Input-Output on World Wide Web

Unit-III

The two most common request types of HTTP are get & post. A get request typically gets or retrieves information from a server. Common uses of get requests are to retrieve an HTML document or an image or to fetch search results based on a user submitted search term.

A Post request typically posts (or sends) data to a server. Post method is used to send information to a server, such as authentication information or data from a form that gathers user input.

An HTTP request often posts data to a server-side form handler that processes the data.

A get request sends information to the server as part of the URL (eg. www.search-engine.com/search?name=value), where 'search' is the name of a server-side form handler, name is the name of a variable in HTML form & value is the value assigned to that variable. A ? in URL separates the Query String from the rest of the URL in a get request. A name/value pair is passed to the server with the name & the value separated by an equal (=) sign. If more than one name/value pair is submitted, each pair is separated by an ampersand (&). The server uses data passed in query string to retrieve an appropriate resource from the server. The server then sends a response to the client.

Yes or No

Scanned by CamScanner

A get request may be initiated by submi an HTML form whose method attribute is se to "get".

Eg. < form method = "get" action = "search" >
< input type = "text" name = "search term" >

A search for the term "Massachusetts" might send a get req. for the URL www.searchengine.com/search? searchterm = Massachusetts. The server then search for value Massachusetts & return a web page containing search results for the user.

Post : A post req. is specified in an HTML form by the method "post". The post method sends form data as an HTTP message, not as a part of URL. Because a get request limits the query string to 2048 characters, it is often necessary to send large amout of info. using the post method. The post method is also preferred bcoz it hides the submitted data from the user by embedding it an HTTP message. Browsers typically do not cache the server's response to a post request, bcoz the next post might not return the same ~~page~~ result. eg. In a survey, many users could ~~not~~ visit the same web page & respond to a question. The servey results could then be displayed for the user. Each new response changes the overall results of the survey.

Scanned by CamScanner

XML

XML stands for eXtensible Markup Language. XML was design to add new facilities to existing web technologies. XML is a hierarchical data structure stored in text format files, using a markup language.

XML has following two main reasons for development :

1) Computers don't understand the info. stored in them. eg. there is no way for a search engine or any other computer, to know that this is page contains the introduction part of an XML. All it is a collection of letters & numbers, with HTML formatting around it. If a page or document is written in XML, a computer can understand exactly what it is about.

2) Web-Pages are not compatible across diff. devices. People are now accessing the web pages from a variety of different devices i.e. PCs, Macs, mobile phones, palmtop computers & even televisions. Bcoz of this, web designers must now either produce their pages in several diff. formats to cope with this. XML is used to define what data means & not how it is displayed, it makes it very easy to use the same data on several diff. platforms.

XML is not a way to design your web page. XML is to describe data.

HTML is used to describe how data is formatted.
XML is used to describe what data actually means.

Scanned by CamScanner

(A) XML tags : The tags used in XML, is similiar in cons to HTML. XML tags are case-sensitive. XML tags are not pre-defined like HTML ones are. XML tags can be nested.

B) XML Correctness : XML is that it should be independen of the platform it is running on. As XML does not actually do anything it is up to software developers to make s/w to use this data on a particular platform. If XML has not written in proper syntax, the error will be displayed.

c) Declaring XML : The correct way how to declare an XML document

<? xml version="1.0" ?>

This tells whatever software receives this data that you are writing XML and that it should match the specification for version 1.0.

Two optional attributes can be used in this line

1) encoding : specifies the character set used in the document, possible values are "UTF-8", "UTF-16" etc.

2) standalone attribute is optional and specifies whether the XML document can be completely validated using DTD rules contained in document. Possible values are Yes & No.

4) XML Document must have a Root element. Root element is the parent of all other elements.

Scanned by CamScanner

<antages Of XML

1. XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout & display & be sure that change in the underlying data will not require any changes to HTML.

2) XML simplifies Data Sharing

In the real world, computer systems & databases contain data in incompatible formats. XML is stored in plain text. This provides a S/w and H/w independent way of storing data.

3) XML simplifies Data Transport

Exchanging data b/w incompatible systems over the Internet with XML reduces the complexity. XML is stored in plain text so it makes it easier to expand or upgrade to new O/S, new applications or new browsers without losing data.

T

Yes or No

Scanned by CamScanner

Scrollbar specified is either yes or no. It will either appear or not appear depending on the value taken by this parameter.

Scanned by CamScanner

we have created a number of style sheet commands similar to the one shown above, and have stored them in a file test.css. Then, we can give a reference of this file in our HTML file as:

<LINK =stylesheet HREF="http://www.your.page/test.css" TYPE="text/css">

Now, when a browser receives an HTML page from a Web server, it must first consult the style sheet file and use the definitions from there to display the various tags. Thus, no matter which browser (or version) the user has, the output would always be consistent.

COMMON GATEWAY INTERFACE (CGI)

The Common Gateway Interface (CGI) is the earliest and one of the most popular server-side programming techniques. CGI makes dynamic generation of Web pages possible by executing a program on the Web server. Usually, a user fills an HTML form and when the user presses a *submit* button provided on that form, a CGI program gets invoked on the server. We have already discussed this, and shall not repeat it here.

CGI was the first attempt to make Web pages dynamic. CGI is a standard type of program on a UNIX computer, and this idea is now ported on Windows platform as well.

However, there is one problem with CGI, which is that for each client requesting a CGI Web page, a new process has to be created by the operating system running on the server. That is, the Web server must request the operating system to start a new process in memory, allocate all resources such as stack for it and schedule it, etc. This takes a lot of server resources and processing time, especially when multiple clients request the same CGI Web page (i.e. the page containing the CGI program). The operating system has to queue all these processes, allocate memory to them and schedule them. This is a large overhead.

This is shown in Figure 9.19. Here, three different clients are shown to request for the same CGI Web page (named CG-1). However, the Web server sends a request to create a different process for each of them to the operating system.

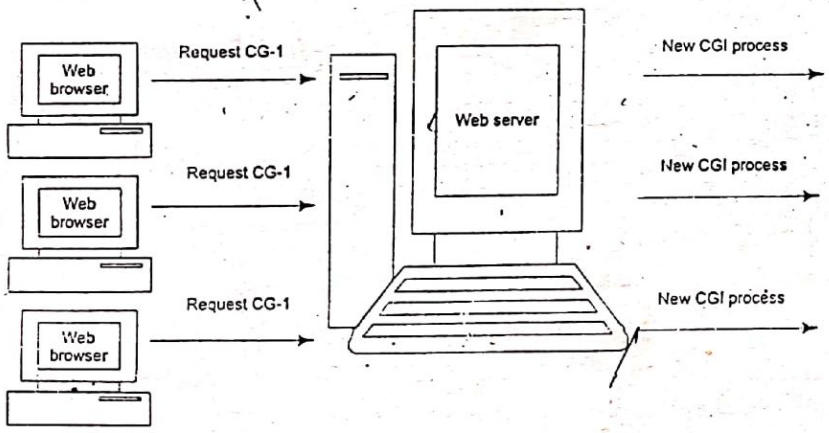


Fig. 9.19 Each CGI Request Results into a New Process Creation

Interestingly, CGI is only an idea, a technique. It is not a programming language. It simply specifies how to make a Web page dynamic. The programs for CGI (i.e. CGI scripts) can be written in many programming languages including C, FORTRAN, Unix shell scripts, etc. However, by far, PERL (Program Extraction and Reporting Language) is the most popular language used for writing CGI scripts.

These days, CGI is not the preferred choice of server-side Web development. Simpler technologies such as ASP, JSP and servlets are getting more popular. To counter them, a new standard called fast CGI has come up, which removes the drawback of spawning a new process for every request. However, we shall not elaborate it further, as it is not so popular.

MICROSOFT'S ACTIVE SERVER PAGES (ASP)

Microsoft's Active Server Pages (ASP) combines all the good features of previous attempts made to make Web pages interactive. It allows easy programming, database access, operating system access and speed. ASP is server-side scripting technology that makes Web pages dynamic.

Unlike CGI, an ASP program is dynamically loaded in the memory of the server only when called for the first time (in Microsoft terms, this is known as a Dynamic Link Library or DLL). This means that only for the first time, a new process is created for an ASP request. For all further requests, a separate thread within the process is created. This reduces the number of processes to be managed (created, scheduled, deleted, etc). This saves valuable resources. In the case of CGI, for every page request (even if the same page is being called more than once at the same time or different times), a new process is created, as we have already discussed.

The Web browser (client) requests for a Web page as usual. This is shown in step 1 of Figure 9.20. The source code for an ASP file is an ASCII text file, just like a standard HTML page as shown in step 2. The file has an .asp extension to tell the Web server that it is an ASP file, just like a CGI file that has a .cgi extension. As a result, before responding to the client's request, the Web server passes the file to the ASP interpreter. The ASP interpreter executes the code in the ASP file (interprets it) line-by-line, translates the results as necessary in the HTML form as shown in step 3 and gives the output back to the Web server as shown in step 4. The Web server then sends the plain HTML page back to the client as shown in step 5. Of course, that can include client-side scripts in the form of JavaScript or VBScript as discussed earlier (for local validations, processing, etc). This is shown in Figure 9.20.

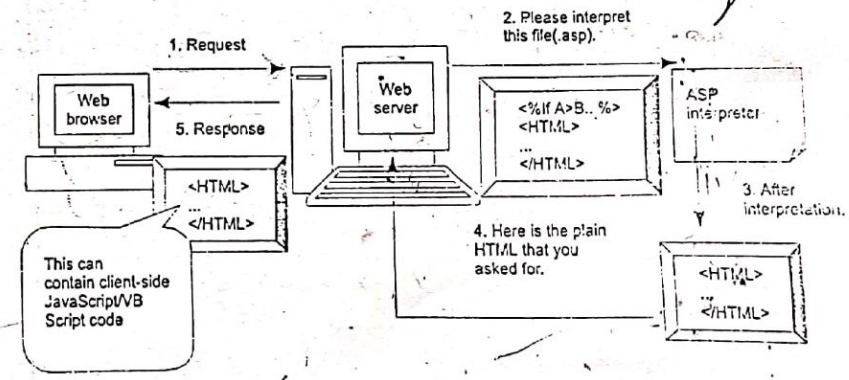


Fig. 9.20 Execution of an ASP Program

Scanned by CamScanner

BASICS OF ASP TECHNOLOGY

ASP scripts (i.e. ASP programs) can be written in VBScript or JavaScript. ASP runs on only Microsoft's Web server, the name of which is Internet Information Server (IIS). It cannot execute on other operating systems or Web servers such as Apache and Java Web server without additional libraries. The reason ASP depends on Windows is the fact that internally, it uses COM components (we shall study COM in brief later) for functioning—the COM components in ASP are known as ASP objects. For example, as we shall see, a request ASP object is used by a Web server to obtain the contents of the HTTP request sent by the browser. The code for this object is internally written by following the COM specification, which is used only in the Windows operating system.

COM, in turn, extensively uses Microsoft Windows registry. Windows registry is like a start up file, wherein every Windows application needs to have an entry. The registry maintains all the parameters, settings, versions and any other entries needed by any Windows application to run, system-defined or user-defined. Therefore, there is a very close relation between COM and the Windows operating system. Because ASP is based on COM, it is also closely tied to Windows. Also, since IIS is also based on COM, it also runs only on Windows operating system—in fact, it is now integrated as a default with Windows 2000. These days, some third-party add-ons are available that allow porting of ASP applications on non-Windows platform. But these add-ons need to be purchased separately.

We will now take a look at an example of an ASP of an page to get a feel of it. This section gives an overview to programmers who have good programming concepts, but for whom ASP is new. Those who do not have any programming background at all may find it a little hard to grasp, though, we believe, they can still generally appreciate what is going on, as the explanation is very simple. So, don't be afraid. Just press on. However, if you are just not interested in programming details, you can skip this section!

ASP EXAMPLE

Before we get into ASP, let us discuss a simple HTML page that calls our ASP. Suppose we have an online bookstore, where a user can search for the availability of a particular book. For this, we show the user a simple form, where the user can type the name of the book to search for, and press the *Search* button. Of course, to get this page, the user would have typed the URL for this page and sent it over the Internet to the Web server. The resulting page is shown first, followed by the HTML code that generates this page; in Figures 9.21 and 9.22 respectively.

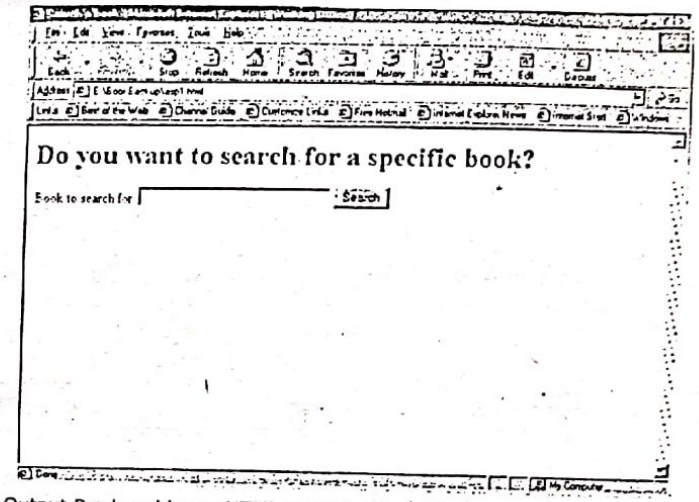


Fig. 9.21 Output Produced by an HTML Page for Searching a Book

The HTML code required for generating the page shown in Figure 9.21 is depicted in Figure 9.22.

```
1.  <HTML>
2.  <HEAD>
3.  <TITLE> Search a book</TITLE>
4.  </HEAD>
5.  <H1> Do you want to search for a specific book? </H1>
6.  <FORM METHOD = "GET" action = "search.asp">
7.  Book to search for: <INPUT type = "text" NAME = "book" SIZE = 30>
8.  <INPUT TYPE = SUBMIT VALUE = "Search">
9.  </FORM>
10. </HTML>
```

Fig. 9.22 HTML Code for Producing the Output as Shown in Figure 9.21

We have added line numbers for ease of reference. In real life, line numbers must not be used.

Let us discuss this HTML code. It has all the usual tags, such as <HTML>, <HEAD>, <TITLE> and their respective closing tags, as shown in lines 1–4 and 10.

In addition, it has a *<FORM>* tag in line 6. As we learned in the section on *Forms*, this tag is required when we want to send user inputs from the client to the server. The *GET* keyword in the same line indicates that the data submitted in the form should be accessible to the Web server. (It *gets* it from the browser). The *action* tag in line 6 indicates the name of the ASP to be executed when the user enters some book name to be searched in the screen shown earlier and presses the *Search* button. In this case, when the user enters a book name and presses the Search button, the browser would send an HTTP request, as usual, to the server. However as we have studied before, the address in this case is not of a static page but that of an ASP program, called as *Search.asp*. Then the server would, with the help of the operating system, locate the file *Search.asp*, load it from the disk and hand it over to the ASP interpreter.

Scanned by CamScanner

Note that the book name entered by the user would be received by the *Search.asp* program in the variable *book*, as shown in the HTML code in line 7. Also, the maximum size of the input field is restricted to 30 characters, which means that the user can enter a book name up to 30 characters for searching. If the user enters more than 30 characters, they would be accepted by the form; however, only the first 30 would be used when sending the form to the Web server—the rest would be discarded. The *<INPUT>* tag in line 8 is used to display the *Search* button on the screen. The input value *SUBMIT* means that when the user clicks on this search button, the value input by the user should be submitted to the server (as a parameter stored in a variable *book*) and a program associated with this form (i.e. in this case search.asp) should be requested to be run on the server:

Suppose the user enters 'Understanding Communism' in the search box and presses the *Search* button. As a result, the URL sent from the browser to the server is:

http://search.asp?book=Understanding Communism

(Actually, this is not quite correct. Special characters replace the spaces before the browser sends the URL. However, we shall ignore this.) Note that the data entered by the user is appended with a question mark to the URL of the ASP to be called next. This data is received as a parameter by the ASP program. Thus, the question mark (?) serves as the delimiter.

What should *search.asp* do? Obviously, it should use the book name entered by the user as the search criterion and look for it in the database of books. If a book with that name is found, it should send a message back to the user. In real life, the program might proceed to see if the user wants to buy it or know more about it. For simplicity, we would not go into further details and simply display either a *Book found* or *Book not found* message on the screen. Let us now take a look at the code of search.asp before we describe it, as shown in Figure 9.23.

```
1.  <% @LANGUAGE = VBSCRIPT %>
2.  <%
3.  Dim objConn, objRS, strQ, strConnection
4.  Dim strBook
5.  strBook = Request ("book")
6.  Set objConn = Server.CreateObject ("ADODB.Connection")
7.  strConnection = "Data Source = Books;"
8.  strConnection = strConnection & "User id=Atul;Password=Atul;"
9.  objConn.Open strConnection
10. Set objRS = Server.CreateObject ("ADODB.Recordset")
11. Set objRS.ActiveConnection = objConn
12. strQ = "SELECT Book_Name FROM Books WHERE Book_Name = strBook"
13. objRS.open StrQ
14. %>
15. <HTML>
16. <TITLE> Search results </TITLE>
17. <BODY>
18. The book you searched for was <%strBook%>. <BR>
19. <%
20. If objRS.EOF
21. Response.Write "Sorry! The book does not exist in our bookstore."
22. Else
23. Response.Write "The book is found."
24. %>
25. </BODY>
26. </HTML>
```

Fig. 9.23 *Search.asp* Page

As we have noted, when the user presses *Search* button on the previous screen, this ASP gets called. It receives the data sent by the previous HTML—in this case, it is a single variable called as *book*.

Let us now discuss the various statements involved in the ASP and how they work.

<% @LANGUAGE = VBSCRIPT %>

The first line begins with a strange set of characters, <%. These characters are specific to ASP. They are special characters, which indicate to the ASP interpreter that this is an ASP code and that it should process this portion. Therefore, as a simple rule, always remember that whenever the ASP interpreter sees anything between the beginning tag <% and the end tag %>, it considers that portion as ASP script (not HTML commands) and processes it. It ignores everything else. We have shown the three ASP related blocks. They are: line 1, 2–14 and 19–24. Each block is encapsulated between the <% and %> tags. The rest is HTML code. In the first block of only one line (line 1) the ASP command instructs the ASP interpreter that the scripting language used in this ASP page is VBScript. It is always a good idea to clarify this point, so that there is no ambiguity. Of course, we could have as well used JavaScript or any other scripting language such as Python. Specifying the scripting language upfront, allows the ASP interpreter to load the appropriate script-interpreting engine, in turn.

<%

On the next line, there is another <% tag. This indicates that there is another portion of ASP script from this point onwards, until a closing %> tag is encountered. Therefore, the ASP interpreter would again process all the statements from this point onwards until it encounters a %> tag. Following this, a few variables are declared using the *Dim* statement. The next line is interesting:

strBook = Request ("book")

We have noted that ASP contains COM components (called *ASP objects*). There are six of them. One of the six major ASP objects is *Request*. It is used to capture the data received from the Web browser. In this case, we are interested in obtaining the value of the variable *book*. Recall that the previous HTML had a *FORM* tag, which also contained a variable called as *book*; the value of this variable would be the book name that the user wants to search. Our ASP obtains this with the help of the *Request* object. So, if the user had entered 'What is Communism?' in the box and pressed the *Search* button, we would now have 'What is Communism?' as the value of the variable strBook.

Now we want to search the database for this book. For accessing data from any database, ASP needs to establish a connection with the database. ASP uses ActiveX Data Objects (ADO) technology for interactions with databases. A detailed discussion of ADO is beyond the scope of the current text. The following statements do this:

```
Set objConn = Server.CreateObject ("ADODB.Connection")
strConnection = "Data Source = Books;"
strConnection = strConnection & "User id=Atul;Password=Atul;"
objConn.Open strConnection
```

The first line here creates an object of type *connection*. We need not go into its details. However, its simple meaning would be that a connection—just like a telephone connection—is to be established between our *search.asp* and the *Books* database, so that our ASP can retrieve the data of interest from the database via this connection. Since the database can be on a different computer/location from that of the ASP page, this connection is necessary. Note that the connection is not yet created—this statement just declares an intention of the connection. As we shall see, for creating this connection, we need to open it (fourth statement in our aforementioned code). The next line identifies the database name, *Books*. In general, we also need to supply the user id and password to access the database. Here, we assume that it

Scanned by CamScanner

is hardcoded into our program. Normally, the user's login id and password can be used for this. The third line specifies these details. The & operator joins the user id and password to the database name. This means that after the third line is executed, the variable *strConnection* would contain *Data source=Books;User id=Atul;Password=Atul;*. The last statement in the above code fraction actually opens the connection between our ASP and the *Books* database. This means that data can now flow from our ASP to the database, or vice versa.

Let us now take a look at the remaining code fraction for database processing:

```
Set objRS = Server.CreateObject ("ADODB.Recordset")
Set objRS.ActiveConnection = objConn
strQ = "SELECT Book_Name FROM Books WHERE Book_Name = strBook"
objRS.open StrQ
```

ASP uses the concept of record sets. A record set is simply one or more rows of data retrieved from a database. Therefore, whenever we want to retrieve data from a database by using ASP, the resulting values would be placed in a record set. Like a connection, a record set first needs to be created and assigned to an open connection. Here, *objRS* is the name of the record set, which is assigned to the database connection created previously. This is shown in Figure 9.24.

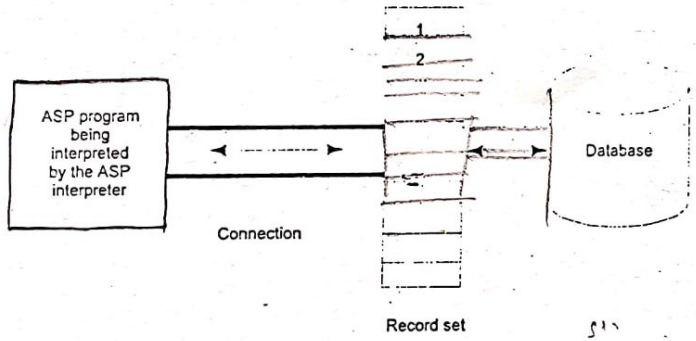


Fig. 9.24 Connection and Recordset in ASP

The third line shows the SQL query that is used to check if we have a book with the name entered by the user. This query is stored in a variable named *strQ*. The fourth statement executes the query and gives the results to the record set. Thus, if there are five books in the database that match the user's search criteria, *objRS* would contain five rows of data.

This is followed by the %> tag. This indicates to the ASP interpreter that the scripting code (for the time being) is over and which indirectly means that some HTML code follows this, in which the ASP interpreter would not be interested. As you can see, there are three HTML statements:

```
<HTML>
<TITLE> Search results </TITLE>
<BODY>
```

We would not bother to describe these. Their meaning should be obvious by now. The ASP interpreter also ignores these lines. The next line is again interesting:


```
The book you searched for was <%strBook%>. <BR>
```

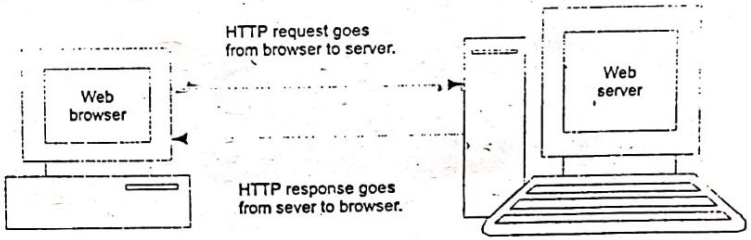
This is for the user's information. For example, if the user had entered 'What is Communism?' while searching, this line would display the following in the browser, when the server finally sends this page back to it:

The book you searched for was What is Communism?.

Note that the variable *strBook* contains the value entered by the user. Since only the ASP knows this variable, it is written in the ASP scripting delimiter tags. The ASP interpreter notes this and replaces *<%strBook%>* with its actual contents. This is followed by another script block:

```
<%
If objRS.EOF
Response.Write "Sorry! The book does not exist in our bookstore."
Else
Response.Write "The book is found."
%>
```

The first line checks to see if the record set is empty (which means that our SELECT query returned no records, that is, there is no book with the title *What is Communism?* in the database of books. Using the VBScript *If* statement, we can check this. The statement performs this task using the record set's end-of-file property. If that is the case, the next statement writes a message *Sorry! The book does not exist in our bookstore.* on the screen. Note that we do this using Response. Whereas the *Request* object is used to retrieve something *from* the Web browser, the *Response* object is used to send something *to* the Web browser, as shown in Figure 9.25.



ASP uses the request and response objects respectively to access the data sent by the client and to send some data back to the client. The request and response objects, in turn, use the HTTP protocol's request and response objects.

Fig. 9.25 ASP and the Request–response Model of HTTP

Of course, if the SELECT query returns some data, we display a message *The book is found.* After the script closing tag, we have the standard HTML closing tags.

This should give us a good idea about the way an ASP page embeds scripting elements written in VBScript, JavaScript, etc. within HTML code. The most important thing to note is that when the Web server receives a request to execute this ASP (*search.asp*), it gives the *search.asp* file to the ASP interpreter. The ASP interpreter examines each line in the ASP page and executes it only if it is within the scripting tags, i.e. within <% and %>. It ignores everything outside of the scripting tags. Thus, the output

Scanned by CamScanner

of the interpreting process is plain HTML. For instance, in this case, after the ASP interpreter completes its interpretation, the following output is produced (assuming the book was found):

```
<HTML>
<TITLE> Search results </TITLE>
<BODY>
The book you searched for was What is Communism? <BR>
The book is found.
</BODY>
</HTML>
```

Note that the ASP interpreter has processed all the scripting elements and only plain HTML is left now. The ASP interpreter returns this output to the Web server. The Web server now sends this plain HTML output to the Web browser, which interprets it. As mentioned before, the only variation would be the case when the ASP page also contains some client-side scripting code. In that case, the client-side scripting code would also travel to the Web browser, which interprets it locally.

MODERN TRENDS IN ASP

Microsoft is coming up with a new version of ASP, called ASP.NET (pronounced as *ASP dot net*). ASP.NET is a better version of ASP, and involves less coding effort on the part of the developer. Also, one can write ASP.NET code in many programming languages, such as Visual C++, Java, COBOL and a new language developed and promoted by Microsoft called C# (pronounced as *C sharp*). The whole package of which ASP.NET is one part, would be called as Microsoft's *dot net* development platform. We shall discuss dot net in Appendix B.

JAVA AND THE CONCEPT OF A VIRTUAL MACHINE

9.10.1 Introduction

Traditionally, the problem with computer languages has been that humans want to write programs in languages that computers do not *directly* understand. Therefore, the natural solution has been to create a series of levels. Each level is built on top of the existing levels. Why is this necessary?

The electronic circuits of a computer can understand and directly execute only a very limited set of basic instructions. Therefore, every program that the computer is expected to execute has to be in this format. These basic instructions are very primitive in nature, such as:

- Compare a number with zero
- Add one number to another
- Copy x bytes from a particular memory location of the computer to another

A set of these instructions that a computer can directly execute, forms what is known as machine language, as it is the closest to the hardware (machine). People designing a new computer must decide in advance what should be the machine language of that computer.

However, directly writing programs in the machine language is quite tedious for humans: it involves writing programs as a series of zeroes and ones! For instance, if you want to add two numbers, the instruction might be 010 (to mean *Add* the numbers that follow the characters 010*)*. Similarly, for comparing a number with zero, the instruction might be 011 (to mean *Compare* the number that follows the characters 011), and so on.

Therefore, computer scientists realized that computers could be structured to form a series of abstractions. Each abstraction builds on the top of the abstraction below it. In this manner, the complexity involved in working with computers can be simplified, as we shall discuss below.

For instance, suppose that we consider the machine language to be at the lowest level (L0). We can now create a new language level say L1 on top of L0. Each L1 instruction can be mapped to exactly one instruction in L0. That is, we can take one instruction of L1, and translate it into its corresponding L0 instruction, which the hardware can directly understand and execute. Of course, we would need to also provide this translator. However, the main reason why L1 would be created as an abstraction level above L0 is that writing programs in L1 would be far easier as compared to writing them in L0. The reason behind this is that L1 would allow us to write programs using English words such as *ADD* (rather than 010), *CMP* (rather than 011), and so on. Thus, we can write an instruction such as *ADD*, give it to the translator, and let it perform the tedious job of translating it to 010. Thereafter, it can be executed as before, as it is now in the L0 format, which the hardware can understand and execute directly. For one instruction, this might seem trivial, but when we consider that a real-life program can contain thousands of such instructions, we can very well appreciate the ease of using L1 rather than L0. Traditionally, the L1 language level is called Assembly Language and the translator that is used for translating as L1 program into L0 is called an Assembler.

Soon people thought that if it were possible to go from L1 to L0, why not have L2 as the third level of languages (above L1), which can be even simpler from a human perspective? For instance, although the L1 language has an instruction such as *ADD* to add two numbers, there was no direct provision in L1 for adding more than two numbers, say six numbers, for instance. Similar limitations existed for other L1 instructions. The reason behind these limitations was technical—any L1 instruction was supposed to directly correspond to an L0 instruction. In order that the L1-to-L0 mapping be kept one-to-one any L1 instruction necessarily had to be mapped to exactly one L0 instruction. Thus, an L1 instruction could not be expected to perform complex tasks. Therefore, we could now have L2 that could perform more complicated tasks than L1 (i.e. it could perform the job of more than one L1 or L0 instruction at one go). As before, we can write a translator for translating an L2 program directly into L1 and then, that into L0 using our earlier L1-to-L0 translator, which can then be executed directly by the hardware as before. Thus, we can now have an instruction such as *Add a, b, c, d, e to f* as an L2 instruction, which was not possible in L1 (or L0). Internally, the translator could translate this instruction to a series of L1 instructions such as:

```
Add   f   a
Add   f   b
...
Add   f   e
```

These L1 instructions could then be translated by the L1-to-L0 translator into L0, which the hardware can execute. As we can see, each level of abstraction eases the tasks of the humans who are writing the programs. Each higher language level is closer to their thinking. L2 is traditionally known as a High Level Language (HLL). Examples of L2 are COBOL, FORTRAN, Pascal, C, C++, Java, etc.

Similarly, we can now imagine that we can have even higher-level languages (L3, L4, and so on) until we are satisfied. Each language level builds on top of its predecessor. For instance, the Structured Query Language (SQL) is at L3 (i.e. Very High Level Language (VHLL)) and supports a very high level instruction such as *SELECT * FROM EMPLOYEE*, which retrieves all the records from the *Employee* database. Performing the same job in a L2 language would require many instructions (such as *Open File*,

Scanned by CamScanner

would require a full C++ compiler targeted for that CPU. The problem in this is that compilers are difficult to write and take some time if each time the source code has to be compiled, linked and executed. Therefore, C++ for example, cannot be used on *any* platform: we must have a compiler for that environment. To solve this problem, James Gosling, Patrick Naughton, Chris Wath, Ed Frank and Mike Sheridan began working on a project at Sun Microsystems in 1991. The project took 18 months to complete and the language developed was initially called *Oak* due to the Oak tree that was visible to all of them from their workplace. However, it was renamed to *Java* in 1995.

Java was thus created with the aim of writing code that could be run on a variety of CPUs under different environments. This experiment was not very successful. The team did not know what to do with the newly created language. Interestingly, almost at the same time, another phenomenon was catching the attention of the entire world: the Internet. Since the Internet is a gigantic network of computers, all running different operating systems on different CPUs, Java suddenly became the focus of attention. The consumer devices were a much smaller issue. People soon realized that Java's inherent platform-independence suited the needs of the Internet quite well. This was the time when Java started to become extremely popular. Today, it is one of the most important programming languages.

Java's syntax was intentionally kept similar to C++ (and hence, C) to enable programmers to start using Java relatively quickly. Java's design is also aimed at removing the unwanted features of C++, especially because it was meant to be used in a non-secure environment such as the Internet. For instance, a Java program downloaded from the Internet cannot write to the hard disk of the person who downloaded that program. Thus, Java addresses the security issues in depth. However, the real key to Java's success is its platform independence.

As we have noted, Java is extremely similar to C++ in terms of syntax. Let us look at a sample Java program that uses a class called *Test*. Then we create two objects of this class, and initialize the value of the variable *somedata* declared inside this class to 100 and 50 respectively. Finally, we display these values. This example can be safely ignored if you are not very keen about the Java language syntax elements.

```
// A very simple Java program.
// This program demonstrates a class named Test.
class Test                          // Declare a class named Test. Note that the convention is to name classes in
                                    // upper case in Java. In C++, it is not so.
{
        int somedata;               // The class has only one attribute named somedata. Note that it is private.
        void setdata (int d)        // Method for setting the value of somedata. It accepts a parameter d.
                                    // Using this parameter, it changes the value of somedata to the value of
                                    // the parameter.
        {somedata = d;}             // This is the actual code in the method setdata. "somedata = d;" makes
                                    // the value of somedata same as the value of d, i.e. the parameter. The
                                    // semicolon indicates that the statement is over.
        void showdata ()            // Method for displaying the value of somedata.
        {System.out.println ("Value of somedata = " + somedata) ;}
// "System.out.println" is used for displaying something in Java. Do not worry much about its exact syntax.
// It is like saying 'Please display whatever follows, on the screen'. Therefore, it prints whatever is given in
// the bracket after it. In this case, it prints Value of somedata =. Note that the quotation marks (")
// themselves are not printed. They simply indicate the beginning and end of what is to be printed. The +
// sign is used for displaying two strings together. The + sign itself is not printed. Finally, somedata after
// the + sign indicates that we want to display the value of somedata and not the text somedata. Suppose
// the value of somedata is set by the method setdata to 100 in the program. Then, the showdata method
// would display a message Value of somedata = 100
```



```
};
class MainFunction
{
        // Note that main method is also inside a class.
        public static void main (String [] args)        // Ignore the portion in the brackets.
{
        // For class test, we have declared two objects named t1 and t2. Java's syntax here is different from
        // C++. Test t1 indicates that we want to declare an object t1 of class Test. Then, = new Test ()
        // indicates that this object needs new memory to be allocated for it. How much of memory? It is
        // the same as the memory required by the class Test. In C++, this was not required. Simply saying
        // Test t1 would be good enough in C++. However, in Java, memory needs to be explicitly
        // allocated for an object using the new keyword. The same thing happens for object t2 of the class
        // Test.
                Test t1 = new Test ( );
                Test t2 = new Test ( );
                t1.setdata (100);       // Call method setdata of the class Test. This would set the value of
                                        // somedata in object t1 to 100. Note that 100 is received by setdata as
                                        // a parameter with the name d. You could name the parameter
                                        // differently.
                t2.setdata (50);        // Similar to above. Set the value of somedata of object t2 to 50.
                t1.showdata ( );        // Call method showdata for object t1. This would display the value
                                        // of somedata for t1, i.e. 100.
                t2.showdata ( );        // Call method showdata for object t2. This would display the value
                                        // of somedata for t2, i.e. 50.
        }
}
```

Web technologies such as Java servlets, Enterprise Java Beans (EJB) and Java Server Pages (JSP) are based on Java. Applets, that make Web pages active, are also written in Java. This discussion would have given you a good overview of the Java technology in general.

JAVA SERVLETS AND JAVA SERVER PAGES (JSP)

When the Java programming language was first introduced, its main purpose was to make the Web pages interactive. This was done with the help of applets. We will learn more about applets in detail later. However, suffice it to say that applets are small pieces of Java code embedded in the HTML tags. The whole thing is stored on the server as a Web page. This Web page is sent from the server to the client (browser), where it executes. Thus, applets make client-side interactive. However, soon, the issue of (lack of) speed became prominent. Downloading applets on the client side is a slow process. Therefore, a movement was started to add Java also to the server side.

Like a CGI program, or an ASP script, a Java servlet is a program that executes on the Web server in response to client requests for dynamic Web pages. Since servlets are written in the Java programming language, in theory, they should be able execute on any platform—thanks to Java's Virtual Machine (JVM) concept. Unlike ASP, almost all platforms and Web servers support servlets for this simple reason.

Soon, Sun Microsystems realized that servlet programming was pretty cryptic. In contrast, Microsoft's ASP was far simpler, and therefore, getting more popular. To counter the success of ASP, Sun came up with Java Server Pages (JSP), which can be considered as the next logical step of servlets, because JSP is easier to code than servlets. When the developer writes a JSP, internally, the compiler produces a servlet before it is executed. The JSP technology is extremely similar to ASP in concept. Having discussed ASP, we shall now study servlets and JSP.

Scanned by CamScanner

JAVA SERVLETS

9.12.1 Introduction

A Java servlet contains HTML tags written within Java statements, unlike a Java applet, where Java code is embedded in the HTML tags. A servlet is written in Java and compiled on the server into its bytecode file and stored on the server. When a client requests for a servlet, the Java bytecode (i.e. the code resulted from the servlet compilation) for that servlet is executed by the JVM residing in the Web server. This produces some output, which is converted to produce HTML statements by the servlet. The resulting final output is thus plain HTML, which the Web server sends back to the client. This is shown in Figure 9.28.

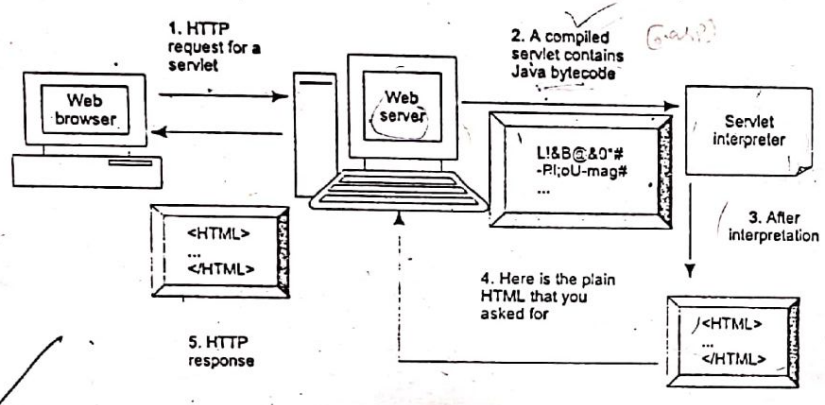


Fig. 9.28 Execution of a Java Servlet

Let us now discuss the life cycle of a servlet.

9.12.2 Life Cycle of a Servlet

The life cycle of a servlet is very straightforward. The servlet needs to be first compiled into the Java bytecode. For this, when the Web server receives the very first request for the execution of a particular servlet (from any client), the Web server locates the servlet file, loads it in its memory, and gives it to the servlet compiler. The servlet compiler compiles it into Java bytecode. This bytecode is loaded by the Web server in its memory and is interpreted by the Java Virtual Machine (JVM) for this as well as for any future requests for this servlet. Thus, the servlet is compiled only once—when it is called for the very first time (or, of course, when it is changed and therefore, needs to be recompiled). Once the bytecode file of the servlet is loaded in the memory of the Web server, the following things happen:

1. A servlet is firstly *initialized*.
2. It then *services* one or more clients until the servlet application is no longer required.
3. At this point, the servlet is *destroyed*.

Thus, unlike CGI, a servlet (i.e. its compiled bytecode) is loaded in the memory only once and it stays resident in memory until it is destroyed.

Every Java servlet has three methods: *init()*, *service()* and *destroy()*. The Web server invokes these methods at specific times. Let us understand how these methods are useful, with the help of the following steps:

1. The Web server receives the client's HTTP request. It maps it on to a particular servlet. Now, there are two situations:
   (a) If this servlet is being called for the very first time, the server locates the servlet file on the disk and gives it to the servlet compiler. The servlet compiler compiles the servlet into Java bytecode. Now, this bytecode can be directly interpreted by the JVM inside the Web server.
   (b) If the servlet was called previously, it must already be compiled into bytecode and loaded into the memory of the server. Therefore, the server simply hands this bytecode to the JVM for interpretation. Here, re-compilation of the servlet is unnecessary.

This is shown in Figure 9.29.

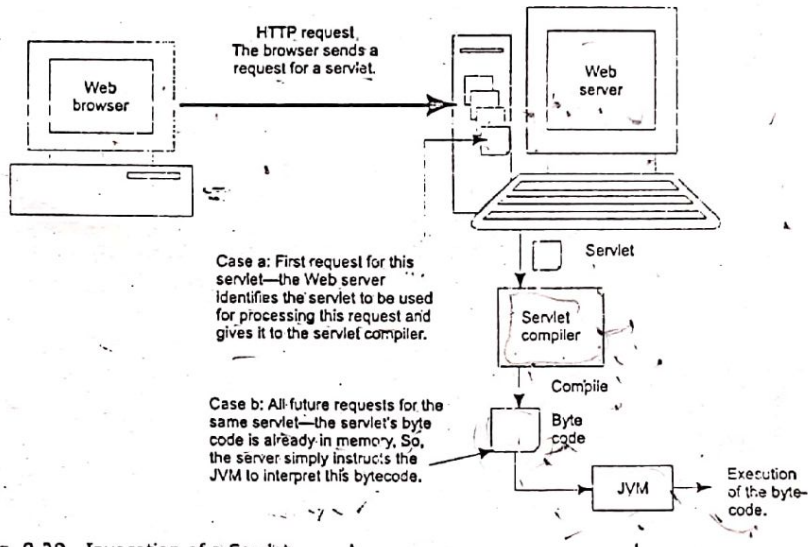


Fig. 9.29 Invocation of a Servlet

2. The server (note that actually, it is the JVM—however, since the JVM is inside the server anyway, we shall ignore this minute detail and assume that the server executes the bytecode of the servlet) then invokes the *init()* method of the servlet. The *init()* method performs all the first-time

Scanned by CamScanner

initializations, such as opening database connections, files, initializing global variables, etc. Note that this is done only when the servlet is loaded in memory for the first time. For all future requests, this step is not performed at all. This is shown in Figure 9.30.

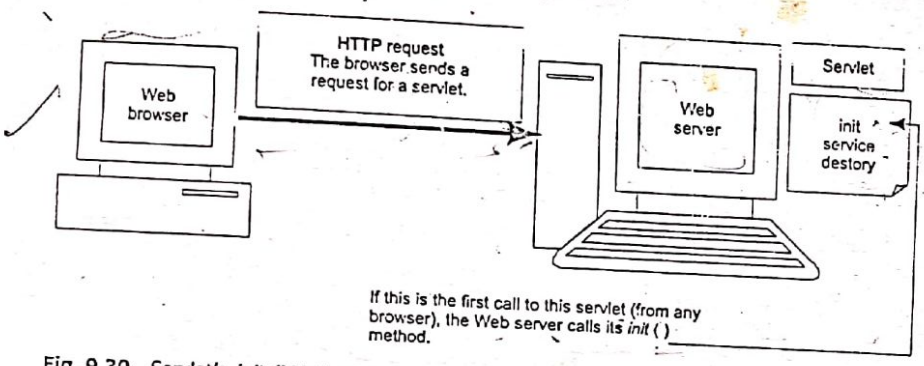


Fig. 9.30 Servlet's *init ()* Method is Called

3. Next, the Web server calls the servlet's *service ()* method for processing the HTTP request. A servlet has access to the complete HTTP request sent by the client. Thus, it can read data sent by the client, perform the necessary processing and formulate the HTTP response. The Web server then sends the HTTP response back to the client, as shown in Figure 9.31.

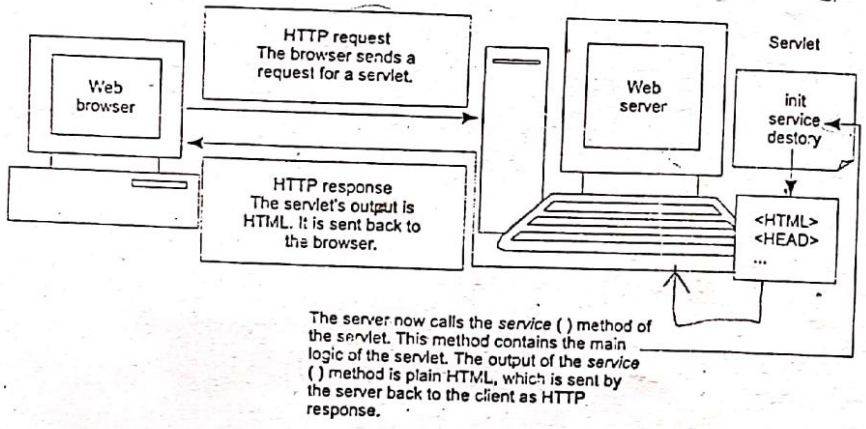


Fig. 9.31 Servlet's *Service()* Method is Called

4. The servlet then remains in the address space of the server. It is available for processing any more HTTP requests received from the same or any other client. The *service ()* method is called for each HTTP request, which invokes that servlet.
5. At some point of time, the Web server decides to unload the servlet from its memory. The server calls the *destroy ()* method to unload the servlet, as shown in Figure 9.32. When to destroy a servlet is a decision left entirely to the Web server.

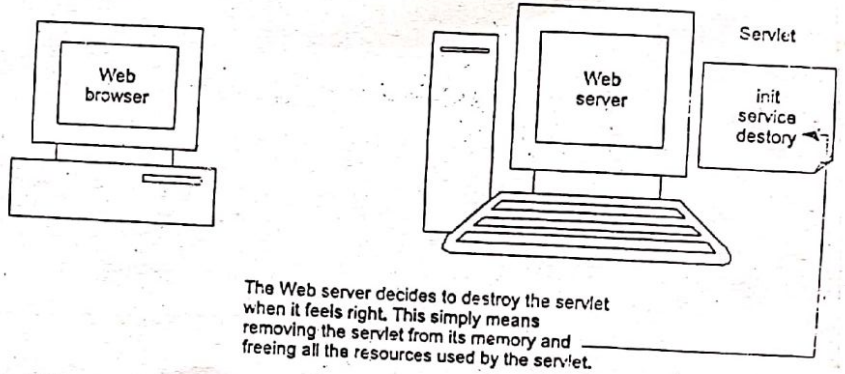


Fig. 9.32 Servlet's *Destroy()* Method is Called

9.12.3 Servlet Example

Let us study a simple servlet example. Since servlet code can be extremely tough for those not at all familiar with Java, we shall restrict ourselves to a very primitive example, basically to illustrate the concepts. Let us have a screen for a user to select one of the colours (red, green and blue) in the browser and to press the *Select* button. A servlet on the Web server detects which colour the user has chosen, and simply displays the name of the chosen colour. Of course, a servlet may not be required in practice for this: even a client side script can do this. However, this example is taken just for illustrative purposes.

Suppose the user gets the following screen, as shown in Figure 9.33.

Scanned by CamScanner

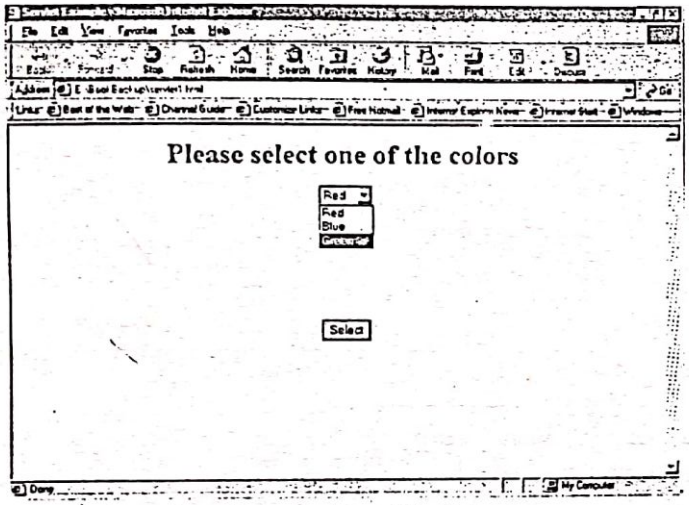


Fig. 9.33 Sample Screen from which a Servlet would be Called

As Figure 9.33 shows, the user can select one of the three colours from the drop-down list and press the *Select* button. Before the user took cursor to the drop-down list, *Red* was the default, and hence, is shown in the selection list as well as the box where the final selection would be made. Currently, the user has taken his cursor to *Green*. The HTML code required for the above page is shown in Figure 9.34.

```
<HTML>
<TITLE> Servlet Example</TITLE>
<BODY>
<CENTER>
<FORM NAME="FORM1" ACTION="ColorServlet">
<H1> Please select one of the colors </H1>
<SELECT name="color" SIZE=1>
<OPTION VALUE="Red">Red</OPTION>
<OPTION VALUE="Blue">Blue</OPTION>
<OPTION VALUE="Green">Green</OPTION>
</SELECT>
<BR><BR><BR><BR><BR><BR><BR><BR>
<INPUT TYPE=submit VALUE="Select">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Fig. 9.34 HTML Code Corresponding to the Output Shown in Figure 9.34

Note that this is a plain HTML static page. All tags here are standard HTML tags. The *<SELECT>* tag begins the code for drop-down list. The three choices for colours are displayed using the *<OPTION>* tags. The user can choose one of the three colors and press the *Select* button. At this point of time, the selected colour gets stored in the variable *colour* (specified in the *<SELECT>* tag) and is passed to the Web server along with a call to a servlet known as *ColorServlet*. Suppose the user selects *Blue*. The URL from the browser to the server would be:

http://ColorServlet?color=Blue

Thus, the HTML *<FORM>* property sends the form data to any server-side application (CGI, ASP, servlet or any other server side technology) in the same way. At the server side, the *ColorServlet* is waiting to receive this selection. What would its code look like? Let us first view the code as shown in Figure 9.35 and then discuss it. We have shown the servlet code as bold. All the rest are comments.

```
// import statements are used to add the standard readymade Java classes. Here, three import statements
// are used to import all the classes that contain the functionality for Java input-output, servlet and http mechanisms.
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// The following statement declares our servlet. Also, it is inherited from the standard HttpServlet class.
// Thus, our servlet obtains all the attributes and methods of the Java standard servlet class.
public class ColorServlet extends HttpServlet
{
        // doGet is a method as the counterpart of our HTML <FORM> tag. Thus, whatever data
        // the <FORM> tag sends to the servlet is available to the doGet method here. This method
        // has two parameters: one to receive the HTTP request from the client and another to
        // form an HTTP response to be sent back to the client. Ignore the throws portion—it is Java
        // standard that need not be discussed in the current text.
        public void doGet (HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException
        {
                // Like ASP's request object, we have a request object here as well. This gives us an
                // access to the value of the color variable sent by the browser along with the HTTP
                // request to the server.
                String color = request.getParameter ("color");
                // Again, we have a response object here that is used to send data back to client.
                // The first line is a required portion of the HTTP header. We can ignore it.
                Response.setContentType ("text/HTML");
                // The following code is used to obtain an object for writing. Again, Java specific!
                // However, note that we want to write to the HTTP response object.
                PrintWriter pw = response.getWriter ();
                // Note how HTML code is embedded within Java's println method.
                pw.println ("<B>You have selected: ");
                pw.println (color); pw.println ("</B>");
                pw.close ();
        }
}
```

Fig. 9.35 Servlet Code

We have added a lot of comments (indicated by the // characters) to the servlet to make it readable. Therefore, we shall not describe every statement again. Rather, let us have an overview of the servlet. Also, the actual code is in boldface for the ease of reading.

Scanned by CamScanner

The servlet is a set of Java statements. There is a user-defined class named *ColorServlet*, which must be the same name by which the previous HTML code calls it. This class implements a method for obtaining the HTTP request object from the client. From that object, it retrieves the value of the variable *color*, and displays it along with a message. Thus, if the user had selected blue in the browser, the servlet would display a message:

You have selected Blue

Now the obvious question would be: what happened to the servlet lifecycle? Where have the *init ()*, *service* and *destroy ()* methods gone? Actually, because we have inherited our *ColorServlet* class from the standard Java class HttpServlet, we need not include these methods in our servlet. These methods are already defined in HttpServlet (or the classes from which HttpServlet itself is inherited). Therefore, although we have not defined these methods, they are already available to the Web server. We are not overriding them, as we do not need them in the application. See the power of inheritance?

As a result, when our servlet gets called from a client for the first time, it is compiled by the servlet compiler into Java bytecode, and loaded into memory. The JVM in the Web server's address space then interprets the above servlet code, which results into an HTTP response object getting created. The Web server sends that HTTP response back to the client. In the meanwhile, our servlet remains in the memory of the Web server, waiting to service more client requests. If it is not called for a long time by any more clients and is occupying valuable Web server memory, which is needed by other servlets or processes, the Web server might destroy it.

JAVA SERVER PAGES (JSP)

9.13.1 Introduction

Java Server Pages (JSP) is one step ahead of servlets. Initially, JSP did not exist. Only servlets technology was available. However, as you might have already concluded, servlets are cumbersome to write, because of the cryptic syntax of the Java language as well as because you have to write these cryptic statements within HTML tags: making it even more complex to understand. As we have seen, a servlet contains HTML tags embedded in Java language.

For instance, you need to write tags such as <HTML>, <B> etc within Java code, in the case of a servlet. The resulting code is not easy to understand or debug. The normal scripting practice is the other way round: that of embedding scripting code within HTML tags. This makes servlets very cryptic. As a result, Sun Microsystems came up with JSP, which is a far more friendly technology.

A JSP page contains Java statements within HTML tags. This is the traditional scripting method, which makes the code look a lot simpler as well as easier to test and maintain. This is similar to ASP, which allows writing VBScript and JavaScript in HTML tags. Let us understand how this works.

9.13.2 Lifecycle of a JSP

1. When a client invokes a particular JSP for the first time, the Web server locates the JSP file.
2. If found, the Web server loads a *JSP engine* (which is nothing but a JSP compiler) and passes control to it along with the JSP file.
3. The JSP engine compiles the JSP file into a temporary servlet file. The temporary servlet is then compiled into Java bytecode, which can be interpreted by the Java Virtual Machine (JVM) inside the Web server whenever the servlet is to be executed. Note that the temporary servlet is not available directly to the developer. From the developer's point of view, the JSP itself can be modified, not the temporary servlet.

4. The servlet is now executed, like any other servlet.
5. When any more clients send requests for the same JSP, the Web server knows that the servlet corresponding to it is already in memory. So, it uses the same servlet bytecode. This is the reason why there is a slight delay the very first time a JSP is called, as the JSP needs to be compiled into a servlet and then subsequently into the Java bytecode. But all subsequent calls to the JSP are automatically re-directed to the bytecode that is already in memory; and therefore, are processed much faster.

This is shown in Figure 9.36.

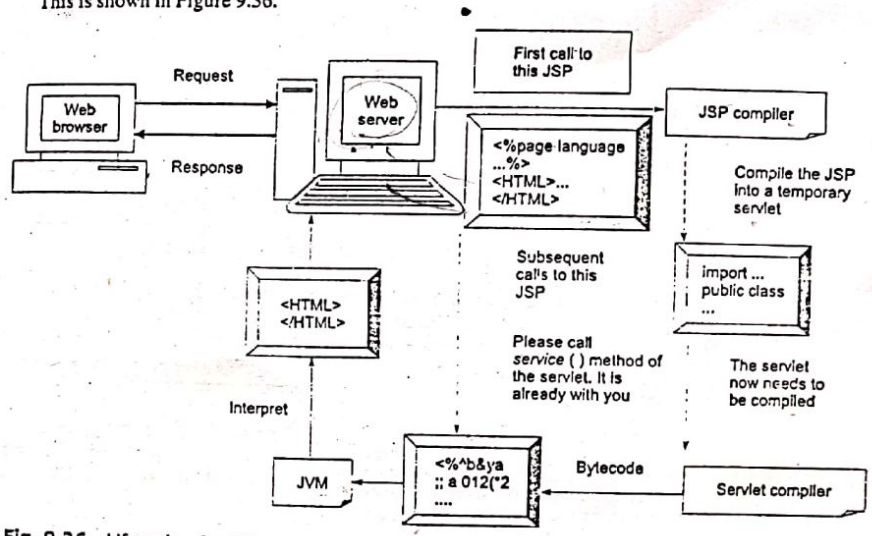


Fig. 9.36 Lifecycle of a JSP

9.13.3 JSP Example

Let us take a look at the JSP code that implements the same functionality as our servlet example, as shown in Figure 9.37. Like ASP, the JSP code is much simpler to understand as compared to a servlet. Also, a JSP has the logic (scripts) in blocks that are distinct from than the presentation (HTML)—making the code look cleaner. This is another advantage of JSP over servlets.

```
<%@ page language="java" %>

<HTML>
You have selected: <%request.getParameter ("color")%>
</HTML>
```

Fig. 9.37 JSP Example

Scanned by CamScanner

The four lines accomplish the same functionality as the servlet. Let us study them one by one.

<%@ page language="java" %>

This line indicates to the Web server that any scripts found in this page should be treated as Java code. This simply means that the Web server should load the appropriate compiler for compiling it (first into a servlet, and then into bytecode, as explained earlier).

<HTML>

Like any other HTML page, this indicates the start of the HTML page.

You have selected: <%request.getParameter ("color")%>

This line displays the message regarding the colour selected by the user. The portion up to the colon (:) is pretty straightforward—it is pure text to be sent back by the server to the browser as it is. The next portion indicates that the server should obtain the value of the parameter *color* (which arrives with the HTTP request from the browser) and append it to the earlier message. So, if the user has selected *Blue*, this line would produce this output upon execution:

You have selected: Blue

</HTML>

Like any other HTML page, this indicates the end of the HTML page.

No wonder JSP is very popular among Java server-side programmers as compared to servlets! It is much cleaner and simpler to code and maintain.

JSPs are used in conjunction with another technology known as Java Beans. A Java bean is a small and self-sufficient program. For instance, a bean could represent a button on the screen. A JSP can then use this bean to check if the button is right-clicked, left-clicked, released, etc. If the JSP treats everything else on the screen as beans, a whole hierarchy can be constructed where beans interact with each other, and with JSP. Since the code of each bean is stored in a separate file, calling beans from JSP minimizes code cluttering even further. Also, because beans are Java classes, the reusability of beans is another important factor to be considered when using beans with JSP.

Summary

A dynamic Web page is created at *run time*, on the fly, rather than being created and stored on the hard disk of a Web server. Dynamic Web pages are extremely important from the context of electronic commerce.

When a browser sends a request for a dynamic Web page in the form of a program's URL, the Web server uses this URL to locate the program, loads it in the memory of the server, and executes it with the help of the appropriate interpreter and sends the resulting output (which is in plain HTML format) back to the browser. To create dynamic Web pages, some sort of programming mechanism is required. There are various programming tools/development platforms available for making Web pages dynamic. Active Server Pages (ASP), Java Servlets and Java Server Pages (JSP), and Common Gateway Interface (CGI) are examples of dynamic Web page technologies.

For client-side validations and local processing, client-side scripting can be used. In general, a typical dynamic Web page contains both server-side and client-side scripts. The two most popular Web-browsers are *Netscape Navigator* and *Microsoft Internet Explorer*. These and other browsers can execute these scripts embedded in HTML Web pages. Scripts are written in languages such as JavaScript or VBScript.

Style sheets allow pre-defining the way different tags should be interpreted for the purpose of displaying them. This technique leads to the technology of Dynamic HTML (DHTML).

The Common Gateway Interface (CGI) is the earliest and one of the most popular server-side programming techniques. CGI makes dynamic generation of Web pages possible by executing a program on the Web server.

These days, CGI is not the preferred choice of server-side Web development. Simpler technologies such as ASP, JSP and servlets are becoming more popular. To counter them, a new standard called as fast CGI has come up, which removes the drawback of spawning a new process for every request.

Microsoft's Active Server Pages (ASP) combines all the good features of previous attempts made to make Web pages interactive. It allows easy programming, database access, operating system access and speed.

Microsoft is coming out with ASP.NET: a new standard of ASP. ASP.NET is an improvement over ASP in many areas such as session management, and also allows the scripts to be written in programming languages such as C++, COBOL and a new language known as C# (pronounced C sharp).

Java servlet is a program that executes on the Web server in response to client requests for dynamic Web pages. Servlets are written in the Java programming language. Therefore, they execute on any platform. Soon, Sun Microsystems realized that servlet programming was pretty cryptic. So, Sun came up with Java Server Pages (JSP), which can be considered as the next logical step of servlets. The JSP technology is extremely similar to ASP.

Key Terms and Concepts

Active Server Pages (ASP)
ActiveX Data Object (ADO)
Apache Server
Applet
Bytecode
Client-side script
Common Gateway Interface (CGI)
Dynamic HTML (DHTML)
Internet Information Server (IIS)
Intranet
Java beans
Java Server Pages (JSP)
Java Servlets
JavaScript
JScript
Python
Server-side programming
Style sheet
VBScript

Review Questions

OBJECTIVE QUESTIONS

State whether the following are True or False:

1. A dynamic Web page is created at *run time*.
2. A dynamic Web page is simpler than a static Web page.

Scanned by CamScanner

6. What are the advantages and issues involved in dynamic Web pages?
7. What are style sheets? How are they useful?
8. Discuss how CGI works at a conceptual level.
9. What is the biggest disadvantage of CGI?
10. How does ASP score over CGI?
11. Why is ASP a Microsoft-only technology?
12. With the help of an example, describe how ASP works.
13. Describe ADO and how it can be used to interact with databases.
14. Why are servlets platform-independent?
15. Describe the lifecycle of a servlet.
16. Discuss with the help of an example, how servlets work.
17. Why was JSP required?
18. Discuss the lifecycle of a JSP.
19. What are Java Beans?

EXERCISES

1. Investigate what one needs to do to execute CGI, ASP, servlets or JSP programs. Actually set up the necessary environments for at least two of these technologies.
2. Assume that your shop contains the following items.

Item code	Item name	Price	Discount %
01	Pencil	5	1
02	Pen	20	1.5
03	Notebook	40	2
04	Diary	80	2.25

   Now write a CGI program that accepts an item code and displays the corresponding item name. (Hint: Refer to a CGI book for syntax).
3. Write an ASP program to accept an item code from the user, and display its original and post-discount price. (Hint: Refer to an ASP book for syntax).
4. Write a simple servlet that accepts the user's age, and displays a message according to the following rules:
   (a) Age < 15: Message: You are a kid!
   (b) Age between 16 and 40: You are young!
   (c) Age above 40: You are old!
   Use JavaScript to ensure that the user enters a number, and display the message from the servlet. (Hint: Refer to a book that explains servlets and JavaScript syntaxes).
5. Try displaying the above message from within JavaScript itself.
6. Write a JSP program using the idea of Exercise 2. (Hint: Refer to a JSP book for syntax).

Chapter 10

Active Web Pages

Active Web pages came into existence as a result of queries such as:

- Why do images on the Internet take so long to download on to the client?
- Why can the Internet pages containing multimedia such as sound and video not have smooth movement like television pictures, one after the other?

These problems of continuity can be attributed to a simple fact that the Internet was created to work with static documents, and not to transport video. The initial solution for adding animation capabilities was known as **client pull**. We know that if images are shown to us at the rate of about 30 per second (theoretically more than 10 per second), it can fool our eyes and create the illusion of continuous motion, i.e. animation. This is because of a simple biological fact that an image is retained on the retina of our eyes only for 1/10th of a second. Thus, before the first one fades away, if the second image is superimposed on it, it creates an illusion of continuity.

These images are stored on the Web pages, which are stored on the Web server. In this technique of client pull, the browser running on the client repeatedly requests the server for Web pages containing images and on receiving them, displays them one after the other at the rate of 30 per second to create the animation. This process is relatively slow. This is because we know that for downloading every Web page, a client has to make a separate request through the HTTP protocol. This is shown in Figure 10.1.

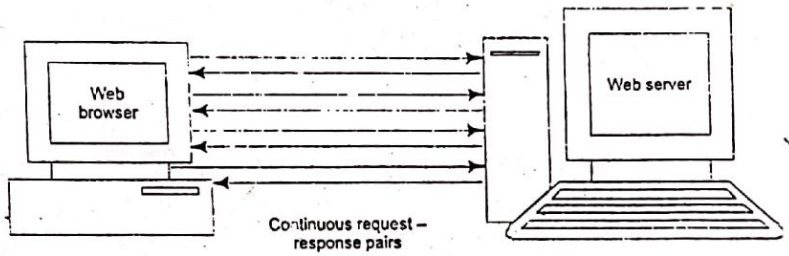


Fig. 10.1 Client Pull

How does the Web browser know how many pages to pull? The information is actually stored in the page itself. When someone creates a Web page that intends to employ client-pull, tags need to be added which

Scanned by CamScanner

specify that a browser should pull another page after this. Thus, the first page can instruct the browser to pull the second page, the second page can instruct the browser to pull the third page, and so on. The last page in this sequence instructs the browser to pull the first page again, thus creating an illusion of animation, if all these pages are shown 30 times each second.

The problem with this scheme is that it seems fine in theory. However, there are practical problems in implementing it. The problems are to do with the speed. Even an extremely powerful computer running a browser and using a high-speed Internet connection cannot receive images so fast as to fool the human eye and make it believe that these discrete images are actually continuous. There is a clear distortion and delay. The browser takes some time to pull these images, thus defeating the aim. This means that although client-pull seems to be useful, in practice, it allows for only jerky motion and primitive animation.

ACTIVE WEB PAGES IS A BETTER SOLUTION

The problems associated with client-pull are overcome using newer and more powerful Web technologies. These technologies allow smooth motion and animation. Here, the aim is to make Web pages active. An active Web page actually contains a computer program for drawing an image on the screen of the computer. This program executes inside the client—that is the browser. This works as follows:

1. Similar to static and dynamic Web pages, active Web pages are also stored on Web servers. With each active Web page, a URL is associated.
2. The retrieval of an active Web page is also no different from static and dynamic Web pages. When a browser specifies the URL of an active Web page, the Web server locates the file containing the active Web page and sends it to the client, with a difference that dynamic pages involve some processing at the server side with the help of CGI, ASP, JSP, etc. to generate a Web page in HTML format dynamically, before downloading on to the client.
3. Here is where active Web pages are different from static and dynamic Web pages: A static or dynamic Web page, once in the final HTML format on the client machine, is simply interpreted by the browser. However, in the case of an active Web page, the browser actually runs a computer program embedded in the active Web page. This program draws an image on the user's computer screen. Thus, active Web pages execute on the client and not on the server, unlike technologies discussed earlier. That is why they are faster, once downloaded to the client.

JAVA APPLETS

Sun Microsystems has developed a very popular active Web page technology. This involves the use of Java applets. An applet is a small program written in the Java programming language, and is embedded in HTML page to form a Web page. An applet makes a Web page active. The applet gets downloaded to the Web browser (client) along with the requested Web page and is executed there under the control of the Java Virtual Machine (JVM) installed in the browser. An applet then creates animations on the client computer. Other similar technologies are based on this primary concept. Also, we must remember that though they were used originally used primarily for animation, applets can be used for many other applications, as discussed later.

This is shown in Figure 10.2.

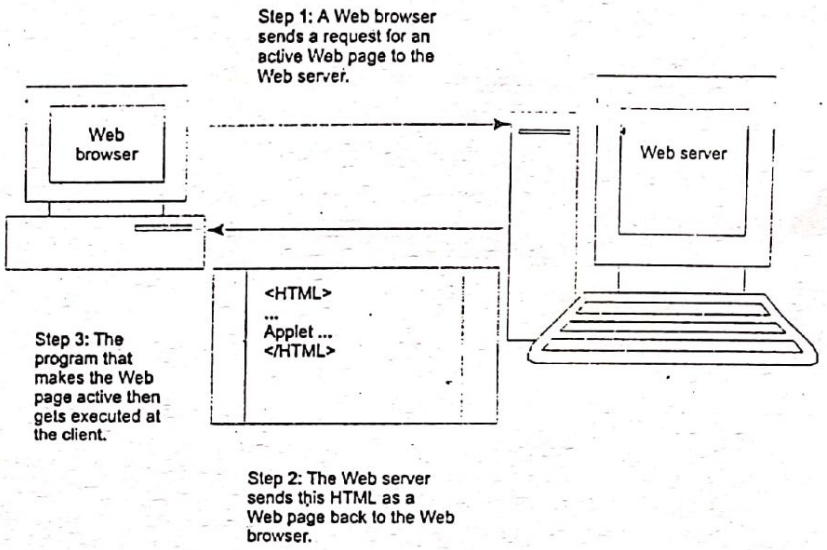


Fig. 10.2 Active Web Pages and Java Applets

WHY ARE ACTIVE WEB PAGES POWERFUL?

Active Web page technology is powerful because of the following reasons:

1. Active Web pages get downloaded on to the client computer. There, they locally perform computations and tasks such as drawing images and creating animations. Therefore, there is no delay between the creation of an image and its display. Obviously, once an active Web page is downloaded on to the client computer, there is no reason for contacting the Web server again, unlike client pull. As a result, the client computer has full control in terms of displaying animations. Slow Internet connection speed does not matter. Even if the connection is slow, it will take a little longer to download the entire Web page, but once downloaded, the animation will look continuous and not jerky.

Other uses of applets are still being contemplated. For example, the income tax Web site could embed applets in Web pages that can be downloaded with the Web page to the client. The applet could then open up a spreadsheet where the user could enter all his tax data for the year, such as earnings, deductions, etc. The applet could then calculate the taxes based on the figures, and when it is final, the user could upload the spreadsheet back to the income tax site. This can significantly reduce the manual processing and paperwork involved for such purposes. Applets can be used in such e-commerce applications in future. Apart from this, applets can be used as explained in the chapter on Web architectures.

Scanned by CamScanner

2. Since the client computer takes the responsibility of executing the program, the Web server is relieved of this job. This reduces the burden on the part of the Web server. Recall that this is in contrast to the dynamic Web pages. In the case of dynamic Web pages, the program is executed at the Web server. So, if many users are accessing it, the Web server might not be able to serve all the requests very fast. In the case of active Web pages, this is clearly the client's responsibility.

WHEN NOT TO USE ACTIVE WEB PAGES

Active Web pages are mainly useful for client-side animation, as explained earlier. However, they are not useful when server-side programming is important. Server-side programming is useful for business rules checking, validations against some databases (e.g. referential integrity) unlike only local validations for which the client side scripting can be used, database operations, etc.

For instance, when a user enters his id and password, active Web pages cannot be used because these details have to be validated against a database of valid ids and passwords stored on the server. It is very important to understand the difference between dynamic and active Web pages. Dynamic Web pages are mainly used for server-side processing (although they allow client-side scripting for basic validations, etc.) whereas active Web pages have to be executed on the client browser entirely.

However, the main processing must always be done at the server side. For example, access/changes to the databases, validation routines (against databases), etc. must be executed on the server for reasons of security and bandwidth. However, in addition, the dynamic Web pages might add small client-side code for screen validations (e.g. *only 3 items can be selected*), for which a trip to the server is not desirable. When the dynamic web page is requested for by the browser, it is executed and the resulting HTML code along with this client side script (in JavaScript/VBscript, etc.) as it is, is sent back to the client. At the client side, the browser interprets the HTML code and interprets the script.

Active Web pages are mainly used for client-side execution of code, e.g. applets.

LIFECYCLE OF JAVA APPLETS

An applet is a windows-based program (note that the term *windows* here should not be confused with Microsoft's Windows operating system; it actually means the windows that we see on the screen). Applets are event-driven—similar to the way an operating system has Interrupt Service Routines (ISR). An applet waits until a specific event happens. When such an event occurs, the applet receives intimation from the Java Virtual Machine (JVM) inside the browser. The applet then has to take an appropriate action and upon completion, give control back to the JVM.

For example, when the user moves the mouse inside an applet window, the applet is informed that there is a mouse-move event. The applet may or may not take an action, depending on the purpose it was written for and also depending on its code.

Here are the typical stages in the lifecycle of an applet:

1. When an applet needs to be executed for the first time, the *init ()*, *start ()* and *paint ()* methods of the applet are called in the said sequence.
   (a) The *init ()* method is used to initialize variables or for doing any other start up processing. It is called only once during the lifetime of an applet.
   (b) The *start ()* method is called after *init ()*. It is also called to restart an applet after it has been stopped. Whereas *init ()* is called only once, *start ()* is called every time the Web page open containing the applet is displayed on the screen. Therefore, if a user leaves a Web page open and comes back to it, the applet resumes execution at *start ()*.

   (c) The *paint ()* method is called each time the applet's output must be redrawn. This can happen for a variety of reasons. For instance, windows of other applications can overwrite the window in which the applet is running, or the user might minimize and then restore the applet window. It also gets called when the applet begins execution for the very first time.
2. The *stop ()* method is called when the user leaves the Web page containing the applet. This can happen when the user selects or types the URL of another Web page, for instance. The *stop ()* method is used to suspend all the threads that are running for the applet. As we have seen, they can be restarted using the *start ()* method if the user visits the Web page again.
3. The *destroy ()* method is called when the environment determines that the applet needs to be removed completely from the client's memory. This method should then free all the resources used by the applet.

As in the case of a servlet, an applet need not use all these methods. The applet can use only the methods that are useful to it—others get inherited from the various Java classes anyway, and need not be overridden. So, one applet may use just the *paint ()* method and leave everything else to the other default methods that are inherited from the applet's super classes.

Let us take a simple applet example. Suppose our Web page contains the following code for executing an applet, as shown in Figure 10.3.

```
<HTML>
...
<applet code="TestApplet" width=200 height=60>
</applet>
...
</HTML>
```

Fig. 10.3 HTML Page Containing an Applet

As we know, when the Web server sends this HTML page to the client, it also sends the bytecode of an applet named *TestApplet* along with it.

Let us take a look at the applet's code, as shown in Figure 10.4. The applet sets the background colour to cyan, the foreground colour to red and displays a message that shows the order in which the *init ()*, *start ()* and *paint ()* methods of an applet get called.

```
// As before, import statements are used to add the standard ready-made Java classes to your code.
import java.awt.*;
import java.applet.*;

public class TestApplet extends Applet
{
        String msg;

        // public and void are Java keywords and can be ignored for the current discussion.
        public void init ()
        {
                msg = "*** In the init () method ***";
        }

        // Initialize the string to be displayed.
        public void start ()
        {
                msg += "*** In the start () method ***";
        }
```

Scanned by CamScanner

```
        // Display string in the applet window. Note that we have a parameter of the type Graphics to
        // this method. This is a ready-made object that contains various graphics-related methods.
        // For example, you can draw a window using a method of the Graphics class.
        public void paint (Graphics g)
        {
                msg += "** In the paint () method **";
                g.drawString (msg, 10, 30);
        }
}
```

Fig. 10.4 Applet Code

When a client browser requests for this HTML page, the bytecode of the above applet would also be sent to the browser. There, the above code would execute.

The *init ()* method would be called first. It would set the variable *msg* to a message ** *In the init () method* **.

Next, the *start ()* method gets called. It adds a string ** *In the start () method* ** to the *msg* variable. Thus, *msg* variable now contains ** *In the init () method* ** ** *In the start () method* **. Finally, the *paint ()* method gets called. It adds its own message to the msg variable, making it ** *In the init () method* **** *In the start () method* **** *In the paint () method* **. It then prints it, using the standard *drawString ()* method provided by Java. Here, 10 and 30 indicate the pixel coordinates of the screen on which the value of *msg* is to be displayed.

The *stop ()* and *destroy ()* methods are not used in this simple applet as they are not required here. The output of the applet would be:

** In the init () method **** In the start () method **** In the paint () method **

This is shown in Figure 10.5.

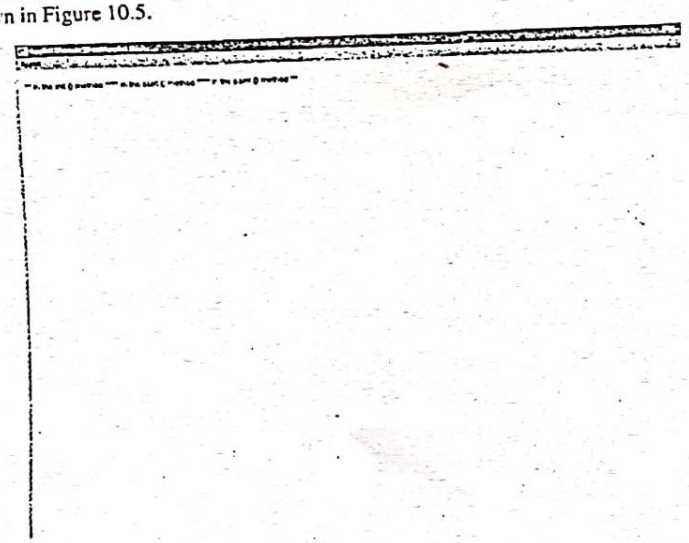


Fig. 10.5 The Output of the Applet

The drawback of applets is that they make the overall execution slow. First, they need to be downloaded from the Web server, and then interpreted by the JVM installed in the Web browser. Therefore, the key is to keep them as small as possible. Applets do not have an exact equivalent in Microsoft technologies— however, Microsoft's ActiveX controls come the closest, which we shall discuss now.

ACTIVEX CONTROLS

ActiveX controls (also known as ActiveX objects) are similar to the Java applets in the sense that they are also tiny program that can be downloaded from the Web server to the Web browser, and executed locally at the browser. However, there are two major differences between an applet and an ActiveX control, as evident in the following discussion:

1. An applet has to go through many security checks (for example, an applet cannot write to the hard disk of the browser computer), but an ActiveX control can actually write to the local hard disk. This makes its behaviour suspect, although it can offer richer functionality as a compensation for this.
2. An applet gets downloaded every time it is accessed. This means that if a user accesses a Web page containing an applet, it gets downloaded to the user's browser and executes there. When the user closes the browser session, the applet is removed from the user's computer because the applet is stored in the main memory of the client computer during it's execution. In contrast, when downloaded, ActiveX controls are stored on the hard disk of the client machine. They remain there even when the browser is closed. Therefore, when the user accesses the same Web page containing the same ActiveX control, the ActiveX control from the client's computer is used, and is not downloaded once again from the server. (Note: New technologies may allow persistent applets, soon).

ActiveX, as mentioned, is Microsoft technology. Therefore, ActiveX control can run only on Internet Explorer browser. The reason for it being Microsoft-specific is again the Windows registry. All ActiveX controls must be recorded in the registry of the operating system that the Web server is running. This means that the Web server must run on an operating system that supports the concept of registry, i.e. Windows. We shall not discuss ActiveX further, since the conceptual framework is similar to applets.

One more point needs to be noted. These days, the concept of code signing has gained prominence. In simple terms, the organization which develops the program code, declares (digitally) that it has developed a particular piece of code, and the person who has downloaded it (in the form of applets or ActiveX controls) can trust it not to perform any malicious actions. For example, an applet coming from Sun Microsystems could declare that the applet is developed by Sun Microsystems, and that the user can trust it not to do any wrongdoings. Moreover, such signed applets or signed ActiveX controls can actually have more privileges than unsigned applets or ActiveX controls (e.g. that can perform disk operations). Of course, a signed applet can still contain malicious code, the only advantage here is that you know where this malicious code came from, and take an appropriate action.

The war between client side technologies is going to become hotter, as Microsoft has decided to remove Java support from Version 6 of its popular browser, Internet Explorer. This means that applets cannot execute inside Internet Explorer from Version 6, unless the user installs the Java Virtual Machine (JVM) plug-in by downloading it from the Internet—it would not be done automatically, anymore. Furthermore, as Internet Explorer is gaining in popularity, and as people realize that applets can make the downloading and processing time slower, it is appearing that ActiveX controls, or some other client side technology, if and when it appears on the scene, will become the key to active Web pages.

Scanned by CamScanner

8. Describe the various keywords of Java.
9. How is FrontPage used as Interactivity tool?
10. Write shortnotes on :
    JavaScript Events
    JavaScript Operators
    JavaScript Assignments
    JavaScript Comparisons
    Common Gateway Interface

Answers to Objective Type Questions

Multiple Choice Questions

1. b	2. c	3. c	4. b
5. b	6. b	7. b	8. c
9. a			

True/False Questions

1. T	2. F	3. T	4. T
5. T	6. T	7. T	8. T
9. T	10. T		

Matching Columns

1. e	2. d	3. a	4. f
5. b	6. c		

Fill in the Blanks with Appropriate Words

1. e	2. f	3. a	4. b
5. c	6. d		

Recommended Readings

1. Professional Web Design (Techniques and Templates) *by Clint Eccher.*
2. HTML & JavaScript for Visual Learners *by Chris Charuhas.*
3. Keeping Ahead—Java 2 *by Benjamin Aumaille.*
4. Keepng Ahead—JavaScript and VBScript *by Banjamin Aumaille.*
5. Front Page 2002 from A to Z *by Heather Williamson.*
6. Front Page 2000 for Visual Learners *by Chris Charuhas.*

—Published by Firewall Media (An imprint of Laxmi Publications Pvt. Ltd.)

12 Multimedia and Graphics

INTRODUCTION

Internet and to some extent the Web pages, are the excellent examples of multimedia. They make the full use of graphics, sound and video in such a way that the site makes a very interesting visual experience. All this is done using the various options available on the computer only. You have the various graphics in the form of clips, sound in the form of various wavesound files. Now a days even the video clips are also available free of charge, which can be used while preparing the web site. Since Internet is also an interactive media. You can share these clips among others. While using video conferencing you can exchange these video clips.

Using your computer and Internet together have made you to listen to live to concert being broadcasted somewhere on the Internet. And you can even have live video conference with people from any where in the world. For all this you do not need specialized hardware and software. The internet allows you to create remarkable, online multimedia content, combining animation, sound and programming using VRML and Micromedia's Shockwave, to name a few, there are others too.

Audio Files on the Internet

Most of the time you would probably not use sound or music on the web site, but there are sites which would make the best use of them. There are many kinds of audio files you will be able to find on the Internet. All these files have been digitized so that a computer can play them. Some of the extensions of the sound files are .WAV and .AU. To play those files, you will first have to download them and then have audio player software play them on your computer. Some browsers such as NetScape Navigator and Internet Explorer have these built in software players. Some of the other audio formats are discussed next.

RMF (Rich Music Format): It is a new audio format. It allows you to interact with music in many ways. For example, you can use RMF to alter the music you are listening to from the Internet by eliminating some instruments. You can even alter the music's tone and pitch. It is similar to word processor where you play with words, here you play with all the constituents of the music file. Since the sound files tends to be long, you need to download them first which may take as long as 15 minutes to half an hour.

Streaming Audio: It handles audio in a better way. Using this, you do not have to wait until the entire audio file is downloaded to play it. Instead, you listen to the audio while it downloads to your computer. RealAudio is one popular audio streaming technology that can help you to play the file without fully downloading the entire sound file.

Working on RealWorld Streaming: As soon as your Web browser clicks on a link to a Real Audio sound clip on a home page, the link does not lead directly to sound file. Instead, your browser contacts the Web server, which sends back to your browser a file called a RealAudio metafile. The metafile is a small text file that has the true location – the URL – of the RealAudio sound file you want to play and also has instructions that tell your browser to launch the RealAudio sound player, which is required to play the clip. The metafile launches the RealAudio sound player, which contacts the URL contained in the metafile. The URL different RealAudio server designed to deliver RealAudio sound clips. The

Scanned by CamScanner

RealAudio server and the RealAudio sound player "talk" to each other so that the server knows at what speed the user is connected to the internet. If the connection is a low-speed connection, a smaller RealAudio file is sent that contains less data. This will be a file of lesser quality than a file sent via a high-speed connection. If a high-speed connection is used, a larger, higher-quality sound file is sent. This will provide for better sound quality. The RealAudio clip is compressed and encoded. If the file was not compressed, the sound file would be too large and would take too long to send and the played.

The clip is sent in IP packets using the User Datagram Protocol (UDP) instead of the Internet's TCP. The packets are sent to a buffer on the receiving computer. Once the packets are exceed the capacity of the buffer, they are sent to the RealAudio player, which then plays the sound file. RealAudio allows you to jump ahead or back in a sound or music clip. When you move to a different place in the clip, the RealAudio player contacts the server and tells it to start sending the film from that new place in the clip.

Video Conference

Using this, as mentioned earlier, you can not even talk to other people on the net, but even see them in person. This is known as whiteboard application. Following three are the different technologies that make it possible to achieve the live video conference or Whiteboard applications.

MBone (Multicast Backbone) Technology: This is a special Internet high-speed backbone capable of sending vast amount of information. Many video transmissions – specially live ones – are sent across the MBone because of its high bandwidth.

Streaming Video Technology: The Steaming video technology solves the problem by first compressing the video file so it is much smaller as it is transmitted across the Internet Secondly, steaming video lets the receiving computer start playing the video while the file is being transmitted. So, if you receive a streaming video file, you watch the video as you receive it. No waiting is needed for the entire file to download. But you will need a special player to watch the video.

Videoconferencing

This technology lets you use your computer to have live video conferences across the Internet. Videoconferencing, is done live, although the technology can also be used to broadcast taped videos as well. National aeronautics and Space Administration of USA uses this technology to broadcast live from the Space Shuttle as well as to broadcast taped videos about space exploration.

VIRTUAL REALITY ON THE INTERNET

This is among the latest technology used on the net. It allows you to walk through the 3-dimensional space on the net. This is all what we call as virtual reality (VR in short). You will find many virtual worlds you can explore on the Internet. You will be able to walk through a giant computer, explore bizarre art galleries, visit outer space, go to the sites of what seem like ancient ruins, explore inside the human brain. Virtual words are created using a computer language called Virtual Reality Modeling Language (VRML), this is discussed next. This language instructs computers on how to build 3D geometric objects. VRML is a 3D equivalent to HTML, but with more capabilities than just putting a 3D shop on your Web page. A VRML world is created by an ASCII text file containing VRML language commands. Graphic files can be added to this world as well. Because the virtual world is only an ASCII file, with few graphic files, it can be downloaded quickly to your computer from the Internet. When a virtual world is created, it is posted on an Internet server. When you want to visit that world, you will type in its URL or click on a link to it, just as you do to visit any other location on the WWW. To display the virtual world, you will need a program that is able to display the world – either a separate virtual reality browser or a plug-in player that configures itself to your normal Web browser.

VRML (VIRTUAL REALITY MODELING LANGUAGE)

As mentioned above it allows you to create 3D worlds to describe the geometry of a scene. VRML files are also much smaller than graphic files. VRML files are text files that contain instruction for drawing the VRML world. These files end in a .WRL extension. After a file is created, it is posted on a Web server. There are now two official versions of VRML. However, unlike HTML versions, there is no backward compatibility. You need to make a choice of the version that you are going to use.

Browsers which support VRML are:

NetScape and Live3D: Live3D is distributed as a standard part of browsers from Navigator version 2 onward. It offers most of the standard feature that you find in the other browsers.

Microsoft and Active VRML: Microsoft markets the Active VRML against VRML 2.0 and it does have the potential to read VRML 1.0 files. However, Active VRML is not a true VRML.

Intervista's WorldView: WorldView is available either as a stand-alone program or as a plug-in to NetScape Navigator as well as for Microsoft's Internet Explorer. WorldView has one advantage over Live3D in that it can run in any of your favorite browser environment. One of WordView's nice feature is its ability to navigate your own camera positions while navigating a world and then return back to them. The rendering is very smooth, but it is not very accurate for handling mesh objects.

Sony's CyberPassage: Cyber Passage was the first VRML 2.0 browser to be available for the Internet. CyberPassage only operates as a stand-alone product, CyberPassage supports Java for scripting, which makes it much more extendible.

VRML Modeling Tools

The easiest and quickest way to construct a VRML world is to use a modeling package. Modelers can be divided into three categories:

VRML Exports from Traditional Tools: One of the first ways that complex VRML models were constructed was with non-VRML modeling tools like AutoDesk's 3D Studio. These tools export to a standard format like DXF, which then has a third-party converter like wcvt2pov change that into VRML. This provides a very quick working base for many VRML worlds.

With Internal support through plug-in exporters or native exporters: Today a number of these tools contain plug-in exporters that can automatically produce VRML output files. One of the most widely used one is Syndesis Corporation's Interchange, Interchange acts as a plug-in file exporter for most popular modeling tools which supports VRML 2.0 export. Another exporters comes from Kinetix which plugs into 3D Studio.

Dedicated VRML Modelers: One of the most interesting effect of VRML is the number of software companies that have released separate, dedicated VRML authoring tools, which are based on their non-VRML modeling tools. There are a wide range of tools available, ranging from those that barely hide VRML from you to those that can create any sort of file format. VRealm Builder is one of the most popular tools used in the VRML community today. VRealm Builder presents a four-view layout on the right side while you see the structure of the VRML file graphically produced on the left.

Setting up the Web Server

VRML files end with the extension .wrl. Sometimes you may see them end with .wrl.gz. These are files that have been compressed to make them smaller. Both kinds are legal and the Web browser will handle them properly. On the server end, you need to make sure that your Web server is configured for the

Scanned by CamScanner