCS-053</title></head><body>
<% @ include file="header.html" %>
<form action="actionJsp.jsp">
<fieldset ><legend>Login Details</legend>
Username <input type="text" name="uname"/><br>
Password <input type="password" name="pwd"/> <br></fieldset>
<input type="submit" value="submit"> </form><br>
<jsp:include page="footer.jsp" />
</body></html>
```

Source code for actionJsp.jsp

```
<html><head><title>BCA:BCS-053</title></head><body>
<% @ include file="header.html" %>
<% // Get the User Name from the request
    out.println("<b>User Name:</b>" + request.getParameter("uname"));
    out.println("<br>");
    // Get the Password from the request
    out.println("<b>Password:</b>" + request.getParameter("pwd")); %>
</body></html>
```

For storing and running the above programs, you can use the following steps. The following steps are based on windows operating system. In lab manual block (BCSL057), the detailed description of Netbeans with Web Server - Apache and Glassfish are given for creation and deployment of JSP program. It is best practice to use IDE for web development. In this section, you can go through the simple steps for running your JSP program using Apache Tomcat without any IDE. You can use Notepad to write your program.

Step 1 - Download and install Java development Kit.

Step 2 - Download and install Tomcat

Step 3 - After successfully installation of Java and Tomcat, you can set the environment variables using environment variables. For this right click on System icon → properties → Advanced System Setting → environment variables and ok. Now, click on New tab and enter variable name as JAVA_HOME and in variable value, write the path of Java installation directory and click on ok then the following screen comes after the selection of the step3.

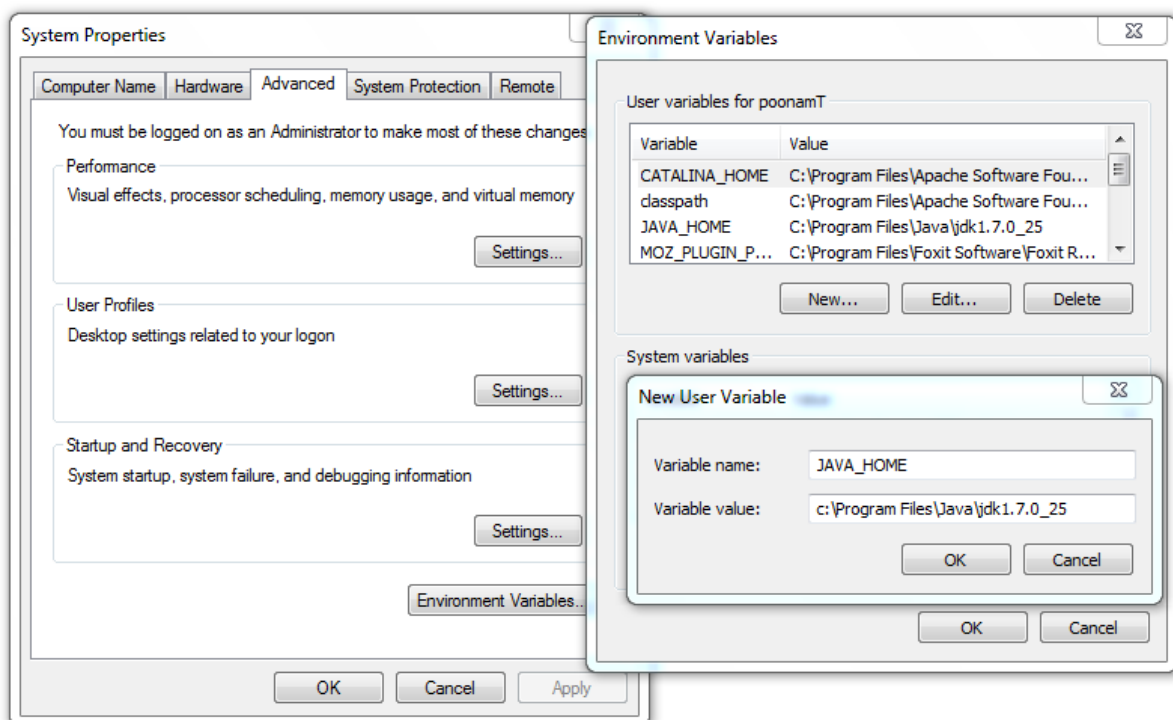


Figure 16: Screen for setting Environment variable.

In similar way, you can set the variable name and variable value for the following environment variable using step 3:

variable name	variable value:
classpath	C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\Servlet-api.jar;C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\jsp-api.jar;
path	C:\ProgramFiles\Java\jdk1.7.0_25\bin;C:\ProgramFiles\Java\jdk1.7.0_25\lib;
CATALINA_HOME	C:\Program Files\Apache Software Foundation\Tomcat 6.0

Step 4 - After installation, the tomcat folder has the following folders are bin (binary files for Tomcat and friends), conf (configuration files), lib (library JAR files), logs (server log files), temp (temporary files), webapps (area Tomcat looks for web application, it contains JSP files, Servlets and other content) and work (working space for holding translated JSP files). You can see these folders in following figure 17.

Step 5 - Create directory “ex” under the webapps as per the following figure 17:

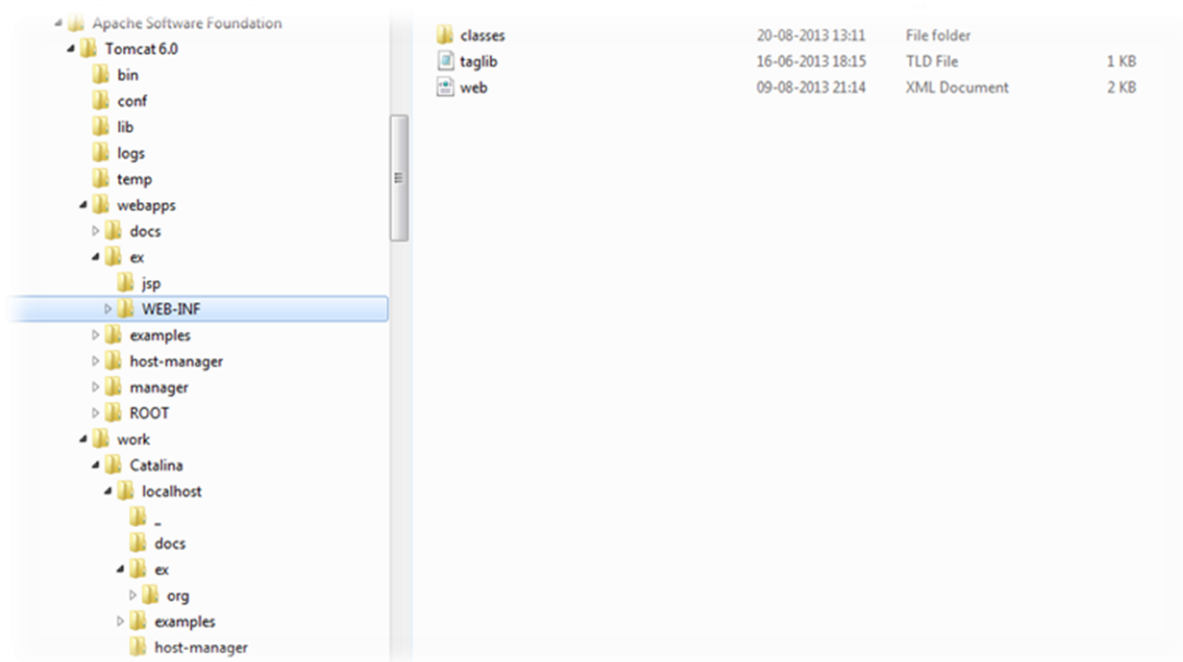


Figure 17: Directory Structure for Tomcat

Step 6 - Create directory “WEB-INF” and “JSP” folder under the “ex” folder. Create another folder named “classes” in WEB-INF folder. All your java classes used in your web application should be placed in classes folder.

Step 7 - Copy web.xml file from ROOT directory and paste into “WEB-INF” folder under the “ex” folder.

Step 8 - Save the files header.html, footer.jsp, main.jsp, actionJSP.jsp in the following location: C:/ Program Files/.../.../webapps/ex/JSP folder.

Step 9- Compile visitCount.java file and place the visitCount.class file in the following location: C:/ Program Files/.../.../webapps/ex/Web-INF/classes folder as you can see in following figure:

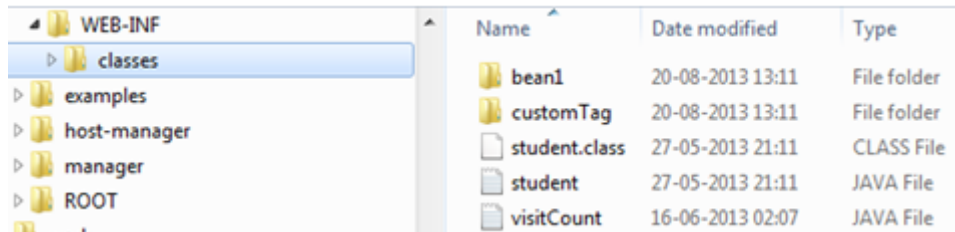


Figure 18: Screen shows classes Directory

Step 10- Start Tomcat server. Open new tab in browser or open new window :
<http://localhost:8080/ex/jsp/main.jsp>

The outputs of the above programs are as follows:



Figure 19: Login Screen

When you will click on the submit button of the above form, the following screen will be displayed with the login information.

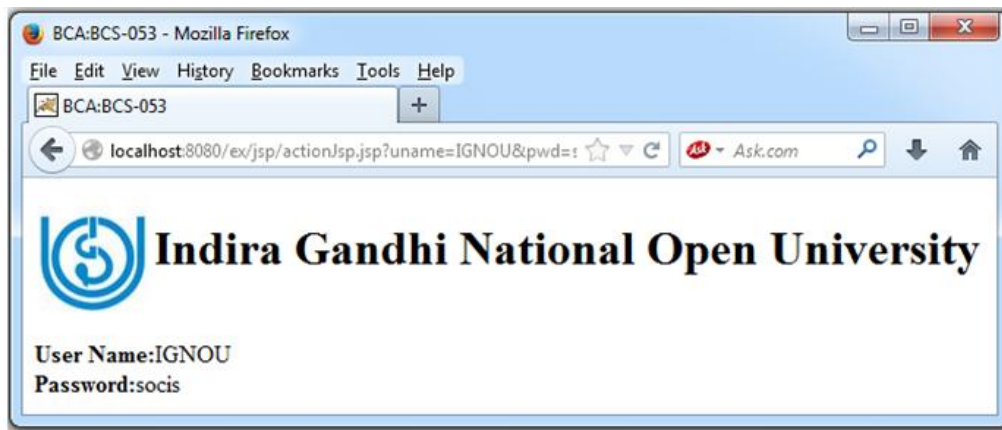


Figure 20: Output screen for above program

The above example contains many small program files such as header.html, visitCount.java, footer.jsp, main.jsp and actionJsp.jsp. The header file is used for displaying a heading for the program page. In visitCount.java, visitCount is a Java Bean class that acts as a visitor counter. It has a single int type property count that holds the current number of times the beans property has been accessed. This program also contains the methods for getting and setting the count property. In footer.jsp, the visitCount bean is integrate with JSP page using the `<jsp:useBean>` standard action. Other two action elements `<jsp:setProperty>` and `<jsp:getProperty>` are also defined for the count property. In main.jsp file, header and footer files are included by using the include directive and include action. Two input form control is defined for accepting the user's login information. Another file actionJsp.jsp is also included in main program. When you will run main program named main.jsp, it will show you a form, in which you fill information and submit the form and result will appear on the screen.

2.8 Summary

In this unit, you gone through the basics of Java Server Pages. Now, you can explain the JSP technology and how is used to create web application. It is an extension of the servlet because it provides more functionality than servlets. Aside from the regular HTML, there are three types of JSP components that you embed in your document i.e. directives, scripting elements and actions. Directives are used to control overall structure of the JSP page. You have learned three types of scripting elements such as declaration, expression, and scriptlets. Some Standard actions are specific to Java Bean and other actions are used to include or forward request, add applets or plugin within JSP page. You have also learned about the objects that are available for use in scriptlet or expression without being declared. Now, you have able to create a web page in JSP and also able to create/access a Java Bean within the JSP.

2.9 Solutions/Answers

Check Your Progress 1

Ans1:

JSP stands for Java Server Pages. It was developed by Sun Microsystems in 1999. JSP page or document is normal html page with embedded Java code. A JSP page contains static data

as well as dynamic data. The static data is written in normal html form and dynamic content is constructed JSP elements. The file extension for the source file of a JSP page is .jsp. JSP is a technology used to develop interactive Web pages.

Ans2:

Both Servlet and JSP are server side technology but the development process of JSP page is easier than Servlet. The content and display logic is separated in a JSP page and you can insert directly java code inside the JSP page while in servlet, you can write java code with plain html using plenty of `out.println()` statement for front end display of the page which is very tedious work. Another advantage of JSP page over the Servlet is that the JSP page can use custom tags while in Servlet, it is not possible. The power of JSP is to provide a framework for Web application development.

Ans3:

The `_jspService()` method is defined in `javax.servlet.jsp.HttpJspPage` interface and it is invoked every time a new request comes to a JSP page. This method takes `HttpServletRequest` and `HttpServletResponse` objects as an arguments. It returns no value. A page author cannot override this method. It is defined automatically by the processor.

Ans4:

When a client requests a JSP page, the browser sends a request to web server which is forwarded to the JSP engine. If it is a new request from the client then it translated and compiled by the JSP engine into a servlet. Now, servlet is ready for servicing the client request and generates response which returns back to the browser via web server. For the second time same request from the client including browser and web server request to JSP engine, the JSP engine determines the request that the JSP-Servlet is up to date, if yes then it immediately passes control to the respective JSP-Servlet.

Ans5:

The life cycle of JSP document contains three phases from initialization to destruction. These phases is controlled by three methods i.e. `jspInit()`, `_jspService()` and `jspDestroy()`.

The initialization phase include `jspInit()` method which is called only once during life cycle of a JSP. The `jspInit()` method is used to initialize objects and variables that are used throughout the life cycle of JSP. Another method of JSP life cycle is `_jspService()` method. It is called every time the JSP page is requested to serve a request. The destruction phase of the JSP life cycle starts when a JSP is being removed from use by a container. It has no parameters, return no value and thrown no exceptions. Override `jspDestroy()` when you need to perform any cleanup, such as releasing database connections or closing open files.

Check Your Progress 2

Ans1:

The directives are JSP component that provide global information about the page. It has further three elements such as page Directive, include Directive and taglib Directive.

The syntax of the directive is as follows:

```
<% @ directive {attribute = "value" } %>
```

Ans2:

The standard actions are tags that are embedded in a JSP page such as forward, include and many more. It is also called as action elements. You can dynamically insert a file, reuse a Java Bean, forward or include a request to/from the other page. There are three action elements specific to Java Bean i.e. <jsp:useBean>, <jsp:setProperty> and <jsp:getProperty>.

Ans3:

This include directive is used to insert text and code at the translation time. You can not pass the parameter in this directive. The syntax is: <%@ include file="relativeURL" %>

The include action is used to include static as well as dynamic content in the current JSP page at the request or run time. You can pass the parameter using param action. The syntax is :

```
<jsp:include page ="relativeURL"/> or
<jsp:include page ="relativeURL"/> <jsp:param ../></jsp:include>
```

Ans4:

The declaration tag is used to declare variables and methods which are placed inside the declaration part of the generated servlet. In JSP page, the declaration tag is start with <%! and end with %>. The code inside this tag must end with semicolon. For example:

```
<html><head><title>JSP Example</title></head>
<body><p>Example of declaration tag </p>
<%! private int i = 4; %>
<%! private int squire(int i)
{ i = i * i ; return i; }%>
<%= squire(i) %> </body></html>
```

Ans5: Source code for <jsp:include> action :

```
<html> <head><title>JSP:Include example</title></head><body>
  <jsp:include page="faculty.jsp">
    <jsp:param name="name1" value="Sh. Shashi " />
    <jsp:param value="Director" name="desig1"/>
    <jsp:param name="name2" value="Sh. Ram Kumar" />
    <jsp:param value="Associate Professor" name="desig2"/>
  </jsp:include>
</body></html>
```

Source code for faculty.jsp file

```
<html><head><title>faculty.jsp</title></head><body>
  <b><i>
    <% out.print("Name : "+request.getParameter("name1")); %></i></b> <br>
    <% out.print("He is a "+request.getParameter("desig1")+ " Dept.: SOCIS"); %> <br>

    <b><i><% out.print("Name : "+request.getParameter("name2"));%></i></b><br>
    <% out.print("He is a "+request.getParameter("desig2")+ " Dept.: SOCIS "); %>
  </body></html>
```

Check Your Progress 3

Ans1:

In Java Server Pages, there are certain objects such as request, response, session and application are automatically available for JSP documents. These are called implicit objects and are summarized in the following table:

Implicit Object	Type
request	javax.servlet.HttpServletRequest
response	javax.servlet.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
page	javax.servlet.jsp.HttpJspPage
pageContext	javax.servlet.jsp.pageContext
out	javax.servlet.jsp.JspWriter
config	javax.servlet.http.servletConfig
exception	java.lang.throwable

Ans2:

The request object is used to access request parameters. You can do this by calling the `getParameter()` method of request object. For example,

```
<html><head><title>A request example</title></head>
<body>
<% String UserName = request.getParameter("UserName");
out.println("User Name = " + UserName);
%>
</body></html>
```

Ans3:

The `pageContext` object is an instance of `javax.servlet.jsp.PageContext`. It is used to represent the entire JSP page. The `pageContext` object is used to set, get and remove attribute of the JSP page. It provides a single point of access to many of the page attribute such as directives information, buffering information, `errorPageURL` and page scope. It is also provides a convenient place to store shared data.

Ans4:

```
<% @ page language="java"%>
<html><head><title>Odd number example written in JSP</title></head>
<body><p>Odd number are:</p>

    <% for(int i=0;i<=100;i++) {
        if((i%2)!=0)
        { out.println(i); out.println(""); } }
    %>
</body></html>
```

Ans5:

```
<html><body>  
<% out.println("Remote Address is :" + request.getRemoteAddr());%>  
</body></html>
```

2.10 Further Reading

- Professional JSP..... Brown-Burdick, Apress, SPD.
- Java for the web with Servlets, JSP...Budi Kurniawan, Techmedia
- Pure JSP....Java Server Pages, James Goodwill, Sams, Techmedia
- JavaServer Pages, Hans Bergsten, O'Reilly
- <http://tomcat.apache.org>
- <http://www.ibm.com/developerworks/java/tutorials/j-introjsp/section2.html>

UNIT 3 JSP - Applications

Structure

- 3.0 Introduction
- 3.1 Objective
- 3.2 Exception and exception handling using JSP
 - 3.2.1 Error handling at the page level
 - 3.2.2 Error handling at the application level
- 3.3 Session Management
 - 3.3.1 Cookies
 - 3.3.2 URL Rewriting
 - 3.3.3 Hidden Fields
 - 3.3.4 Session Objects
- 3.4 Managing Email using JSP
- 3.5 Summary
- 3.6 Solutions/Answers
- 3.7 Further Readings

3.0 Introduction

In the previous unit, you have learned how to create JSP web pages. When you include java code inside the html page, it becomes Java Server Pages (or JSP). Java Server Pages are simple but powerful technology used to generate dynamic web pages. Dynamic web pages are different from static web pages in which web server creates a web page when it is requested by a client or user. For example, when you see your online results on IGNOU website, different pages are generated for different students. It is not the same static page for all; rather IGNOU web server dynamically creates different pages depending on your roll number. You might have seen that when you enter your roll number in the input field, it shows you some roll number for your selection, do you know how the web server maintains or remembers such data. In fact, it is all possible because of cookie. You will study more about how web server maintains session management through cookie, session object, hidden form field and URL rewriting.

You have been introduced to operate a JSP page but without debugging and handling error, the development process is not complete. This unit will also introduce you about exception handling and managing email using JSP.

3.1 Objectives

After going through this unit, you will be able to:

- define and implement JSP error page
- write program for handling exception at page and application level in JSP
- explain the use of deployment descriptor
- define the session management
- use different techniques to achieve session management i.e. cookie, session objects
- sending email using JSP

3.2 Exception and exception handling using JSP

Web applications can sometimes generate a numbers of errors that you do not want to see. You can only expect an information rich page is display instead with an ugly and unexpected HTTP status code such as “404 Not Found” and “408 Request Timeout” in web browser. You can handle these error codes and runtime exception with an informative error messages. You can do this by creating an error page. You can write program codes for handling exception after going through this unit.

An exception is an abnormal condition that arises in a program code at run time. The exception can appear any time in web application, so it is necessary to write code for exception handling. Exception is a run time error or you can say that the exception is an object that is thrown at run time and exception handling is a process to handle the runtime errors. An exception can occur if you trying to connect to a database which does not exists or the database server is down, it may be thrown if you are requesting for a file which is unavailable, then the exception will be thrown to you.

In Java Server Pages (JSP), there are two types of errors i.e. translation time or compilation time error and run time or request time error. These errors can occur in JSP in two different phases of its life.

JSP Compilation Time Error

The first type of error comes at translation or compilation time when JSP page is translated from JSP source file to Java Servlet class file. These errors are usually the result of compilation failures due to some syntax error or spelling mistakes. These errors are known as translation time errors and are reported to the client browser with some error status code 500. The compilation time error is handled by JSP engine.

For example: In following scriptlet code, tag is improperly terminated.

`<% out.println("Hello Students in JSP World"); >`

When you will run the above code, it shows the following error description. The syntax error is in the one line scriptlet code. In following figure, you will see the error “unterminated % tag” in first.jsp named file. It means that the program code line is not terminated properly with %>; the percentage sign is missing.



Figure 1: JSP Compilation time error

When you will run the following proper scriptlets code line terminating with % sign, it will display the correct result of program code.

```
<% out.println("Hello Students in JSP World");%>
```

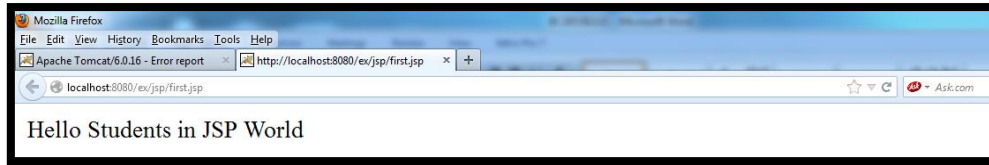


Figure 2: Simple JSP page

JSP Request Time Error

The second type of JSP error which is called as request time error occurs during run time or request time. These run time errors results in a form of exception. These run time exception can be caught and handled by the calling JSP. These exceptions occur in the body of the JSP Page. In following section, you will find more examples for the run time exception.

For exception handling, you need to create and define JSP error page. You can define an error page in two ways i.e. page level and application level. In page level, an error page contains the exception handling code description for a particular page. It means that the exception handling code is defined on each page and if there is an unhandled exception thrown from that page, the corresponding error page will be displayed. The page wise error page is defined when dealing with exception using page directive. In application level, error page contain source code for not any specific web page rather it is defined for all pages by describing in deployment descriptor i.e. web.xml file.

The following sections describe in detail how to define and implement the error pages. Also, exception can be handled in Java server pages in the following way:

3.2.1 Error handling at the page level

In this approach, you can write the code for exception handling only for the particular JSP page using page directive and Java standard exception mechanism. In the page directive, there are two attribute for exception handling i.e. `errorPage` and `isErrorPage`. You can create an error page using `errorPage` attribute of page directive element of JSP. In standard Java mechanism, you can use try and catch clause to capture the exception which is thrown within scriptlets element of the current JSP page.

3.2.1.1 Using page Directive

You have already been learned about the page directive in the previous unit. The page directive is used to specify the properties of the JSP page. The two attributes of the page directive such as `errorPage` and `isErrorPage` are useful for in JSP exception handling and also, one implicit object “exception” is used in handling JSP errors.

The errorPage attribute

The `errorPage` attribute of page directive is used to specify the name of error page that handles the exception. You can handle the exceptions in JSP by specifying `errorPage` in the page directive. It is used as follows:

```
<%@ page errorPage="relative URL" %>
```

The isErrorPage attribute

The isErrorPage attribute of page directive indicates whether or not the JSP page is an errorPage. The default value of this attribute is false.

```
<%@ page isErrorPage="true" %>
```

Here is the example for exception handling at page level or within current JSP page using errorPage and isErrorPage attributes of page directive. In this case, you must create an error.jsp named file which contains source code for handling exception and the other page named main.jsp file, where may exception occur, define the errorPage attribute of page directive. The third file input.jsp is used for input values. This example is for dividing two values and displays the result.

Source code for input.jsp:

```
<html> <head> <title>Input.jsp</title> </head> <body>  
<form action="main.jsp">Enter Number 1<input type="text" name="num1">  
<br>Enter Number 2 <input type="text" name="num2"> <br>  
<input type="submit" value="submit"> </form> </body></html>
```

Source code for main.jsp

```
<% @ page errorPage="error.jsp" %>  
<html> <head> <title> main.jsp </title> </head> <body>  
  <% String n1 = request.getParameter("num1");  
  String n2 = request.getParameter("num2");  
  int Var1 = Integer.parseInt(n1);  
  int Var2 = Integer.parseInt(n2);  
  int Var3 = Var1 / Var2;  
  out.println("First number = "+ Var1);  
  out.println("Second number = "+ Var2);  
  out.println("Division of two numbers are "+ Var3); %>  
</body></html>
```

Source code for error.jsp

```
<% @ page isErrorPage="true" %><html> <head>  
<title>error.jsp</title></head> <body>  
  Your page generate an Exception : <br>  
<%= exception.getMessage() %> </body></html>
```

Output of the above programs:

When you will run the above program code, it shows the following two screens one after another. In the first screen, there are two input box where you will input two integer value and click on submit button. In the second screen, the browser displays the result after the division of two numbers.



Figure 3: JSP page for division of two numbers

Now, you will run the above same program again and input an integer value in first text box and zero in second text box and click on submit button. Now, the program will generate an Arithmetic exception: division by zero. When you will input any float value in any text box, then also it will generate exception.



Figure 4: JSP page for handling exception

3.2.1.2 Using Java Mechanism

If you want to handle exception within the current JSP page, then you can also use standard java exception handling mechanism. For this, you can use try and catch clause. In try block, you can write the normal code that might throw an exception. The catch block is used to handle the exception. Both the try and catch clause are written inside the scriptlets component of the JSP page. Unlike page directive, there is no need to be written an error page for this mechanism.

You can use the try and catch block like the following:

```
<% try { // code that thrown an exception }
    catch (exception e) { // exception handler for exception occur in try block }
%>
```

The following program code is for handling exception using try and catch block clause of Java. In this program, normal coding is defined in try clause and exception handling code is in catch block. In try block, there are three integer variable x, y and z are defined. The variable x contains a value zero and y contains value 18. When the program tries to divide the value of y by x then exception is occurred. The program control flow is preceded and display only those statement which are included in catch block.

```

<html><head><title>Using standard Java mechanism</title></head>
<body><h3>Exception handling through try and catch clause</h3>
<%
    int x, y, z;
    try { // monitor a block of code.
        x = 0; y = 18; z = y / x;
        out.println("This will not be printed.");
    }
    catch (ArithmeticException e) { // catch divide-by-zero error
        out.println("Division by zero."); }
    out.println("After catch statement."); %>
</body></html>

```

Output of the program:

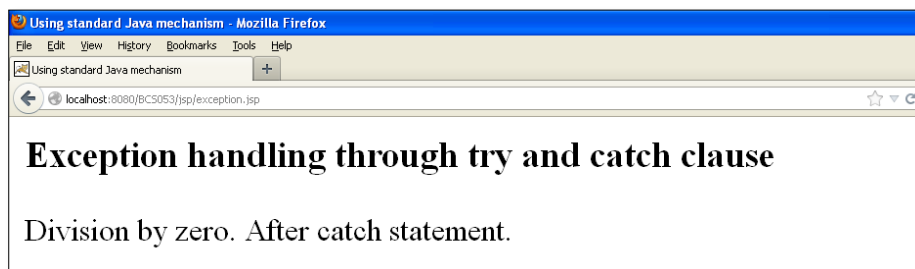


Figure 5: Exception handling through standard Java mechanism

3.2.2 Error handling at the Application level

In previous section, you have learned about the exception handling at page level using page directive and standard java mechanism. The following section describes the exception handling at the application level.

3.2.2.1 Using Deployment Descriptor

You can also handle exception in JSP page at application level by specifying the error page using `<error-page>` element in the deployment descriptor. The deployment descriptor is a file named `web.xml`. It resides in the web applications under the `WEB-INF/` directory. This approach is better if you want to handle any exception in any JSP page. Under the `<error-page>` element, you can specify either an `<exception-type>` element with the class name of the expected exception such as `java.lang.ArithmeticException` or `<error-code>` element with an HTTP error code value such as 500 with `<location>` element. The `<location>` element tells JSP container for URL path of the resource to show when the error occurs. Unlike page directive, you do not need to specify the `isErrorPage` attribute in each JSP page. You can specify only one time the name of error page in the `<location>` element.

To include a generic error page for all exceptions at application level in the following way in `web.xml` file:

```

<error-page>
  <exception-type>Exception</exception-type>
  <location>/error.jsp</location>
</error-page>

```

If you want to handle exception using any specific error status code such as File not found error 404, Server error 500, then you can specify `<error-code>` element instead of `<exception-type>`. This is the best way to declare error page using error-code element in web.xml file. It is used as follows:

```
<error-page>
  <error-code>500</error-code>
  <location>/jsp/error.jsp</location>
</error-page>
```

Here is the example for exception handling at application level using `<error-page>` element. The example code is very similar to the above used example in page directive option. For this example, four files are needed to run the program:

- input.jsp file for input values (It is same as the above example in page directive section 3.2.1)
- main.jsp for dividing two numbers and displaying the result
- error.jsp file for displaying the exception (which is also same as the above example defined in page directive option)
- web.xml file for specifying the `<error-page>` element

Source code for main.jsp: Now, you do not need to specify `errorPage` attribute of page directive in each JSP page.

```
<html> <head> <title> main.jsp </title> </head>
<body>
  <% String n1 = request.getParameter("num1");
  String n2 = request.getParameter("num2");
  int Var1 = Integer.parseInt(n1);
  int Var2 = Integer.parseInt(n2);
  int Var3 = Var1 / Var2;
  out.println("First number = "+ Var1);
  out.println("Second number = "+ Var2);
  out.println("Division of two numbers are "+ Var3); %>
</body></html>
```

Source code for web.xml: You can include the following code in your web application under the WEB-INF/web.xml file. The `<location>` element tells the web container for location of the error page. In this example, error.jsp file is placed under the jsp/error.jsp folder in web application.

```
<web-app>
.....
.....
<error-page>
  <error-code>500</error-code>
  <location>/jsp/error.jsp</location>
</error-page>
.....
</web-app>
```

Output of the program:

When you run the input.jsp program file and input two integer value in the respective boxes, the program is successfully run and display the following output screens.



Figure 6: Simple program without an exception

When you run the above program code again with incorrect values such as zero or float values in input text field then program will generate an exception according to your input values.



Figure 7: Exception handling at application level by using deployment descriptor

Note: The page directive declaration overrides any matching error page configurations in deployment descriptor. If the JSP page throws `java.lang.ArithmeticException` and deployment descriptor has an exception-type attribute for that exception, the web container invokes the page directive instead of any URI specified in deployment descriptor (web.xml) configuration.

Check Your Progress 1

1. Explain the term JSP error Page.

.....

2. Explain the two attributes of page directive which is included in exception handling in JSP page.

.....

3. What is the purpose of `<error-page>` element in JSP page and where it is used?

.....

4. How can you write code fragment for 'File Not Found Error 404' using deployment descriptor?

.....

5. Explain the use of deployment descriptor in JSP.

3.3 Session Management

The Hypertext Transfer Protocol (HTTP) is the network protocol that web server and client machine use to communicate with each other. HTTP is a stateless protocol which means that it cannot persist the data or you can say that it does not remember a thing. When you send a request as a client to server through HTTP, HTTP treats each request as a new request. So every time you will send a request, you will be considered as a new user. It is not reliable when you are doing some kind of business transaction such as online banking or other work that are most important, where the persistence of the data is necessary. To remove this obstacle, you can use session management.

Before going in detail of session management, you should know two things that are necessary for implementing flexible business transaction across the multiple request and response. These are as follows:

- **Session:** The server should be able to identify a series of request from the same user form a single working ‘session’. For example: In online banking application, each user can distinguish from another user by associating a specific request with a specific working session.
- **State:** The server should be able to remember information related to previous request and other business decisions that are made for the request. For example: In online banking application, the state contains user’s account number, amount or transaction made within the particular session.

Session management also called session tracking, does not change the nature of HTTP protocol i.e. statelessness feature and it just provides a way to remember the information that has been requested or entered by the user. Session management allows JSP/Servlet to maintain information about a series of request from the same user.

Generally, there are four techniques to session tracking:

- URL rewriting
- Cookies
- Hidden fields
- Session object

Each of the above techniques are differ in their implementation but all are based on one principle that is to exchange data in the form of token or session ID between the client and server. In the following figure, client sends a request to server for the first time, a unique

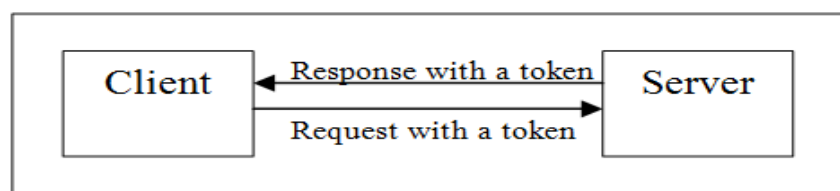


Figure 8: Client - Server model for session tracking

token is generated by the server and transmitted to the client by the response object and

stored on the client machine as a cookie. Whenever, the client visits a server again, it will send a request along with token to the server. On the server side, server can recognize client with this token. This is the way of working nature of session tracking.

The following section describes the above said four techniques and how these techniques represent the tokens:

3.2.1 Cookies

A cookie, also known as an HTTP cookie, web cookie, or browser cookie, is a small piece of data sent from a website and stored in a user's web browser while a user is browsing a website. When the user browses the same website in the future, the data stored in the cookie is sent back to the website by the browser to notify the website of the user's previous activity.

Cookies are widely used session tracking technique. A cookie is small piece of information that is sent by the server to client browser. It is stored at the client machine in the form of text file. It is basically information about the user viz., username, password, email id, date of login and identification number etc. When the next time, browser sends request along with cookies, then the server uses these cookies to identify the user.

Typically, cookies are set at the beginning of the JSP page because they are sent as a part of HTTP headers. The creation and maintenance of cookies are the same the servlet. For this, servlet API provides a `javax.servlet.http.Cookies` package. You can create a cookie by calling a constructor of `Cookie` class and passing a name /value pair of parameter. The signature of the `Cookie` class is defined as follows:

```
public Cookie(String name, String value)
```

For example, the following source code line creates a cookie object named `cookieOne`. This object `cookieOne` has the name “nameCookie” and value of “valueCookie”. Both the name/value pair is defined by the user.

```
// create a new cookie  
Cookie cookieOne = new Cookie("nameCookie", "valueCookie");
```

You can add this cookie to JSP in-built response object using `addCookie()` method of `HttpServletResponse` interface:

```
response.addCookie(cookieOne);  
// send the cookie to browser to be stored on client machine
```

Now, you can retrieve a cookie from the request using `getCookie()` method of `HttpServletRequest` interface:

```
request.getCookie(cookieOne); // retrieve a cookie
```

You can also set the life of the cookie using the `setMaxAge()` method. Look the following code:

```
public void Cookie.setMaxAge(int expiry)
```

You can specify the age of cookie using the above method in seconds. A negative value indicates the default value; it should be expired when browser exits. A zero value indicates that the cookie is immediately deleted by the browser. For example:

```
cookieOne.setMaxAge(60 * 60) // expire in 1 hour  
cookieOne.setMaxAge(0) // to remove the cookie
```

Here is the complete example of the cookie:

In following example, there are three programs namely CookiePrg1.jsp, CookiePrg2.jsp and CookiePrg3.jsp. In first CookiePrg1.jsp program, there is defined only one text box where you can input university name and click on the Go button. When this form is submitted, control transfer to the second program named CookiePrg2.jsp, where the cookie is set an age using the cookie.setMaxAge() method. In the third CookiePrg3.jsp program, the cookie is retrieved using the method request.getCookies() and the value of the cookie is displayed. Normally, you can get the value of the form fields using request.getParameter() method on the second or action page but the purpose of this example using three programs is used to show you the value of cookie is displayed on the third page. In this way, you can get the value of cookie on any web page.

Source code for CookiePrg1.jsp

```
<html><body>  
  <form method = "post" action="CookiePrg2.jsp">  
    University Name<input type = "text" name = "name"><br>  
    <input type = "submit" name = "submit" value = "Go" >  
  </form></body></html>
```

Source code for CookiePrg2.jsp

```
<% @ page language="java" import="java.util.*"%>  
<%  
  String name=request.getParameter("name");  
  Cookie cookie = new Cookie ("Name",name);  
  response.addCookie(cookie);  
  cookie.setMaxAge(60 * 60);  
  %>  
<a href="CookiePrg3.jsp">Continue</a>
```

Source code for CookiePrg3.jsp

```

<p>Display the value of the Cookie</p>
<%
    Cookie[] cookies = request.getCookies();
    for (int i=0; i<cookies.length; i++)
    {
        if(cookies[i].getName().equals("Name"))
            out.println("University Name = "+cookies[i].getValue());
    }
%>

```

When you will run the above programs, the following output is displayed in your browser.

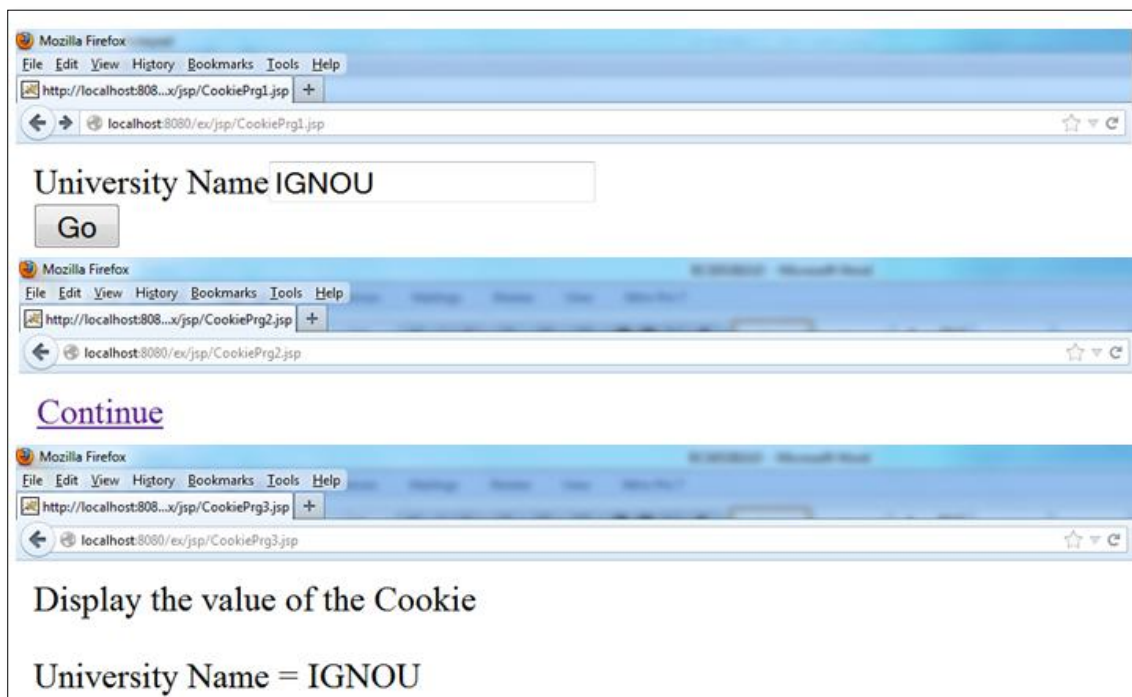


Figure 9: Simple program for Cookie

3.2.2 URL Rewriting

URL rewriting is used in place of the cookies when the browser cookies are functionally turned off. It is used to maintain the session. In this approach, the token is embedded in each URL. In each dynamically generated page, the server embeds an extra query parameter or extra path information. When client submits request using such URL, the token is retransmitted to the server. You can send name/value pairs of parameter in the following format:

<http://ignou.ac.in/student.jsp?name1=value1&name2=value2&.....>

A name and value parameter is separated by equal (=) sign, a parameter name/value pair is separated by another parameter name/value pair using ampersand (&) sign. When you click the hyperlink, the parameter name/value pairs will be passed to the server. From JSP page,

you can use `getParameter()` method of `HttpServletRequest` interface to retrieve the parameter value. You can write the following code for retrieving parameter value.

```
request.getParameter(name1);
```

For example: In the following hit counter program, the two variable `count` and `counter` are defined. `Count` is integer type and `counter` is string type variable. For the first time, the value of `counter` is retrieve using `request.getParameter()` method and it is equals to null then set the value of `count` is one. The program is display value of hit counter is one. When you will click on 'click@Link' link, value of `count` variable is passed to `counter` variable like the following way:

<http://localhost:8080/ex/jsp/index1.jsp?counter=1>

The `String.valueOf(count)` method is used to convert the integer value of `count` in string type. For the second time, when if condition is false and control transfer to else part of the program and value of `counter` variable is incremented by one each time, when you click on the link.

```
<html><head></head><body>
<h3>Page Counter Example using URL Rewriting </h3>
<% int count;
if (request.getParameter("counter") == null) count=1;
else { count=Integer.parseInt(request.getParameter("counter"))+1; }%>
<p>This is Hit Counter No.:<%=count%><br>
<a href="index1.jsp?counter=<%=String.valueOf(count)%>">Click @Link</a></p>
</html>
```

Output of the above program:

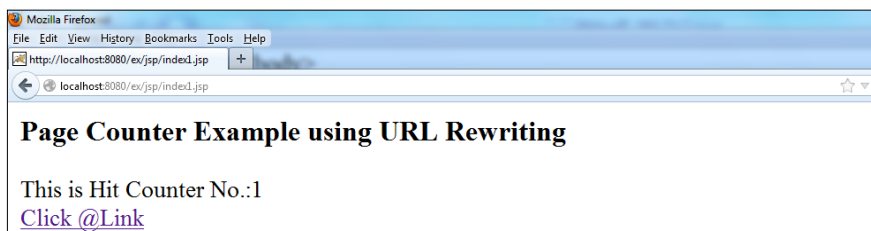


Figure 10: Simple program for URL Rewriting

3.2.3 Hidden Fields

Another technique for managing user session is by passing a token as the value for an HTML hidden field. When the client submits a form, the additional field values will also be send in the request in the form of field. Unlike the URL rewriting, the value does not display in address bar but it can be read by viewing html source code. It can be retrieved by using `getParameter()` method.

For example, the following HTML input control of type `HIDDEN`:

```
<input type="hidden" name="userId" value="ignou">
```

For example: the following example is same as the above hit counter example in URL Rewriting session management techniques.

```

<% int count;
if (request.getParameter("counter") == null) count=1;
else { count=Integer.parseInt(request.getParameter("counter"))+1; }%>
<html><head></head><body>
<form action="HiddenForm.jsp" method="get">
<h3>Page Counter Example using Hidden Form Fields</h3>
<p>This is Hit Counter No.:<%=count%></p>
<input type="hidden" name="counter"
value="<%=String.valueOf(count)%>">
<input type="Submit" value="Access New Counter"></form></body>
</html>

```

When you run the above program named “HiddenForm.jsp” example, you will get the following screen. When you run the program for the first time, the value of integer type counter variable is null and hit counter takes value one and after in a subsequent manner, it is incremented by one. Whenever you click on ‘Access New Counter’ button in same session, the form is again submitted and the value is incremented by one. The URL of the JSP page in address bar is looks like the following:

<http://localhost:8080/ex/jsp/HiddenForm.jsp?counter=2>

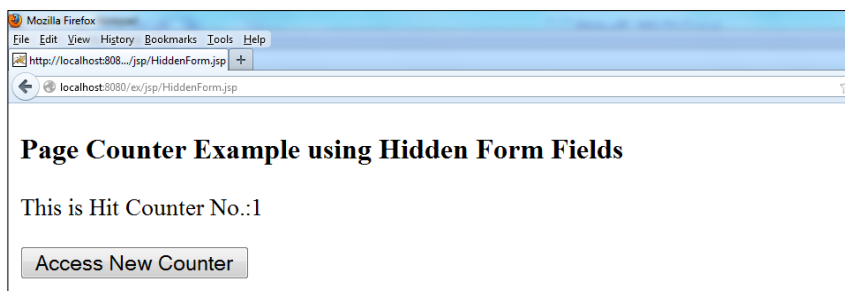


Figure 11: Program for Hidden Form Field

3.2.4 Session Object

This is easiest to use and powerful for session management. Session and Cookies go hand in hand, the only difference between the cookies and session is that the cookie is stored in the client side and session is stored on server side which is another method to handle a session.

Any web application can use the concept of sessions and Session object is used to store data for a particular client when that client is connected to server. For example, when a client logs on to a Shopping site web application, client login name, password, and other pertinent information might be stored in a Session variable and maintained during her/his visit to the site so that it can be accessed when needed. When a session begins, the requesting browser is given unique piece of information, or "token" that is presented by the browser on subsequent visits to identify the client. The Web application can then, for example, customize the settings for that client when she/he visits, since it can find her/his personal preferences using the information stored in the Session object referenced by the token. If there are 10 simultaneous clients, then 10 Session object will be created in the server and each client can access only

own HttpSession object. The Session object is represented by the javax.servlet.http.HttpSession interface.

Java Server Pages has been provided an implicit object session. So, you do not need to create a session object explicitly as you can do in servlet. The session object is defined inside the page directive of the JSP page by the following way:

```
<% @ page session = "true|false" %>
```

In JSP, the default value of session object is true. If you do not declare the session object inside the page directive then session will be automatically available to JSP page as it is default by true. The following are the some methods that are applied for session management using session object in JSP:

setAttribute() method

This method is used to add a name/attribute pair to the HttpSession object. The setAttribute() method returns an IllegalStateException on invalidated HttpSession object. This method has the following signature:

```
public void setAttribute(String name, Object attribute) throws IllegalStateException
```

You can add attributes to the session in the following way:

```
<% session.setAttribute(number, new Float(48.8)); %>
```

The above statement creates number as a session variable and assigns to a float value. To retrieve this object, you can use getAttribute() method.

getAttribute() method

This method is used to retrieves an attribute from the HttpSession object. Like the setAttribute() method, the getAttribute() method returns an IllegalStateException, if it is called upon an invalidated HttpSession object. The signature is as follows:

```
public Object getAttribute(String name) throws IllegalStateException
```

To retrieve information from a session, you simply use the getAttribute() method, like this:
Number :

```
<%=session.getAttribute(number); %>
```

Here is the example for session management using session object:

The example is included three program named sessionJSP1.jsp, sessionJSP2.jsp and sessionJSP3.jsp. In the first program, there is a form which includes two form text fields i.e. Name and Password. When you will enter these values in the respective text boxes and form is submitted, the control goes to the second program where set the username as a key for session variable, name and password is validated, if it is correct then control transfer to the third program. In the third program, value of the session object is retrieved using getAttribute() method and display a message for valid user. If it is not valid then shows an error message "name and password is invalid" using response.sendError() method. The

session is always managed with data that is stored on persistent storage. You will learn database handling concept in next unit of this block. This session management program is a static program. For testing purpose, you will input username 'IGNOU' and password 'socis' then program will run correctly otherwise it will show an error message.

Source code for sessionJSP1.jsp:

```
<html><head>
<title>Session Management: Using Session Object</title>
</head><body>
<h3>Session Management: Using Session Object</h3>
<form method = "post" action = "sessionJSP2.jsp">
  Name<input type = "text" name = "name"><br/>
  Password<input type="password" name = "pwd" ><br/>
  <input type = "submit" name = "submit" value = "submit" >
</form></body></html>
```

Source code for sessionJSP2.jsp:

```
<% String name = request.getParameter("name");
String password = request.getParameter("pwd");
if (name.equals("IGNOU") && password.equals("socis"))
{
    session.setAttribute("UserName",name);
    response.sendRedirect("sessionJSP3.jsp");
}
else { response.sendError(404, "Name and Password is invalid."); } %>
```

Source code for sessionJSP3.jsp:

```
Hello! User: <%= session.getAttribute("UserName") %>
```

Output of the program: when you will run the above program, it will display the following two screens.

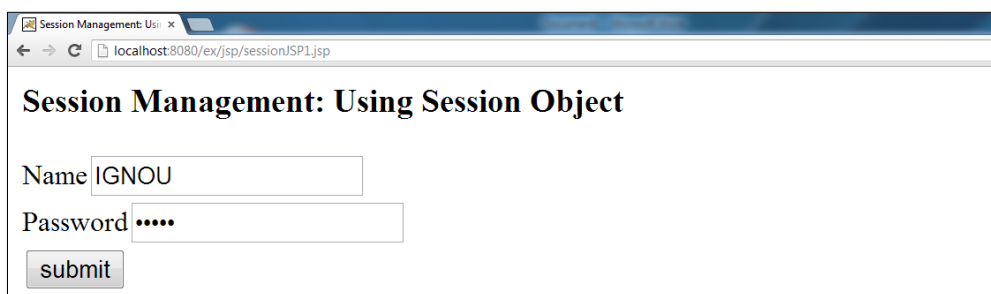


Figure 12: Session Management using Session Object

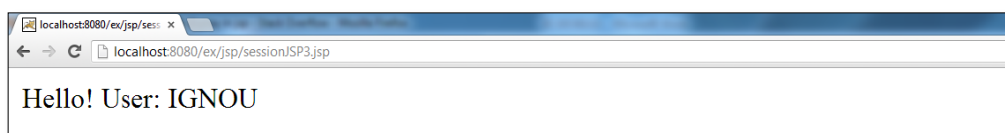


Figure 13: Validate User in Session Management

When user name and password are not validated from the program then the following screen is displayed.

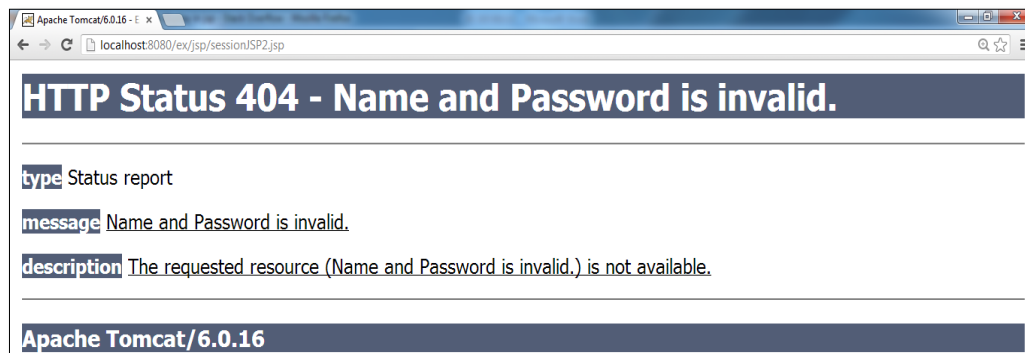


Figure 14: Error message for Invalidate user

Now that you know how session management works using the HttpSession object. The java.servlet.http.HttpSession interface has the following some additional methods for handling session:

isNew() method

This method indicates whether the session object was created with this request and if the user has joined the session, it contains true value otherwise false. The method has the following signature:

```
public boolean isNew() throws IllegalStateException
```

getCreationTime() method

The getCreationTime() method returns the time when the session was created, measured in milliseconds. The signature is as follows:

```
public long getCreationTime() throws IllegalStateException
```

getLastAccessedTime() method

The getLastAccessedTime() method returns the time when the session was last accessed by the client. The signature of method is as follows:

```
public long getLastAccessedTime()
```

getId() method

The getId() method returns the session identifier. The signature of method is as follows:

```
public String getId()
```

getMaxInactiveInterval() method

The getMaxInactiveInterval() method returns the maximum time interval, in seconds, that the container will keep the session open between client accesses. The signature of method is as follows:

```
public int getMaxInactiveInterval()
```

setMaxInactiveInterval() method

This method sets the numbers of seconds between client requests before the container will invalidate this session. The signature of method is as follows:

```
public void setMaxInactiveInterval(int interval)
```

Consider the following example for defined above all the function related to session management. The following program uses session tracking to keep track of how many times it has been accessed by a particular user and to display some details of the current session.

```
<% @ page import = "java.util.Date" %>
<h3>Session Management through Session Object</h3>
<h5>Session Details : </h5>
<%
session = request.getSession(true);
Integer count = (Integer) session.getAttribute("count");
if (count == null)
{ count = new Integer(1);}
else { count = new Integer(count.intValue()+1);}
session.setAttribute("count", count);

out.println("You have visited this page " +count+((count.intValue() == 1) ? " time." :
" times."));
out.println("<br/>");
out.println("Session ID : " + session.getId());
out.println("<br/>");
out.println("New Session : " + session.isNew());
out.println("<br/>");
out.println("Creation Time : " + new Date(session.getCreationTime()));
out.println("<br/>");
out.println("Time Out : " + session.getMaxInactiveInterval());
out.println("<br/>");
out.println("Last Access Time : " + new Date(session.getLastAccessedTime()));
%>
```

When you run the above program, it shows you the following output screen:

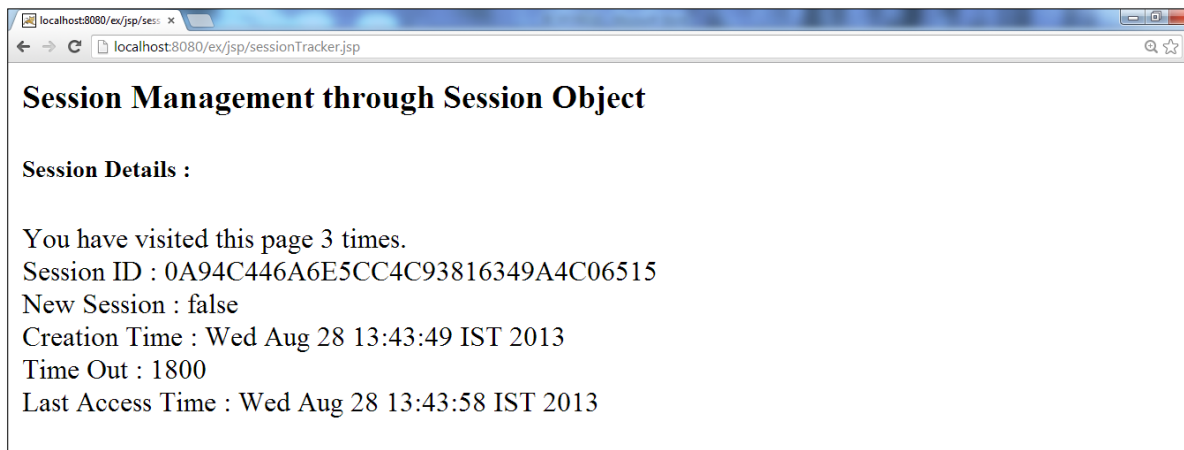


Figure 15: Session Tracking through Session Object

Check Your Progress 2

1. What are the Cookies? How do you delete Cookies within a JSP?

2. Explain Session Management in JSP?

3. What are the different session tracking methods?

4. Explain getSession() method.

3.4 Managing Email using JSP

JavaMail API is the standard mechanism used for sending email. The JavaMail API provides platform-independent and protocol-independent framework for sending and receiving emails. The javax.mail and javax.activation packages contain core classes of JavaMail API. These packages are comes in form of jar i.e. mail.jar and activation.jar. You can place these files under library directory of your web application. There are various ways to send email using JavaMail API. You must have SMTP server that is responsible for send mails. For this purpose, you can install any mail server such as Apache James server, Postcast server, email server etc.

Mailing protocols

In order to correctly send an email, client machine is used network protocol to connect the server. Basically, a protocol represents standard method used at each end of a communication channel to properly transmit information.

Generally, four protocols are used to send and receive email:

Simple Mail Transfer Protocol (SMTP):

SMTP is an Internet standard for electronic mail (e-mail) transmission across the networks. SMTP is used to deliver email to the recipient mail server. SMTP communications are transported by TCP to ensure reliable end-to-end transport.

POP (Post Office Protocol 3):

This protocol defines a single mailbox for a single user and a standardized way for users to access mailboxes and download messages to their computers as well. It provides facility to users to retrieve e-mail when connected. Once the messages are downloaded from the server, you can disconnect the Internet connection and read your mail.

IMAP (Internet Message Access Protocol):

It is a protocol by which you can read email via Internet. The email messages are stored on servers. When you check your inbox, your email client connects to the server to view your messages. When you read an email message you aren't actually downloading or storing it on your computer; instead, you are only reading it on the server, this is done by this protocol.

MAPI (Messaging Application Program Interface):

MAPI is used to send Email with in windows application and provide facility to take advantage of word processors, spreadsheets, and graphics applications.

Process of sending an email:

On the Internet, each domain has email Server, when you send an email then the following steps may be follows:

1. When you send email, your email client whether its outlook express or your JSP program connects to your SMTP Server. For example, smtp.ignou.ac.in
2. When client program communicates with SMTP Server, it sends sender and recipients' mail address along with message body.
3. The SMTP Server of the sender machine checks recipient email address especially its domain. If the domain name is same as the senders', the message is directly send to recipient domain's POP3 or IMAP server. If the domain name is different than SMTP server will have to communicate with other domain server.
4. After finding the recipient server, the SMTP server of the sender machine has to communicate with Domain Name Server (DNS). The DNS translates the recipient address into IP address.
5. The SMTP server of client machine is connected to recipient's SMTP server.
6. Now that the recipients SMTP server forwards the message to the domain's POP3 or IMAP server.

Here is a simple email program:


```

<% @ page import="java.io.*, java.util.*, javax.mail.*"%>
<% @ page import="javax.mail.internet.*, javax.activation.*"%>
<% @ page import="javax.servlet.http.*, javax.servlet.*" %>

<%
    String result;
    // Recipient's email ID needs to be mentioned here.
    String to = "abc@gmail.com";

    // Sender's email ID needs to be mentioned here.
    String from = "xyz@yahoo.com";

    // it is assumed that you are sending email from localhost
    String host = "localhost";

    // get system properties object
    Properties properties = System.getProperties();

    // set SMTP mail server
    properties.setProperty("mail.smtp.host", host);

    // get the default Session object.
    Session mailSession = Session.getDefaultInstance(properties);

    try{
        // Create a default MimeMessage object.
        MimeMessage message = new MimeMessage(mailSession);
        // Set From: header field of the header.
        message.setFrom(new InternetAddress(from));
        // Set To: header field of the header.
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
        // Set Subject: header field
        message.setSubject("This is the Subject Line!");
        // Send the actual message
        message.setContent("<h1>This is actual message</h1>", "text/html" );
        // Send message
        Transport.send(message);
        result = "Sent message successfully....";
    }catch (MessagingException mex) {
        mex.printStackTrace();
        result = "Error: unable to send message....";
    }
%>
<html><head><title>Send HTML Email using JSP</title></head>
<body><center><h3>Send Email using JSP</h3>
</center><p align="center">
<%
    out.println("Result: " + result + "\n");
%>
</p></body></html>

```

When you run this program and send an email to given email ID and would display following response:

Send Email using JSP

Result: Sent message successfully....

Check Your Progress 3

1. What are the important packages in JavaMail API?

2. Write the names of essential network protocol used in email.

3. Explain MIME?

4. Write the process of sending an email from client program to destination server.

3.5 Summary

In this unit, you have covered the exception and exception handling of Java Server Pages, session management and also managing email through JSP. Now you have known that the JSP technology is a powerful technology that is used to create web application, managing email. You can also do the exception handling in JSP pages at current and application level.

An exception is an abnormal condition that occurs at the time of program execution and it disrupts the normal flow of program code. For the exception handling, you have to create and define error page which includes the exception handling code. You can have two options for error handling at the page level and application level. At page level, you can use page directive and standard java try - catch clause for handling exception in current JSP page and also include error handler at the application level by using the deployment descriptor.

You have already known that the HTTP is a stateless protocol. When a client sends a request to server, the server sends back response to the client but does not keep information about the client request and state. The solution of this problem is session management. There are four techniques you can use to manage user sessions. In this unit, each technique has been described with examples. Additionally you learned about managing email through JSP page.

3.6 Solutions/Answers

Check Your Progress 1

Ans1:

A JSP error page is designed to handle runtime errors and display a customized view of the exception. You can include an error page in your application at page or application level. At page level, you can use page directive or standard java mechanism options. At application level, you can only use an <error page> element of deployment descriptor.

Ans2:

The `errorPage` and `isErrorPage` are two attributes of page directive which is included in exception handling in JSP at page level.

The `errorPage` attribute of page directive is used to define the name of error page that handles exception and display a customized error statements. You can use the following syntax for `errorPage` attributes.

```
<%@ page errorPage="relative URL" %>
```

The `isErrorPage` attribute of page directive indicates whether the current page can act as an error page for another JSP page. The default is *false*. It is used as follows:

```
<%@ page isErrorPage="true|false" %>
```

Ans3:

You can handle exception in JSP page at application level by specifying the error page using <error-page> element in deployment descriptor. This is the best way to handle an exception in any JSP page. The deployment descriptor file resides in the web applications under the WEB-INF/ directory. The <error-page> element is used to define the exception type or error code and location of the error page. Under the <error-page> element, you can specify <exception-type> or <error-code> and <location> element. You can use only either <exception-type> or <error-code>.

Ans4:

You can write code fragment for any specific error status code such as File not found error 404 using <error-code> element. This is the best way to declare error page using error-code element in web.xml file. It is used as follows:

```
<web-app>
.....
<error-page>
  <error-code>404</error-code>
  <location>/jsp/error.jsp</location>
</error-page>
.....
</web-app>
```

Ans5:

This file is an xml file whose root element is <web-app>. It resides in the web applications under the WEB-INF/ directory. You can configure JSP tag libraries, welcome files, customizing HTTP error code or exception type. You can use the <error-page> element in deployment descriptor to specify exception type or HTTP error code and location of the error page. The JSP tag libraries can be defined using the <tag-lib> element of deployment descriptor.

Check Your Progress 2

Ans 1:

A Cookie is a small piece of data sent from a website and stored in a user's web browser while a user is browsing a website. When the user browses the same website in the future, the data stored in the cookie is sent back to the website by the browser to notify the website of the user's previous activity. You can remove cookie in the following way:

```
<%  
    Cookie cookie = new Cookie( "name", "" );  
    cookie.setMaxAge( 0 );  
    %>
```

In the above example we have created a new instance of cookie with a "null" value and the age of the cookie is set to "0". This will remove or kill the cookie.

Ans2:

The HTTP is a stateless protocol. When a client sends a request to server, the server sends back response to the client but does not keep information about the client request and state. In a web application or website, a client has to visit number of pages for completing his task. For example, a user wants to buy some books from an online book store. User should add each book to cart and at the end and pay for them in order to complete his task. The server must maintain this type of conversation between user and the web application using different techniques i.e. Cookie, Session Object, Hidden Form field and URL Rewriting.

Ans 3:

Cookies: With this method, you can use HTTP cookies to store information. Cookies will be stored at browser side. If the browser does not support cookies, or if cookies are disabled, you can still enable session tracking using URL rewriting.

URL rewriting: The information is carried through url as request parameters.

HttpSession: Using HttpSession, you can store information at server side. HttpSession provides methods to handle session related information.

Hidden fields: By using hidden form fields, you can insert information in the web pages and this information will be sent to the server. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. For example, you can create hidden form field like the following: `<input type='hidden' name='courseCode' value='BCS053'/>`

Ans4:

The getSession() method returns the current valid session associated with the request. This method has two overloads. They are as follows:

HttpSession getSession(boolean create) : It returns the HttpSession object associated with this request if there is a valid session identifier in the request. Otherwise it return null.

HttpSession getSession() : It returns current session associated with the request or if the request does not have a session identifier, it creates a new one.

Check Your Progress 3

Ans1:

The javax.mail and javax.activation packages contain core classes of JavaMail API.

Ans2:

For sending and receiving an email, client machine is used no. of network protocol to connect the server like SMTP, POP, IMAP and some other protocol. A protocol represents standard method used at each end of a communication channel to properly transmit information.

Ans3:

Multipurpose Internet Mail Extensions (MIME) defines mechanisms for sending other kinds of information in email like images, sounds, movies, and computer programs. MIME is an Internet standard that extends the format of email to support such as text in character sets other than ASCII, non-text attachments, message bodies with multiple parts and header information in non-ASCII character sets

Ans4:

When you send an email from source to destination server the following step are follows:

1. Email client Program sends the message to Email server.
2. Email server contact to the recipient's email server provided in the email address.
3. Email server checks that user name is valid or not.
4. If it is valid send email to the address's email server.
5. When recipient log on his mail account, gets his Email.

3.7 Further Readings

- Professional JSP..... Brown-Burdick, Apress, SPD.
- Java for the web with Servlets, JSP...Budi Kurniawan, Techmedia
- Pure JSP....Java Server Pages, James Goodwill, Sams, Techmedia
- Java Server Pages, Hans Bergsten, O'Reilly
- <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>

UNIT 4 JSP APPLICATION DEVELOPMENT

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 JDBC : An Introduction
 - 4.2.1 Need for JDBC
 - 4.2.2 Two- and Three-Tier Database Access Models
- 4.3 JDBC API
- 4.4 Types of JDBC Drivers
- 4.5 Connection Steps using JDBC
- 4.6 Example Applications using JSP
 - 4.6.1 Using JDBC to Store Data in Database
 - 4.6.2 Using JDBC to Retrieve Data from Database
- 4.7 Application Development and Deployment
- 4.8 Summary
- 4.9 Solutions/Answers
- 4.10 Further Readings

4.0 Introduction

In the previous units, you have learnt about the importance of JSP. You have also studied the different components of JSP, exception handling, session management and managing email through JSP.

As you know that the JSP is basically used for server side programming. You have also known that the basic difference between static and dynamic pages. Whenever we discuss about the dynamic pages, it means that it is interactive pages and it is difficult to imagine the interactive pages of any web application that does not employ database interaction. Most of the web applications rely on back - end relational databases.

Therefore, database handling is the necessary feature of JSP. In this unit, you will learn about JDBC, different types of database drivers, connection steps for connecting database and insertion, manipulation through SQL statement in a database. At the end of this unit, you will go through the process of application deployment.

4.1 Objectives

After going through this unit, you should be able to:

- define the use and role of JDBC
- use different types of JDBC SQL statements
- appreciate the use of different types of database drivers and their usage
- write program using different steps to connect a database
- use JDBC to access and modify database
- develop and deploy database application

4.2 JDBC: An Introduction

Most of the web applications use database. Database accessing is played a significant role in web development. In this unit, you will learn about how data can be stored, retrieved and manipulated using databases on a web application. A Java API (Application Programming Interface) that enables JSP program to execute SQL statements is called JDBC.

JDBC is a Java enabled technology that specifically designed for database connectivity. This technology provides methods for querying and updating data in a database.

Sun Microsystems released first official JDBC API 1.0 as part of JDK 1.1 in 1997. When JDBC 2.0 comes with enhancements to the JDBC core API, it contains new features such as scrollable ResultSets (you can move record pointer both forward and backward) and batch updates (you can submit multiple DML statements at a once like insert, update). In 2001, JDBC 3.0 API has been launched with the features such as reusability of prepared statements by connection pools, passing parameter to Callable Statement and added a new data type i.e. `java.sql.BOOLEAN`.

The Latest version of JDBC is JDBC 4.1 and Features of JDBC 4.1 is same as JDBC 4.0. The main feature of JDBC 4.1 is to autoloading of JDBC drivers. In earlier versions of JDBC, applications had to manually register drivers before requesting Connections. With JDBC 4.0, applications no longer need to issue a `Class.forName()` on the driver name; instead, the `DriverManager` will find an appropriate JDBC driver when the application requests a Connection. JDBC 4.0 Standard Extension API is defined in the `javax.sql` package. It is required for applications that uses connection pooling, distributed transactions, Java Naming and Directory Interface (JNDI) and RowSet API.

JNDI is standard used for looking up distributed components across the network. A J2EE application client may use the JNDI API to look up enterprise beans resources (database) and environment entries. A RowSet object is a java bean component and extends the ResultSet interface. It has a set of JavaBeans properties and follows the JavaBeans event model. A RowSet object's properties allow it to establish its own database connection and to execute its own query in order to fill itself with data.

The core API is defined in `java.sql` package and it is enough for normal database applications. The following section introduces JDBC and then explores how to use it in JSP programs.

4.2.1 Need for JDBC

Today, the most widely used interface to access database is ODBC. ODBC stands for Open Database Connectivity. It is a standard in relational database connectivity published by Microsoft (though originally developed jointly by Microsoft and Sybase). ODBC cannot be used directly with Java program because it uses a Programming Language C interface and it also makes use of Pointers which have been removed from Java. JDBC comes after the ODBC but JDBC is a Java API and contains Java interface for working with SQL.

4.2.2 Two- and Three-Tier Database Access Models

The JDBC provides support for two- and three-tier database access models. Please note that these access models are defined for general applications. In two-tier model, java application is directly connected to the database. This is done through the use of JDBC driver. The JDBC driver sends commands directly to the database and result of these commands are sent back directly to application. The following figure shows the two-tier model.

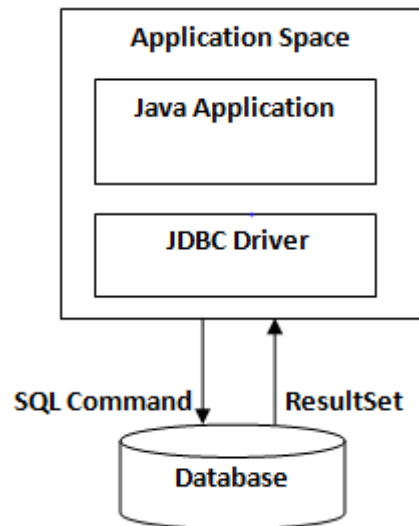


Figure 1: The two-tier Database Access Model.

In three-tier JDBC model, Java application can not connect directly to the database. A middle-tier comes between the Java application and database. When you use three-tier model, JDBC driver of Java application sends commands to middle-tier, which in turn sends commands to the database. The result of these commands is sent back from the database to middle-tier and middle-tier sent back to the application. The following figure shows the three-tier model.

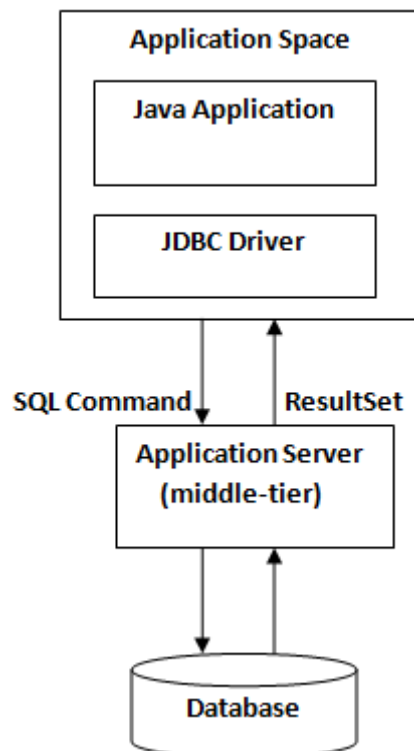


Figure 2 : The three-tier Database Access Model.

4.3 JDBC API

In this section, you will learn about the JDBC API. The JDBC provides a set of classes and interfaces that can be used to write database applications. The JDBC API (Application Programming Interface) is implemented by JDBC drivers. A JDBC driver is a software component that enables a Java application to interact with a database. The JDBC core API is found in the `java.sql` package and additional optional extensions for JDBC are found in `javax.sql` package such as connection pooling, rowsets and distributed transactions. The most important members of JDBC API are as follows:

- `DriverManager` class
- `Connection` interface
- `Statement` interface
- `PreparedStatement` and `CallableStatement` Interface
- `ResultSet` interface

The DriverManager Class

The task of `DriverManager` class is to maintain a list of JDBC drivers. This list contains information about the object reference of drivers and subprotocols that it supports. When a program requests a database connection with `getConnection()` method call, the `DriverManager` goes through the list and match a suitable driver. Each JDBC driver must be registered with the `DriverManager`. The JDBC drivers are provided by the database vendor or third party. You can use different JDBC drivers for different database servers.

The most important method of `DriverManager` class is `getConnection()` that returns a `java.sql.Connection` object. The signatures of this method are as follows:

```
public static Connection getConnection(String url)
public static Connection getConnection(String url, Properties info)
public static Connection getConnection(String url, String user, String password)
```

You can write connection string like the following way:

```
Connection con = DriverManager.getConnection(dbURL);
```

The `dbURL` is specified in the following manner:

`protocol:subprotocol:other parameter`

For example:

```
jdbc:microsoft:sqlserver://127.0.0.1:1433;user=dbusername; password=dbpwdname;
DatabaseName=dbname;
```

The protocol should be `jdbc`. The subprotocol represents database type. This may be Microsoft SQL Server, Oracle, MySQL, `jdbdc:odbc` and so forth. In the above case, it is Microsoft SQL Server. In other parameter category, IP address of the system, port number such as 1433 for Microsoft SQL Server and name, password, username of the database. The above definition is depends on the type of JDBC driver and database vendor.

The Connection Interface

This interface declares the methods that can be used to create a connection for particular database. An instance of the connection interface is obtained from `getConnection()` method of

the DriverManager class. After creating the connection with database, you can execute SQL statements for that particular connection and retrieve the results.

Some of the methods are listed below:

createStatement() method

The createStatement() method is used to create a statement object for sending SQL statements to the database. If you are using the same SQL statement in your application many times, it is more efficient and suitable to use a PreparedStatement object. Its signature of createStatement() method is as follows:

```
public Statement createStatement() throws SQLException
```

prepareStatement() method

This method is used to create a PreparedStatement object. Its signature is as follows:

```
public PreparedStatement prepareStatement() throws SQLException
```

close() method

The close() method is used to immediately close and release a Connection object. Its signature is as follows:

```
public void close() throws SQLException
```

The Statement Interface

The statement interface provides methods for executing SQL statements and obtains the results that are produced. The following is the two important method of Statement interface i.e. executeQuery() and executeUpdate(). Both are used for execution of SQL queries. The signature for both methods is as follows:

```
public int executeUpdate(String sql) throws SQLException  
public ResultSet executeQuery(String sql) throws SQLException
```

executeUpdate() method

The executeUpdate() method is used to executes an insert, delete, update SQL statements and also DDL statements to create, drop and alter tables. It returns the row count for insert, update, delete statement and returns zero(0) for SQL statement that return nothing.

executeQuery() method

The executeQuery() method is used to executes an SQL select statement that returns a single ResultSet object which contains data.

execute() method

execute() method is used to execute stored procedure.

Prepared and Callable Statement Interface

Prepared Statement Interface

When you are used same SQL statements over and over again changing only parameter values, it is best way to use prepared statement i.e. `java.sql.PreparedStatement`. The `PreparedStatement` differ from simple `Statement` in that it specified fill-in-blanks SQL template.

For example, “insert into tablename values (?, ?)”

There is a two unknown parameter in the form of question marks which are set at the run time by your application using `setXXX()` method such as `setString()`, `setInt()`. The `PreparedStatement` is a precompiled statement and has a query plan generated for it once. It is mainly used to speed up the process of insertion, deletion and updation especially when there is a bulk processing. When the program sends SQL queries to database engine, the query is parsed, compiled and optimized, the outcome of this process is called query plan.

For example:

```
PreparedStatement pstmt = con.prepareStatement("update product_table set productcode = ?  
where product_name like ?");
```

```
pstmt.setString(1, P001);  
pstmt.setString(2, "Washing Machine");  
ResultSet rs = pstmt.executeUpdate();
```

Here, ? represents a modifiable value in SQL statement. You can set the value of unknown parameter i.e. question mark with `setXXX()` method which specifies a parameter number and a value.

The Callable Statement Interface

A Callable statement is used for handling stored procedure. A stored procedure is one or more SQL statement stored as a one group in a compiled form inside a database engine. You can simply call the store procedure as you call a method call. It is an extension of `PreparedStatement`. The `CallableStatement`'s object is obtained from `Connection.prepareCall()` method and set the parameter using `setXXX()` methods. The `execute()` method is used to execute stored procedure.

For example:

```
CallableStatement cs = con.prepareCall("{call myProcedure}");  
ResultSet rs = cs.executeQuery();
```

The ResultSet Interface

The `ResultSet` interface represents an object that contains data in the form of row and column. The `ResultSet` object maintains record pointer that points to its current row of data. Initially, the cursor is positioned before the first row. By default, the record pointer of `ResultSet` object is forward only scrollable. You can traverse through the `ResultSet` using `next()` method only. The `next()` method returns false when the last record is reached and no more details can than be retrieved. The `java.sql.ResultSet` interface provides several methods for retrieving column

values with getXXX() method such as getString(), getInt() and getLong(). Some of the methods are defined in the section 4.6 of this unit.

Check Your Progress 1

1. Explain the term JDBC.

2. Why ODBC can not directly be used with Java programs?

3. What is the use of JDBC API?

4. Explain different types of statements in JDBC.

5. What are the different methods used in Statement interface?

6. What is stored procedure? How to call stored procedure using JDBC API?

7. How can you create JDBC Statement?

8. Write the names of important package used in JDBC.

9. Explain the importance of DriverManager class in JDBC.

10. Explain some methods of ResultSet object.

4.4 Types of JDBC Driver

In the previous section, you have learnt JDBC API. Now, you will learn different types of JDBC drivers. To connect with database, there is a need of JDBC driver that you use for different database servers. There are four types of JDBC driver in Java for data connectivity. These drivers are categorized from Type-1 to Type-4. In other words, you can say that the each database server such as Oracle, Microsoft SQL Server used these four types of drivers. These are as follows:

1. JDBC-ODBC Bridge, plus ODBC driver
2. Native API, Partly Java Driver
3. JDBC-net, Pure Java Driver
4. Native-Protocol, Pure Java Driver

Each of these types is described below:

Type 1: JDBC-ODBC Bridge, plus ODBC Driver

The first type of JDBC driver is the JDBC-ODBC Bridge. It is also called TYPE-1 driver. It is freely distributed with the JDK. This bridge driver works with Microsoft vendor such as Microsoft Access and Microsoft SQL Server. The driver translates JDBC method calls to ODBC function calls that are then passed to ODBC driver. The ODBC driver must be configured on the client machine for bridge to work. This type of driver is used for prototyping or when there is not any alternative option for JDBC driver. The JDBC-ODBC Bridge does not support multiple concurrent open statements per connection.

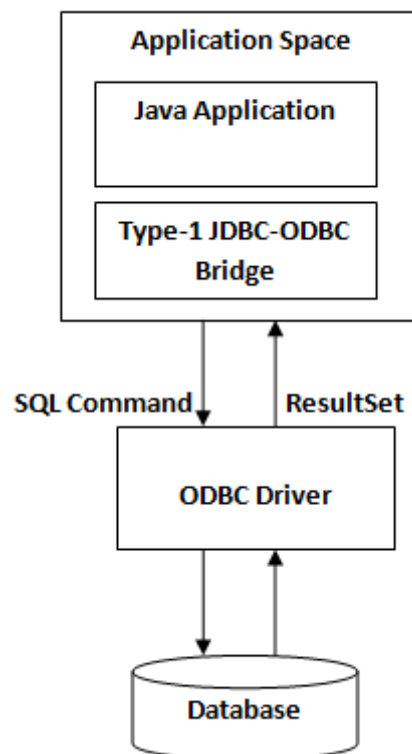


Figure 3: Type-1 JDBC-ODBC Bridge Driver

Type 2 : Native API, Partly Java Driver

The native-API driver, also known as Type-2 driver, converts JDBC commands into DBMS specific native calls. Like the Type-1 driver, this type of driver requires some binary code be loaded on each client machine that directly accesses the database.

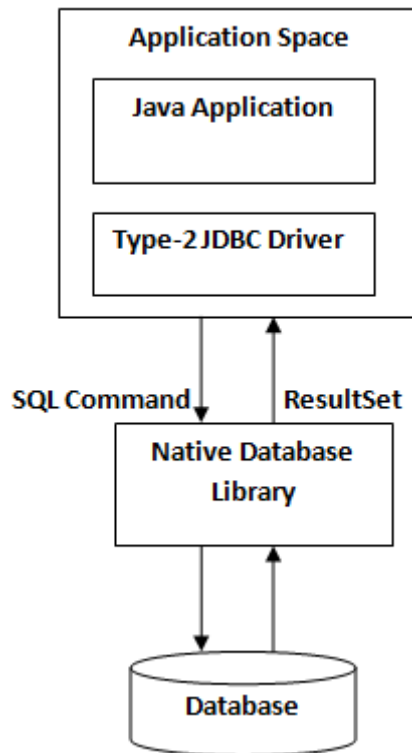


Figure 4: Type-2 Native-API Driver

Type 3 : JDBC-Net, Pure Java Driver

The JDBC Type-3 driver is a database driver which makes use of a middle-tier (application server) between the calling program and the database. It follows the three-tier communication

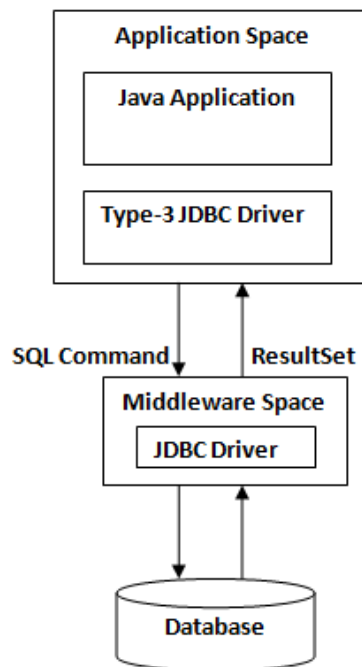


Figure 5: Type-3 JDBC-Net, Pure Java Driver

approach. This driver is ideal for Internet based applications. It translates JDBC calls on a client machine into RDBMS-independent network protocol (for example HTTP) and sends to a middle-tier server. The middle-tier server translates this RDBMS-independent network protocol into an RDBMS-specific protocol which is sent to a particular database. The results are routed through the middle-tier server and sent back to the client. This type of driver is more suitable for pure java client.

Type 4 : Native - Protocol, Pure Java Driver

The type-4 driver is also known as pure java driver. This driver's implementation is a two-tier approach because it communicates directly with a database. They do this by converting JDBC calls directly into a vendor specific database protocol. It is written completely in java and it is platform independent. This driver is install inside the JVM of the client. This type of driver has an advantage over all the other driver types because it has no additional translation or middle-tier server which improves performance of the JDBC driver. The following figure illustrates the working process of Type-4 driver:

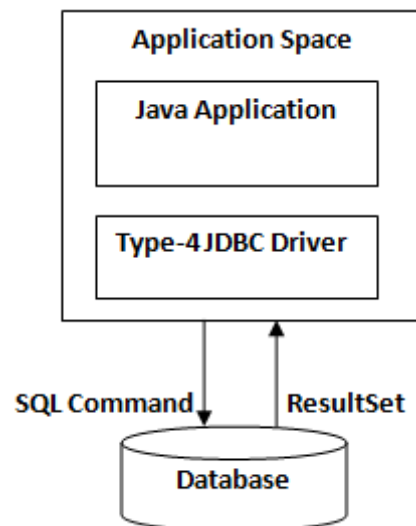


Figure 6: Type-4 Native-Protocol, Pure Java Driver

4.5 Connection steps using JDBC API

In this section, you will learn about access a database and manipulation of data. Before, you can access data from database; you need to connect to that database server. After you get the connection, you can communicate with database server using a SQL query. Once, the connection is established between client machine and database server, you can create, delete and update a table, invoke a stored procedure and you can do many more SQL query against the database.

To use JDBC, you must perform the following steps:

1. Load a JDBC database driver
2. Create a connection

3. Create a JDBC statement
4. Process the results
5. Close the connection to the database

Each of these steps will be discussed with smaller code fragment in the following sections:

Step 1 : Load a JDBC database driver

The first step is to load an appropriate JDBC driver. JDBC driver are available for most of the popular database today such as Microsoft SQL Server, Oracle and MySQL. Each database servers have their own language or syntax for communication in the form of JDBC driver. For this reason, each database server has their own jdbc driver. If you want to connect with a particular database server, you need to get the JDBC driver for that database.

As you have studied earlier, each database servers have four types of JDBC driver. Now, you can choose any one driver from the four database drivers. The JDBC-ODBC driver is more popular among the users. If you are using any other type of driver, it should be installed on your system. The JDBC driver comes in the form of jar file. If you are using Tomcat or other web server, simply copy the jar file into WEB-INF/lib directory under your web application directory and define its pathname in your classpath.

For example, In Microsoft SQL server database, there are three jar files i.e. msutil.jar, msbase.jar and mssqlserver.jar used for database handling. Like the following, if you are using type-4 driver of Microsoft SQL Server database, path should be defined under the classpath environment variable.

```
C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\msutil.jar;  
C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\msbase.jar;  
C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\mssqlserver.jar;
```

To load the driver, you can use the following syntax:

```
Class.forName(drivename);
```

The forName() method of the class Class is used to load the named class.

```
//load the JDBC-ODBC bridge driver  
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
  
//load the SQL Server Type-4 driver  
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

Step 2 : Create a connection

When a JDBC driver is loaded onto memory, it is registered with java.sql.DriverManager class. The DriverManager class maintains a list of instances of JDBC drivers available to a given JVM at runtime. After you register a JDBC driver with DriverManager, you can use getConnection() method of this class to obtain Connection object. In JDBC, Connection is represented by java.sql.Connection interface.

The syntax is looks like the following:

```
Connection con = DriverManager.getConnection(dbURL);
```

Here are several examples for connecting to various databases:

```
//Example1: get the connection using JDBC-ODBC bridge driver
Connection con = DriverManager.getConnection("jdbc:odbc:DSN", "user", "pwd");

// Example 2: get the connection using SQL Server Type-4 driver
Connection con =
DriverManager.getConnection("jdbc:microsoft:sqlserver://127.0.0.1:1433;
user=dbusername; password=dbpwdname; DatabaseName=dbname");
```

In the example 1, the DSN stands for data source name, is the name which you gave in control panel->Administrative Tools ->ODBC while registering a database or table.

In example 2, 127.0.0.1 is an IP address, you can give IP address of your system and 1433 is port number for SQL server database.

Step 3: Create a JDBC statement

Now that a connection is established with a database, it's the time to interact with it by invoking the operation using SQL query on data contained within the database. A statement is used to send the SQL query to Database Management System. You can create a statement and execute the query. The Statement object is defined in java.sql.Statement interface. The Statement object is not instantiated directly rather it takes an instance of active connection object 'con' which is created in the earlier step. The code looks like:

```
Statement stmt = con.createStatement();
```

You can also use PreparedStatement or CallableStatement according to your requirement.

Step 4: Process the Result

In order to execute query, you have to obtain a ResultSet object and call the executeQuery() method to execute the query. You have to pass SQL query as an parameter to the executeQuery() method. The code fragment is as follows:

```
ResultSet rs = stmt.executeQuery("select * from tablename");
```

A ResultSet is a collection of rows and columns corresponding to the results of the query. A row is a record which consists of one or more columns and column indicates fieldnames. The next() method is used to obtain the row of data of ResultSet. To obtain the values of each column, you can use getXxx() method where xxx refers to specific java datatype such as int, float, string and more. These methods like getString() (to retrieve String fields), getInt() (to retrieve Integer fields) and getFloat() (to retrieve Float fields) are defined in the java.sql.ResultSet.

Like the following way, you can get all the data from the ResultSet.

```
while (rs.next())
{

// get the type_id, which is an int
out.println("Type Id=" + rs.getInt("type_id"));
//get type_name which is a string
out.println("Type Name =" + rs.getString("type_name"));

}
```

You can retrieve the values of columns by passing its column index or column name.

Step 5: Close the connection to the database

After successfully performing the above steps, you must close the Connection, Statement and Resultset object by calling the close() method on appropriate objects.

For example

```
rs.close();           // close Connection object
stmt.close();         // close Statement object
con.close();          // close connection object
```

Check Your Progress 2

1. How can you load the drivers?

2. How can you retrieve data from the ResultSet?

3. How can you make a connection using JDBC?

4. Which type of JDBC driver is the fastest one?

5. What are the main steps to make JDBC connection?

6. Assume that there is a table in MS-Access which contains field names such as Id, name, course and course_fee. Write a JSP program to display the records in mentioned table using JDBC type-1 driver.

7. What is ResultSet?

8. What is connection pooling? What is the advantage of connection pooling?

9. What is the latest version of JDBC and new features of it?

10. What is DSN?

4.6 Example Applications using JSP

This section will provide you in-depth knowledge of data access specifically insertion and retrieval of data to/from a database. Consider a table named as Student is created in MS SQL Server database with Roll No, Name and Programme. This table is used in both the following sections. This section defines example for storing/retrieving data into/from a Microsoft SQL Server using type-4 JDBC driver. You can run these programs on any web server such as Tomcat. For running these programs, you need a JDBC driver in .jar form and place them in lib directory under the your web application.

4.6.1 Using JDBC to Store data in Database

The example contains a JSP form for entering data and another program for data processing. In InputFrm.jsp program, there are three input fields such as roll no, student name and programme name. On the submission of this form, program named InsertData.jsp is called.

Source code for InputFrm.jsp :

```
<html><head><title>Using JDBC to store data in database</title>
</head><body>
  <form method="get" action="InsertData.jsp" >
  <h4>Enter values in following fields</h4>
  <table>
    <tr><td>Roll No</td>
    <td><input type="text" name="rno" value="" size=9> </td></tr>
    <tr><td>Name </td>
    <td><input type="text" name="sName" value="" size=15></td></tr>
    <tr><td>Programme</td>
    <td><input type="text" name="prg" value="" size=15> </td></tr>
    <tr><td colspan="2" align="center">
    <input type="submit" value="Submit Data"></td></tr>
  </table></form></body></html>
```

In the following source code, the first step to get the data from InputFrm.jsp program using request.getParameter() method. After connecting to database, an insert query is executed using executeUpdate() method. The program is written using try and catch clause of standard Java mechanism within JSP scriptlets.

Source code for InsertData.jsp :

```
<% @ page import="java.util.*" %>
<% @ page import="java.sql.*;" %>
<html>
<head><title>Insert data into database</title></head>
<body>
<h3>Using JDBC to Insert Data into Database</h3>
<table border=1>
<%
String rollNo = request.getParameter("rno");
String StuName = request.getParameter("sName");
String prgName = request.getParameter("prg");

Connection con = null; //create connection object
Statement stmt = null; // create statement object

// connection string using Type-4 Microsoft SQL Server driver
// you can also change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=bc53;
 DatabaseName=SOCIS";

try{
// load JDBC type-4 SQL Server driver
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
con = DriverManager.getConnection(url);

if (con != null)
{
    stmt = con.createStatement();

    //insert query
    String rsql ="insert into student
values("+rollNo+", "+StuName+", "+prgName+" "+"");

    //execute query
    stmt.executeUpdate(rsql);
    out.println("Your data is successfully stored in database");
}
if(con == null)

    { con.close(); // release connection }

} // end of try clause

catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
%>
```

The following screen is display after running the above programs:

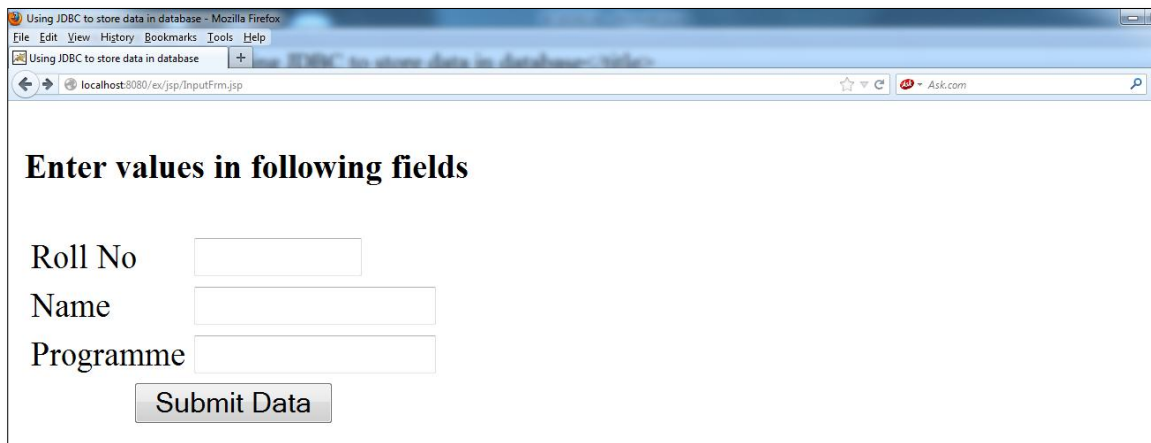
A screenshot of a Mozilla Firefox browser window. The title bar says "Using JDBC to store data in database - Mozilla Firefox". The address bar shows "localhost:8080/ex/jsp/InputFrm.jsp". The page content has the heading "Enter values in following fields". Below this, there are three input fields labeled "Roll No", "Name", and "Programme". Each field has a corresponding text box. Below the text boxes is a button labeled "Submit Data".

Figure 7: Input Form for storing data into database

In the above screen, when you will enter values then the following screen will show you a message for data storage.

A screenshot of a Mozilla Firefox browser window. The title bar says "Insert data to database - Mozilla Firefox". The address bar shows "localhost:8080/ex/jsp/InsertData.jsp?rno=989939956&stName=Akhilesh&prg=MCA". The page content has the heading "Using JDBC to Insert Data into Database". Below this, there is a message "Your data is successfully stored in database".

Figure 8: Data stored in persistent storage

4.6.2 Using JDBC to Retrieve Data from Database

The following example gives you an illustration about how to query a database. After execution of the above program, you have stored sufficient data into database. Now, you will execute the following code for retrieving the data. In this program, one additional ResultSet object is used for retrieving data from select query. The data is retrieved from ResultSet object using getXXX() method such as getInt() and getString(). Note that if the column name is an integer type then you should use getInt() method instead getString() method.

```

<% @ page import="java.util.*" %>
<% @ page import="java.sql.*;" %>
<html>
<head><title>Retrieved data from database</title></head>
<body>
<h3>Using JDBC to Query a Database</h3>
<table border=1>
<%
Connection con = null; //create connection object
Statement stmt = null; // create statement object
ResultSet rs = null;    // create ResultSet object

// connection string using Type-4 Microsoft SQL Server driver
// you can change the next line with your own environment
String url= "jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa; password=bcs53;
           DatabaseName=SOCIS";

try{
// load sql server JDBC type-4 driver
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

con = DriverManager.getConnection(url);

if (con != null) {
stmt = con.createStatement();

// select SQL statement
String rsql ="select * from Student";

//execute query
rs = stmt.executeQuery(rsql);
%>
<tr><td>Roll Number</td><td>Student Name</td><td>Programme</td></tr>
<%
while( rs.next() ){
%><tr>
<td><%= rs.getInt("RollNo") %></td>
<td><%= rs.getString("Student_Name") %></td>
<td><%= rs.getString("Programme") %></td>
</tr>
<%    }}
if(con == null) {con.close();}
}
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
%>

```

After running the above program, following output screen is displayed.

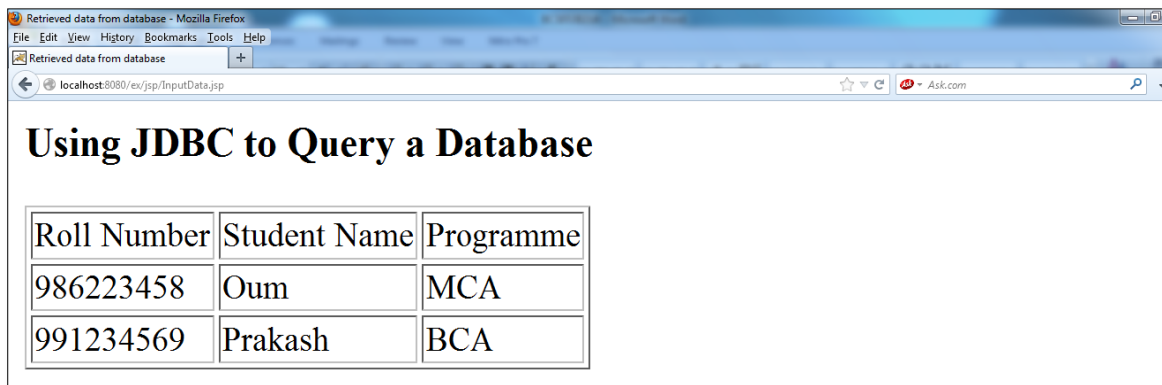


Figure 9: Display data from database

The majority of real projects are build using Oracle database as back end. For this, you can change only port no and Oracle JDBC driver name in above code. You can also refer sample JDBC Code for Connectivity with ORACLE in following link:

<http://www.csc.ncsu.edu/faculty/mpsingh/local/programming/sample-JDBC-code.html>.

4.7 Application Development and Deployment

Let us take an example to understand how to develop and deploy a database application using NetBeans. The detailed description of installation and creating a project in NetBeans IDE are described in lab session block BCSL057 of this course.

In Unit 3 of this block, you have seen a static session program in which user name and password is statically compared and shows the result. Now, this section gives you an illustration using the same program using database values. Let us take a tour of this program; a login page contains two input fields named user name and password. When you want to submit form without entering data values in input form fields, the program called a validate() function to display an error message otherwise it goes for normal processing On the submission of the form, a actionPage.jsp program is called which retrieved data from login page using request.getParameter() method and compared with the data stored in database. If the data is valid and found in database then set the session using session.setAttribute() method and control transfer to the nextPage.jsp program to display welcome message for user otherwise it display a error message for invalid user.

The two files are included in login page namely top.jsp and bottom.jsp using include directive and the default file is index.html which include a hyperlink as 'Student Login'. The code for index.html, top.jsp and bottom.jsp are same as the lab session block (BCSL057) program. This example is connected with MS Access database using Type-1 JDBC driver. For creating system DSN(data source name), you go through the Control Panel\All Control Panel Items\Administrative Tools\ODBC\system DSN and follow some steps and create DSN name for your application.

Source code for LoginPage.jsp:

```
<% @page contentType="text/html" pageEncoding="UTF-8"%>
<% @include file= "top.jsp"%>
<!DOCTYPE html>
<html> <head>

<script type="text/javascript">
function validate()
{
uid=document.form1.uid.value;
pwd=document.form1.pwd.value;

if(uid==" " || uid==null)
{
alert("Please Enter Your User ID");
document.form1.uid.focus();
return false;
}
if(pwd==" " || pwd==null)
{
alert("Please Enter Your Password");
document.form1.pwd.focus();
return false;
}
return true;
}
</script>
<title>Login Page in JSP</title>
</head>
<body onload="document.form1.uid.focus()">
<h3>LOGIN PAGE IN JSP</h3>
<form name="form1" method="post" action="actionPage.jsp"
onsubmit="return validate()">

<table><tr><td><b>User ID</b></td>
<td><input name="uid" type="text"/></td></tr>
<tr><td><b>Password</b></td>
<td><input name="pwd" type="password"/></td></tr>
<tr><td><input type="submit" value="Submit" /> </td>
<td><input type="reset" value="Reset" /></td></tr>
</table> </form><% @include file= "bottom.jsp"%>
</body></html>
```

Source code for actionPage.jsp :


```

<% @page contentType="text/html" pageEncoding="UTF-8"%>
<% @ page import="java.sql.*;" %>
<!DOCTYPE html>
<html> <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Login Action Page</title>
</head><body>

<%
String cid=request.getParameter("uid");
String pass=request.getParameter("pwd");

    Connection con = null; // create connection object
    Statement stmt = null; // create statement object
    ResultSet rs = null;    // create ResultSet object

// load JDBC-ODBC bridge Type-1
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String db = "jdbc:odbc:db1"; //db1 is DSN
con = DriverManager.getConnection (db, "", "");
stmt=con.createStatement();

rs=stmt.executeQuery("select * from Login where userID='"+cid+"' and
Password='"+pass+"' ");
if(rs.next())
{
    session.setAttribute("scid",cid);
    con.close();
    response.sendRedirect("nextPage.jsp");
}
else
{
    %>
<h1>Invalid UserID or Password</h1>
<jsp:include page="LoginPage.jsp" />
<%
}
%></body></html>

```

Source code for nextPage.jsp:

```

<% @page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<% @ page import="java.sql.*" %>
<% @include file= "top.jsp"%>
<%! String scid=""; %>
<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>Login Page</title>
</head><body>
<%
scid=(String)session.getAttribute("scid");
%>
<div id="welcome">
<h2><span>Welcome User:::<strong><font color='Blue'>
<%= scid %></font></strong></span></h2></div>
<% @include file= "bottom.jsp"%> </body></html>

```

When you will run the index page from Netbeans IDE then the following output screen will display:

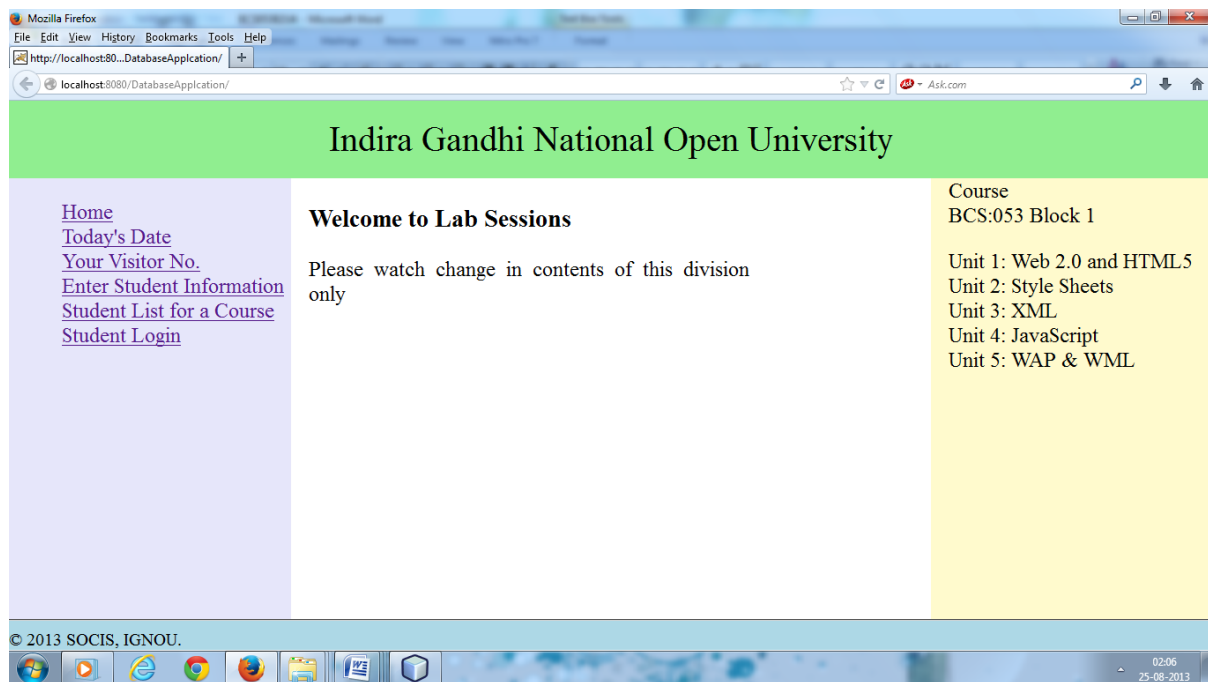


Figure 10: Display an Index page

When you will click on student Login hyperlink, the login page is display in the browser.



Figure 11: Display a Login page

When you submit a blank form field, it shows an error message. Looks like the following screen:

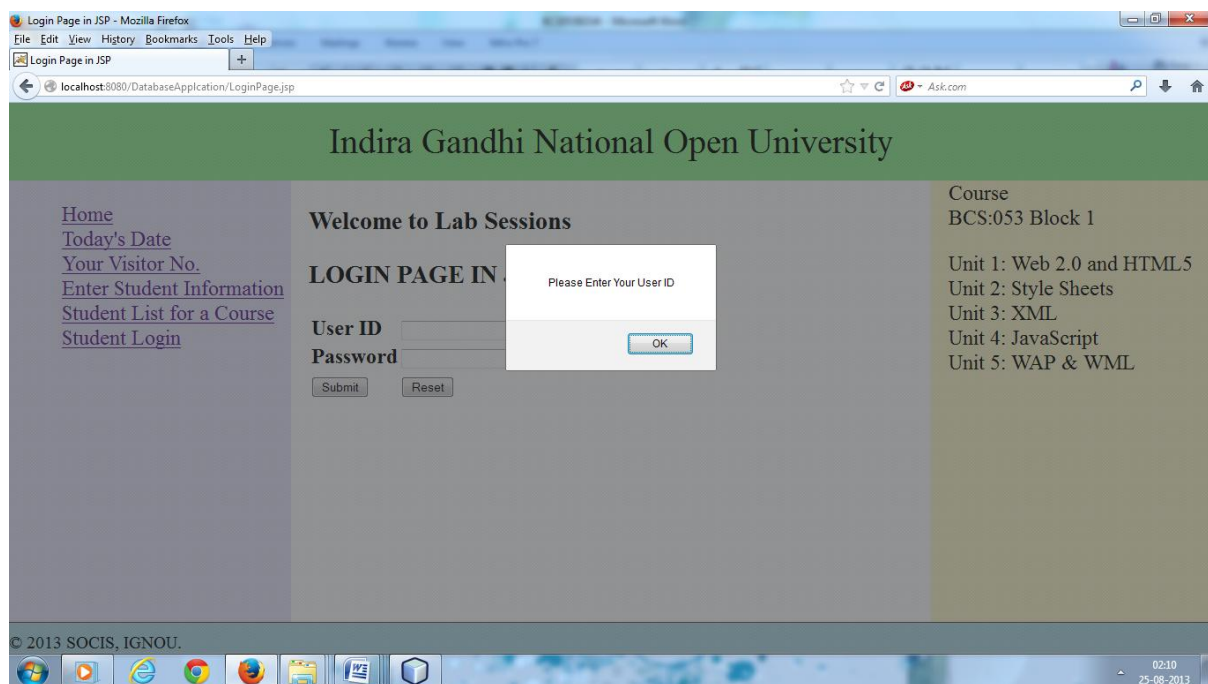


Figure 12: Display an error message

After successfully validated user, it shows welcome message for user and comes with the following screen:

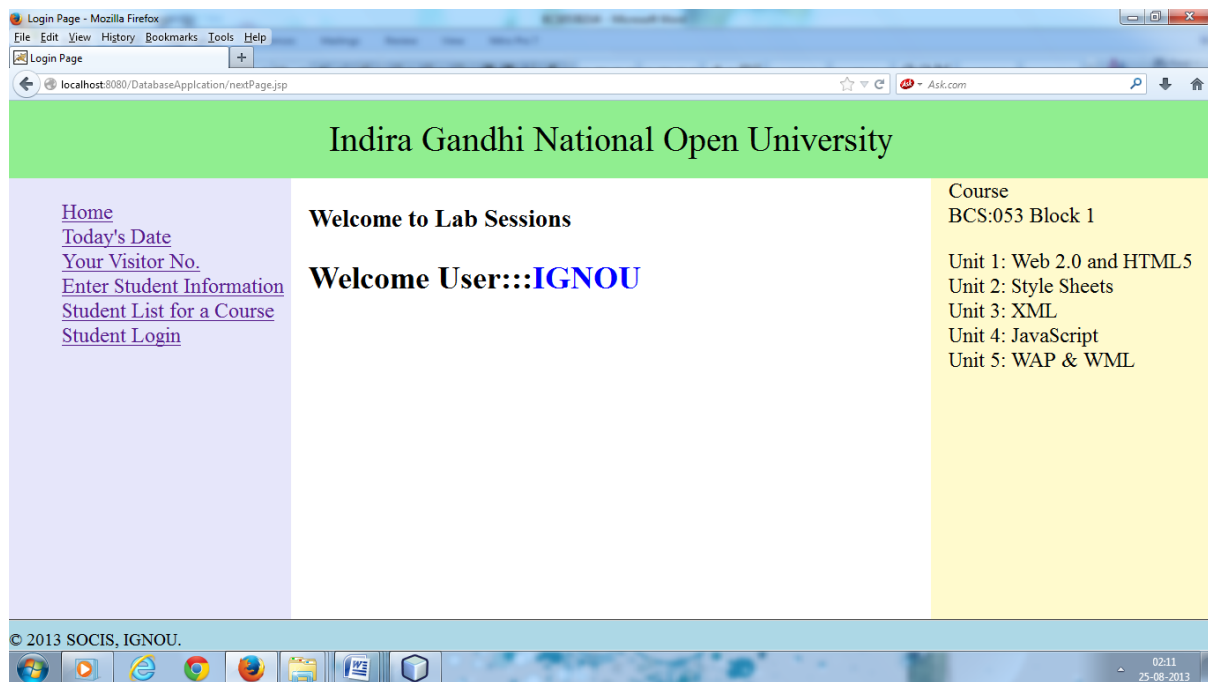


Figure 13: Display welcome message for valid user

Now, you are able to create a web application with database connectivity. Here's some introductory information related to JSP Standard Tag Library (JSTL) SQL tags which are used for data access.

JSTL SQL tag library

JSTL SQL tag library provides various tags to access the database. To use these tags in JSP, you should have used the following taglib:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

There are two important JSTL SQL tag such as `<sql:setDataSource>` and `<sql:query>` which are described below:

JSTL `<sql:setDataSource>` tag

This tag is used for creating a DataSource configuration that may be stored into a variable which can be used as an input to another JSTL database-action. You can use this tag like the following:

```
<sql:setDataSource var="student" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/student" user="netbean"
    password="netbean7"/>
```

The `<sql:setDataSource>` tag has following attributes:

- **var** : This attribute is optional. It specifies the name of the variable which stores a value of the specified data source.
- **dataSource** : This attribute is optional that specifies the data source.
- **driver**: This optional attribute defines the JDBC driver class name.

- **url** : This optional attribute specifies the JDBC URL associated with the database.
- **user** : This is an optional attribute that specifies the JDBC database user.
- **password** : This optional attribute specifies the JDBC user password.

JSTL <sql:query> tag

This tag is used for executing the SQL query written into its body or through the sql attribute. For example:

```
<sql:query dataSource="${student}" var="ProgData">
  SELECT ProgCode, Pname from programme;
</sql:query>
```

The <sql:query> tag has following attributes:

- **var**: This is a required attribute that specifies the name of the variable to stores a value of the query result.
- **scope**: This optional attribute specifies the scope of a variable to display a result from the database. By default, it is page.
- **sql**: This attribute specifies the statement of SQL query to execute.
- **dataSource**: This is an optional attribute specifies the data source.
- **startRow**: This optional attribute that includes the specified starting index.
- **maxRows**: This optional attribute specifies the maximum number of rows.

You will find examples of the JSTL SQL tag in the lab session block (BCSL057).

4.8 Summary

Database access is one of the important features of Web application. Java has its own technology for database connectivity is called JDBC. JDBC provides a standard library for accessing wide range of relational databases like MS-Access, Oracle and Microsoft SQL Server. Using JDBC API, you can access a wide variety of different SQL databases. The core functionality of JDBC is found in java.sql package. In this unit, you have seen many members of this package and learned how to use them. This unit has also introduced to you about the different types of drivers.

To connect a database, you should follow certain steps. The first step is to load an appropriate driver from the type-1 driver to type-4 driver for specific database. The second step is to create a connection to the database using getConnection() method of DriverManager class. Third step is to create a JDBC statement by using createStatement() method of connection object. You can also use PreparedStatement and CallableStatement in place of simple JDBC Statement but all three have different role in JDBC. Statement interface provides different method used to execute query i.e executeUpdate() and executeQuery(). When a SQL query is used many times then PreparedStatement is used whereas CallableStatement is used to execute stored procedure. After creation of statement, the next step is to process result and store result in ResultSet object. To navigate the ResultSet object, you can use next() method and getxxx() method to retrieve fields values. Last step is to close all the connection, statement and ResultSet object. This is the way to query or modify a database records.

A couple of examples have been created in this unit that illustrate various database operations like storage and retrieval of records to/from database. Now, you have much more knowledge about the different JDBC drivers and database servers. This unit defines the examples for database application using Type-1 JDBC driver for MS Access database and Type-4 driver for MS SQL Server database. In the lab session block of this course, you will also get example using MySql database.

4.9 Solutions/Answers

Check Your Progress 1

Ans1.

JDBC is java database connectivity as name implies it's a JDBC API for communicating to relational database, API has java classes and interfaces using that you can easily interact with database. For this you need database specific JDBC drivers.

Ans2.

Open Database Connectivity (ODBC) cannot be used directly with java programs due to the following reasons:

- ODBC uses a C interface. This will have drawbacks in the security and robustness.
- ODBC makes of Pointers which has been not used in Java.
- ODBC driver creates a complex connecting procedure.

Ans3.

The JDBC API defines the Java interfaces and classes. It is used to connect to the database and send SQL queries. A JDBC driver implements these classes and interfaces for a specific database vendor.

Ans4.

There are three types of statements in JDBC API i.e. Statement, PreparedStatement and CallableStatement.

The Statement object is used to send SQL query to database and get result from database, and get statement object from connection object.

Statement: It is used for getting data from database useful when you are using static SQL statement at runtime. It will not accept any parameter.

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery();
```

PreparedStatement: when you want to use same SQL statement many times using different parameter then it is useful and it will accept parameter at runtime. It is precompiled statement.

```
String sql = "update course_table set fee = ? where courseCode = ?";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, 5000);
```

```
pstmt.setString(2, "BCS053");  
ResultSet rs = pstmt.executeQuery();
```

CallableStatement: When you want to access stored procedures then CallableStatement is useful and they also accept runtime parameter. It is called like the following way:

```
CallableStatement cs = con.prepareCall("{call store_procedure_name}");  
ResultSet rs = cs.executeQuery();
```

Ans5.

Stored procedure is a group of SQL statements that written inside the database engine. Stored Procedures are used to encapsulate a set of operations or queries to execute on database. Stored procedures can be compiled and executed with different parameters and results and may have any combination of input/output parameters. Stored procedures can be called using CallableStatement class in JDBC API. Below code snippet shows how this can be achieved.

```
CallableStatement cs = con.prepareCall("{call store_procedure_name}");  
ResultSet rs = cs.executeQuery();
```

Ans6.

The Statement interface provides different methods for execution of the SQL statements. These are as follows:

- executeUpdate() - This method is used to executes an insert, delete, update SQL statements and also DDL statements to create, drop and alter tables. It returns the row count for insert, update, delete statement and returns zero(0) for SQL statement that return nothing.
- executeQuery() - The executeQuery() method is used to executes an SQL select statement that returns a single ResultSet object which contains data.
- execute() - The execute() method is used to execute stored procedure.

Ans7.

You use the statement interface method to execute SQL queries and obtain results. You can create a statement object and then execute query. The statement interface provides some methods for executing SQL statements. For manipulation of data, executeUpdate() method is used and executeQuery() method is used for retrieval of data from persistent storage. You can send these queries using createStatement() method of Connection interface.

```
Statement stmt = con.createStatement();
```

Ans 8.

The most important package used in JDBC is java.sql package. The JDBC core API is found in the java.sql package and additional optional extensions for JDBC are found in javax.sql package such as connection pooling, rowsets and distributed transactions.

Ans 9.

The DriverManager class provides static methods for managing JDBC drivers. The most important method of this class is getConnection() that return java.sql.Connection object. The DriverManager class maintains a list of JDBC drivers. This list contains information about the object reference of drivers and subprotocols that it supports.

Ans 10.

The ResultSet interface provides some methods which are used for movement of record pointer such as isFirst(), isLast(), next() and obtain the values of each column with getXXX() method where xxx refers to specific java datatype such as int, float, String.

Check your Progress 2**Ans 1.**

For loading a JDBC driver, you can use Class.forName() method. If you want to use JDBC-ODBC bridge driver, the following code will load it.

```
Class.forName("sin.jdbc.odbc.JdbcOdbcDriver");
```

Ans2.

JDBC provides ResultSet interface that contains rows you retrieved from database, so you need an instance of ResultSet to hold the results. To create ResultSet object, you use the following code fragment:

```
ResultSet rs = stmt.executeQuery("select emp_name, salary from emp_table");
String s = rs.getString("emp_name");
```

Ans3.

For establishing a connection, you need to have appropriate JDBC driver. If you use type-1 JDBC driver then create a data source name (DSN) using ODBC and use the DSN name in connection string. Like the following way, you can create a connection

```
Connection con = DriverManager.getConnection(jdbc:odbc:datasourcename);
```

Ans4.

Type-4 JDBC Native Protocol Pure Java driver is the fastest driver among the drivers because it converts the JDBC calls to vendor specific commands and directly interacts with database. It follows the two-tier database access model.

Ans5.

The main steps used to connect to database are as follows:

- **Load the Driver:** First step is to load the database specific driver which communicates with database using Class.forName() method.
- **Create Connection:** Next step is to create a connection with database using getConnection() method of Connection interface.
- **Get Statement object:** Using createStatement() method of Connection interface, you can get statement object which is used to send SQL statement to the database.
- **Process the result:** Using statement object, you execute the SQL or database query and get result set from the query.
- **Close the connection:** After getting ResultSet and all required operation performed the last step should be closing the database connection.

Ans 6.

The source code is as follows:


```

<% @ page import="java.util.*" %>
<% @ page import="java.sql.*;" %>
<html>
<head><title>Retrieved data from database</title></head>
<body>
<h3>Using JDBC to Query a Database</h3>
<table border=1>
<%
Connection con = null; //create connection object
Statement stmt = null; // create statement object
ResultSet rs = null;    // create resultset object

// connection string using Type-1 Microsoft Access driver

try{
// load sql server JDBC type-1 driver
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String db = "jdbc:odbc:db1";
con = DriverManager.getConnection (db, "", "");

if (con != null) { stmt = con.createStatement();
String rsql ="select * from Student";
rs = stmt.executeQuery(rsql);
%>
<tr><td>ID</td><td>Student Name</td><td>Course</td><td>Course
fee</td></tr>
<%
while( rs.next() ){
%><tr>
<td><%= rs.getInt("ID") %></td>
<td><%= rs.getString("studentname") %></td>
<td><%= rs.getString("Course") %></td>
<td><%= rs.getString("CourseFee") %></td>
</tr>
<%
}}
if(con == null) {con.close();}
}
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
%>

```

Ans 7.

ResultSet object is used to store result after executing a SQL statement in the form of rows and columns. To navigate the rows of the ResultSet, the next() method is used and obtain the values of each column with a getxxx() method where xxx refers to a specific Java datatype such as Int, Float and String.

Ans 8.

The connection pooling is a technique to reuse the created connection. Connection pooling is provided the faster execution of program to reuse the connection without making new connection. This feature is found in `javax.sql` package. To use the connection pooling, you need to connect to the data source using `javax.sql.DataSource` interface. Using the connection pooling, you can manage number of connection with your application.

Ans 9.

The Latest version of JDBC is JDBC 4.1 and Features of JDBC 4.1 is same as JDBC 4.0. The main feature of JDBC 4.1 is to autoloading of JDBC drivers. In earlier versions of JDBC, applications had to manually register drivers before requesting Connections. With JDBC 4.0, applications no longer need to issue a `Class.forName()` on the driver name; instead, the `DriverManager` will find an appropriate JDBC driver when the application requests a Connection.

Ans 10.

Data Source Name is a name given to the database to identify it in the java program. The DSN name is linked with the actual location of database. It is used with JDBC-ODBC bridge type-1 driver to connect database.

4.10 Further Readings

- Professional JSP..... Brown-Burdick, Apress.
- Java for the web with JSP... Techmedia
- Pure Java 2 ...Sams, Techmedia
- Java for the web with Servlets, JSP...Budi Kurniawan, Techmedia
- Pure JSP....Sams, Techmedia
- Java Database Programming, prateek Patel, Corollis
- list of drivers is available at : <http://www.devx.com/tips/Tip/28818>
- <http://docs.oracle.com/javase/1.3/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html>