UNIT: 2 RMI & SERVLET

CS-25 Advanced Java Programming (J2EE)

- **RMI AND RMI ARCHITECTURE**
- **STUB AND SKELETON**
- **DEVELOPING AND EXECUTING RMI APPLICATION**
- **SERVLET INTRODUCTION**
- **ARCHITECTURE OF A SERVLET**
- **SERVLET API (JAVAX.SERVLET AND AVAX.SERVLET.HTTP)**
- **SERVLET LIFE CYCLE**
- **DEVELOPING AND DEPLOYING SERVLETS**
- **HANDLING SERVLET REQUESTS AND RESPONSES**
- **READING INITIALIZATION PARAMETERS**
- **SESSION TRACKING APPROACHES (URL REWRITING, HIDDEN FORM FIELDS, COOKIES, SESSION API)**
- **SERVLET COLLABORATION**
- **SERVLET WITH JDBC**

: ASSIGNMENT 2:

1. **RMI stands for ?**
2. **_____ defines how the client request to the server for the remote objects and how the server processes for that request?**
3. **_____ is used to get the initialization parameter lll servlet.**
4. **Which exception indicates that a servlet problem has been occurred ?**
5. **Explain Stub class.**
6. **Explain lookup() and rebind() method of RMI.**
7. **What is RMI ? Explain RMI Architecture.**
8. **Which technology which allows the client to Remote object communication and object to object communication between different JVM ?**
9. **_____ is server Side proxy that communicates with the stub.**
10. **_____ Package contains the generic interface and classes that are implemented & extended by all servlets.**
11. **_____ Returns a Print. Writer that can be used to write character data to the response.**
12. **What is Servlet? Explain Servlet Life Cycle.**
13. **Explain Session Tracking Approaches.**
14. **Define task of stub and Skeleton.**
15. **How to read initialization parameter in Servlet? Explain.**
16. **What is a cookie? Explain in brief.**
17. **How to create Servlet using Servlet interface? Explain with example.**
18. **Explain Deployment Descriptor.**

RMI

Remote Method Invocation (RMI) is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

Stub Object: The stub object on the client machine builds an information block and sends this information to the server.
The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

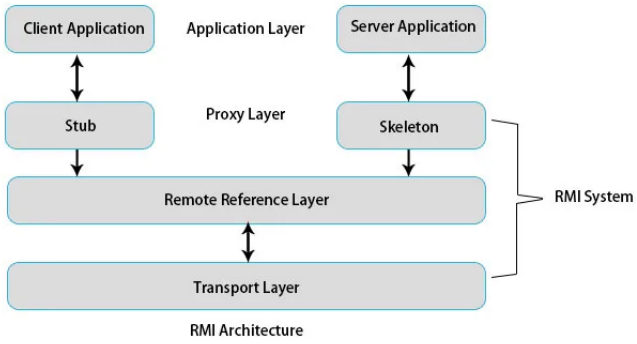
Skeleton Object: The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

- It calls the desired method on the real object present on the server.
- It forwards the parameters received from the stub object to the method.

Overview of RMI Application
RMI Application is divided into two parts Client and Server. Server program creates some remote object and makes the references to them and waits for the client request to invoke methods on the Remote objects.
A client application get a remote reference to one or more Remote object from the server and invoke the methods on them. RMI provides this mechanism by which server and client communication and pass information back and forth. such application is say as a Distributed Object Application.
Distributed Object Application has to include three things:

1. **Locate Remote Object**: An Application can use various mechanisms to obtain references to remote objects. For example, an Application can register its remote objects with RMI registry. Alternatively an Application can pass and return remote object references as part of other remote invocations.
2. **Communicate with remote objects**: All communication between remote objects is handled by RMI. Remote communication look similar to regular Java Method invocation.
3. **Load class definition for object that is passing around:** Because RMI enables objects to be passes back and forth, it provides mechanism for loading an object's class definition as well as for transmitting an object's data.

RMI Architecture



RMI Architecture

The client application and server application are the respective JVMs of the client machine and server machine. In the RMI application, we write two programs: the client program, which resides on the client, and the server program, which resides on the server machine.

Application Layer: consists of client side and server side java programme with remote objects. Here high level calls are made in order to access and export remote objects.Client can access remote methods through an interface that extends java.rmi.Remote When we want to define a set of methods that will be remotely called they must be declared on one or more interface that should extend java.rmi.Remote**.** Once the methods described in remote interface have been implemented, the object must be exported using Unicast Remote Object class of the java.rmi.server package. then the application will register itself with registry and it is used by client to obtain references of the objects. On the client side a client simply request a remote object from either registry or remote object whose references has already been obtained.

Proxy Layer Stub/skeleton layer is responsible for listening to the remote methods calls made by client and redirecting them to the server. This layer consists of stub and a skeleton layer. The stub and skeleton are created using RMI Compiler (RMIC).Stub is client side proxy representing the remote objects and communicates the method invocation to the remote object through a skeleton that is implemented on the server .skeleton is server Side proxy that communicates with the stub.it makes call to remote objects. So it is also known as proxy layer.

Usage of rmic:

Rmic<options><class names>

Option	Description
-keep / -keepgenerated	Returns the generated –java source files for stub ,skeleton,and/or classes and writesthem to the same directory as the .class files.
-v1.1	To create stub /skeleton for JDK 1.1 stub protocol version.
-v1.2	To create stub /skeleton for JDK 1.2 stub protocol version.
-g	Generate debugging information
-depend	Recompile out of date files recursively
-nowarn	Generate no warning
-verbose	Output message about compiler what is doing
-classpath<path>	Specify where to find input source and class files.
-d<directory>	Specify where to place generated class files.
-J<runtime flag>	Pass arguments to the java interpreter

The Remote Reference layer

It is an interface between proxy layer and Transport layer, it handles actual communication protocol. RRL interprets the references made by the client to the remote object on the server. It present on clients and server. RRL on client receive requests for methods from stub, the request is then transferred to the RRL on server side.

RRL used to:

- Handle replicated object .once the feature is incorporated into RMI system; the replicated object will allow simple dispatch to other programs.
- It is responsible for establishing persistence and strategies for recovery of lost connections.

The Transport Layer

The Transport layer takes path of the communication or layer between client & server. It receives a request from client side RRL and establishes a connection with the server through server side RRL. Transport layer has following responsibilities:

- It is responsible for handling actual machine to machine communication and default communication takes place through TCP/IP.
- It creates stream that is accessed by RRL to send and receive data from other machines.
- It sets up the connection to the remote machines.
- It manages the connections.
- It monitors the connections. To make sure that they are live.
- It listens for connections from the machines.

STUB AND SKELETON

STUB-Client side proxy

- It presents the same remote interface as the object of the server.
- It works with JVM on client machine to serialize any arguments to a remote method call and sends this information to the server machine.
- The stub receives any results from remote method and returns it to the client.

SKELETON Server Side proxy

- It receives the remote method call and any associated arguments. It works with JVM and RMI system on server to deserialize any arguments for remote method call.
- It invokes the appropriate method in the server.
- It receives any return value from this method call and works with JVM and RI system on server to serialize the return value and send information back to the client.

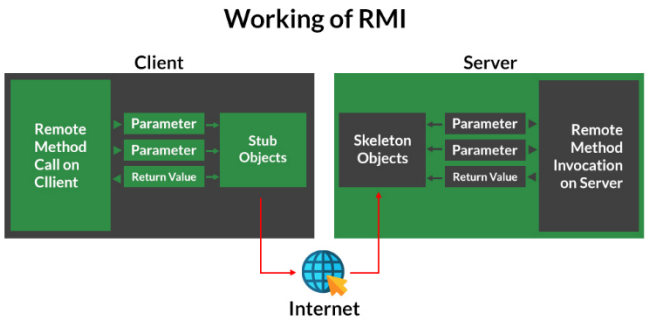
RMI Package:

The Java.rmi.* is imported in all the program in order to communicate with server. The Naming Class of RMI package has declared various methods.

1. **bind ()** It is used to binds a specified remote name to remote object. Having two parameter (1)name of object and(2) object itself
   Naming.bind("Hello",greeting);
2. **unbind()** It removes the binding object with remote object. Having single parameter.
   Naming.unbind("Hello");
3. **rebind()** it is used to rebind name with remote object. Having two parameter (1)string message and(2)new object name
   Naming.rebind("Hello",msg);
4. **lookup()** It returns name of the remote object specified by the RMI URL.
   Naming.Lookup(rmi://hostname/objectname);
5. **list()** It returns the string array with name of objects bound in the registry .Having one parameter
   Naming.list(URL);

Working of RMI/ Developing RMI Application

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows.



These are the steps to be followed sequentially to implement Interface as defined below as follows:

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the 4. implementation class using rmic (RMI compiler)
4. Start the rmiregistry
5. Create and execute the server application program
6. Create and execute the client application program.

Step 1: Defining the remote interface

The first thing to do is to create an interface that will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

```
// Creating a Search interface
import java.rmi.*;
public interface Search extends Remote
{
    // Declaring the method prototype
    public String query(String search) throws RemoteException;
}
```

Step 2: Implementing the remote interface

The next step is to implement the remote interface. To implement the remote interface, the class should extend to UnicastRemoteObject class of java.rmi package. Also, a default constructor needs to be created to throw the java.rmi.RemoteException from its parent constructor in class.

```
// Java program to implement the Search interface
import java.rmi.*;
import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject
                         implements Search
{
    // Default constructor to throw RemoteException
    // from its parent constructor
    SearchQuery() throws RemoteException
    {
        super();
    }

    // Implementation of the query interface
    public String query(String search)
                        throws RemoteException
    {
        String result;
        if (search.equals("Reflection in Java"))
            result = "Found";
        else
            result = "Not Found";

        return result;
    }
}
```

Step 3: Creating Stub and Skeleton objects from the implementation class using rmic

The rmic tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects. Its prototype is rmic classname. For above program the following command need to be executed at the command prompt rmic SearchQuery.

Step 4: Start the rmiregistry

Start the registry service by issuing the following command at the command prompt start rmiregistry

```java
// Java program for server application
import java.rmi.*;
import java.rmi.registry.*;
public class SearchServer
{
    public static void main(String args[])
    {
        try
        {
            // Create an object of the interface
            // implementation class
            Search obj = new SearchQuery();

            // rmiregistry within the server JVM with
            // port number 1900
            LocateRegistry.createRegistry(1900);

            // Binds the remote object by the name
            // geeksforgeeks
            Naming.rebind("rmi://localhost:1900"+
                          "/geeksforgeeks",obj);
        }
        catch(Exception ae)
        {
            System.out.println(ae);
        }
    }
}
```

Step 5: Create and execute the server application program

The next step is to create the server application program and execute it on a separate command prompt.

The server program uses createRegistry method of LocateRegistry class to create rmiregistry within the server JVM with the port number passed as an argument.

The rebind method of Naming class is used to bind the remote object to the new name.

Step 6: Create and execute the client application program

The last step is to create the client application program and execute it on a separate command prompt . The lookup method of the Naming class is used to get the reference of the Stub object.

```java
// Java program for client application
import java.rmi.*;
public class ClientRequest
{
    public static void main(String args[])
    {
        String answer,value="Reflection in Java";
        try
        {
            // lookup method to find reference of remote object
            Search access =
                (Search)Naming.lookup("rmi://localhost:1900"+
                                      "/geeksforgeeks");
            answer = access.query(value);
            System.out.println("Article on " + value +
                               " " + answer+" at GeeksforGeeks");
        }
        catch(Exception ae)
        {
            System.out.println(ae);
        }
    }
}
```

Note: The above client and server program is executed on the same machine so localhost is used. In order to access the remote object from another machine, localhost is to be replaced with the IP address where the remote object is present. Save the files respectively as per class name as Search.java , SearchQuery.java , SearchServer.java & ClientRequest.java

Important Observations:

RMI is a pure java solution to Remote Procedure Calls (RPC) and is used to create the distributed applications in java.

Stub and Skeleton objects are used for communication between the client and server-side.

Servlet

Initially in the early days CGI was used to write server side scripting to create dynamic web pages with most of the languages like C, C++ and also with JAVA. But when user request for something server need to create different process for each and every HTTP request of client. After giving response to the client and executing the CGI script server should release the system resources .It takes much execution time at the cost of performance. It was also expensive to open and close database connection for each client. In addition it is platform dependent. To overcome these limitations servlet was introduced by sun Microsystem.

- The primary reason for development of servlet is limitation of html because can be used to create only static pages and nothing more but technology and usage of www .grew people's demand for more than static pages (dynamic pages).
- Today of web page should show forms accepts input from the users and access the databases, it can store, receives and process the data, re-act the need of the user etc.
- A servlet is generic server extension java class that can be loaded dynamically and expand functionality of the server.
- A servlet is similar to the script except that the servlet runs under the JVM on the server. So it is safe and portable and it does not require the support for the java on a browser.
- Imagine the servlet as server side component. Servlets are servers what the applets are to the browsers servlet code can be downloaded into running server to extend its behaviours to provide new or temporarily services to network clients.
- A servlet is dynamically loaded module that services request from the web-browser. If run entirely inside the JVM.
- Servlet can be used for any number of web-related applications for example developing an e-commerce web site becomes one of the most common usage for the java servlet.

Benefits of Servlets

1. **Performance:** Its performance is better than CGI because it is not necessary to create the separate process to handle each HTTP request, it executes within the address space of web server.so it saves the memory resources and make performance better than CGI.
2. **Platform Independence**: A number of web-server from different vendor supports Servlet API. It is written in Java so program developed for this API can be run on any environment without recompilation.
3. **Security**: Servlet run under the JVM and it is secured .It is also using java security manager .it is server side component, so it inherits the security provided by the web server.
4. **Efficient Servlet Initialization Code:** It is only executed for the first time when the web server loads it. Once the server has been loaded, it is only a matter of calling the service to handle the new request. So it is efficient for each & every request.
5. **Persistent:** Servlets can maintain the state between requests. Once the servlet has been loaded, it takes its state in the memory until the last request will arrive. Taking the advantage of persistent characteristic of servlet will improve the performance of applications.
6. **Portable:** Servlets are developed using java therefore they are portable.
7. **Robust:** Servlets are developed with the jdbc. So they provide maximum robust solution with the help of java's well defined exception hierarchy, garbage collection, network support, file support etc.
8. **Extensible:** Servlets are developed in pure OOP like java. So that can be extended & polymorphed into new object which sets your needs. The best example is of search engine tool which you can put on your each & every page which will be used to search different things on different pages.
9. **Flexible:** Servlets are quite flexible because it can be added to a static page using the servlet tag <servlet>.
10. **Secure:** Servlets are run on server side & gives us the facility & security internally provided by the web servlet.

Servlet Architecture/Servlet API

There are two packages that make up the servlet architecture. They are as follows:

1. The **javax.servlet package** contains the generic interface and classes that are implemented & extended by all servlets.
2. **The javax.servlet.http package** contains the classes that are extended while creating the http specific servlet.

javax.servlet package having following classes and interfaces.

1. **Servlet Interface**: It is having methods which define life cycle of servlet.
2. **ServletConfig Interface**: It provides the basic or initialization parameters of the servlet.
3. **ServletContext Interface**: It provides the runtime environment to the servlet. It also logs the event using log ().
4. **GenericServlet Class**: It implements the Servlet, ServletConfig, Serialization interface.
5. **ServletRequest Interface**: It is used to read the client request.
6. **ServletResponse Interface**: It is used to write the response data.
7. **ServletException Class**: It defines the servlet occurred error.
8. **UnavailableException Class**: It defines that servlet is temporarily or permanently not available.
9. **ServletInputStream Class**: It provides input stream o read the data from the client request.
10. **ServletInputStream Class**: It provides output stream to write the data as request to client.
11. **SingleThreadModel interface**: It provides the mechanism to make servlet thread safe.

Javax.servlet.http package having following classes and interfaces.

1. **HttpServletRequest Interface**: It allows the servlet to read data from HTTP request.
2. **HttpServletResponse Interface**: It allows the servlet to write data to an HTTP response.
3. **HttpServlet Class**: It provides the methods for handling HTTP request and HTTP response.
4. **Cookie Class**: It used to store the state information at client side or on the client machine.
5. **HttpSession Interface**: It used to create session and provides the way of reading and writing the session and accessing the information related to session.
6. **HttpSessionBindingListener interface**: It informs the object that it is bound or unbound to the session
7. **HttpSessionEvent class**: It wraps up the session change events.
8. **HttpSessionBindingEvent Class**: It defines that when session should be bound or unbound with the object value and its attribute.

Implementing Servlet

To implement servlet in the program there are four ways .we can use anyone from following classes or interface to create servlet program.

- **Servlet Interface**
- **GenericServlet Class**
- **HttpServlet Class**
- **SingleThreadModel Interface**

Servlet Interface:

All the user defined servlet should inherit the servlet interface having following methods.

1. **init()**

   It is called by servlet container to initialize the servlet itself. It is called only once when servlet engine loads the servlet.It finishes its work before service () starts for request and response. It's having one parameter with it, which is an object of ServletConfig interface to start up configuration and initialization parameters which returned through getServletConfig ().If any error occurs it throws an Unavailable Exception. Basic format of init () is as follows:

   public abstract void init(ServletConfig config) throws ServletException

2. **service()**

   It used to read client request and write the response data to the client.It passes two parameters first the object of ServletRequest interface and second is the object of ServletResponse interface. The request objects having information related to client request and parameters provided by the client and response object contains the information which is sent back to the client based on request. This method is called after the initialization is completed. If any error occurs during the request and response or input and output it throws IOException and ServletException. Basic format of service () is as follows:

   public abstract void service (ServletRequest req, ServletResponse res) throw ServletException, IOException

3. **destroy():**

   This method is called by servlet Container automatically when objects or resources like memory, thread etc need to be destroyed or servlet stops it execution. Basic format of destroy() is as follows:

   public abstract void destroy ()

4. **getServletConfig():**

   It returns the ServletConfig object which contains initialization parameter and startup configuration of servlet.It passed as a parameter in the init().Basic format of getServletConfig() is as follows

   public abstract ServletConfig getServletConfig()

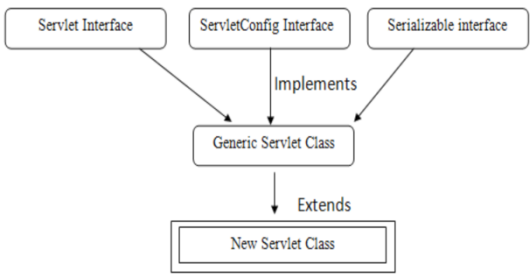
5. **getServletInfo():**

   It returns the basic information related to the servlet such as author name, version, etc. Basic format of getServletInfo () is as follows:

   public abstract void getServletInfo ()

Generic Servlet Class

It belongs to the javax.servlet package. It implements the servlet, ServletConfig, serializable interface in it.It is extended to provide the protocol based things like FTP, SMTP protocol but it is not having a HTTP protocol facility .So most probably with web application we need to extend the HttpServlet Class. Generic servlet class provides the implementation of servlet interface because in most of the classes they are required to use service () to handle request and response. It is having most of the methods which are there with the super interfaces except log().Hierarchy of Genericservlet class as follows:



1. **init()**

It called by servlet container in addition if we override init()with servlet we need to write super.init(config) init to provide reference of config object. Basic format of this method is:
public abstract void init(ServletConfig config) throws ServletException

2. **service()**

It provides the servlet request response facility to each and every servlet should override this method directly writing service () with the program or indirectly using doXXX () of HttpServlet class. Basic format of this method is:
public abstract void service (ServletRequest req, ServletResponse res) throw ServletException, IOException

3. **destroy ()**

It is called by servlet container at the end of the execution. We can override this method in our subclass to do some clean up task, when servlet is taken out of service. Basic format of destroy () is as follows:
public abstract void destroy ()

4. **log ()**

log () is used to write the server log file. it is overloaded method which we can use in two ways. First way log (String message) method writes the servlet name and message to web container log file and other method log (String message, Throwable t) writes the servlet name and the exception stack trace of the given Throwable Exception to web container log file. Basic format of destroy () is as follows:
Public void log (String message)
Public void log (String message,Throwable t)

5. **getInitParameter()**

It is used to get the initialization parameter. If the parameter does not exists it returns null. Basic format of this () is as follows:
public string getInitParameter (String name)

6. **getInitParameterNames()**

It returns the name of initialization parameter names in the form of enumeration. Basic format of this () is as follows:
public Enumeration getInitParameter Names ()

7. **getServletContext()**

It returns the object of ServletContext. Basic format of this () is as follows:
public ServletContext getServletContext ()

8. **getServletName()**

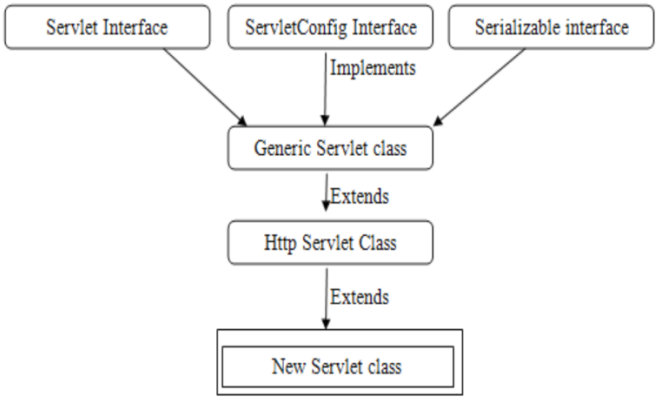
It returns the name of the servlet. Basic format of this () is as follows:
public String getServletName ()

Single Thread Model

In CGI single instance creates multiple threads in order to process mulriple request and response. In the past also a single instance of a servlet creates multiple threads for multiple requests and response. In this process the shared non local variables must be synchronized is implemented at the cost of system performance because thread waits in a queue for the current thread to complete its job.Therefore synchronizing the code increases time to perform a single task,it downs the performance of the system.In certain cases synchronization may not be appropriate method to implement thread safely.so in such a case we need to implement the SingleThreaModel. SingleThreadModel is implemented by SingleThreadModel interface. It ensures that servlet handle only one request at a time. This interface has no methods. If a servlet implements this interface you are guarantee by synchronizing access to single instance of servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet. Note that SingleThreadModel does not solve all thread safety issues. For ex .session attribute and static variable can still accessed by multiple request on multiple request on multiple threads at the same time even when SingleThreadModel Servlets are used. It is recommended that a developer take other means to resolve those issues instead of implementing this interface, such as avoiding the usage of an instance variable or synchronizing the block of the code accessing those resources.

HTTP servlet class

HttpServlet class extends GenericServlet Class. It is commonly used by programmers when developing servlet that receive and process HTTP requests, because it is having HTTP protocol functionality. Hierachrchy of HttpServlet class is shown below



Methods of HttpServlet Class are as follows:

1. **doGet()**

doGet() is called by servlet via service() to handle an Http request. A GET request allows client to sent form data to server. With the get request the form data is attached to the end of the URL sent by the web browser to the server as a query string. The amount of form data that can be sent is limited to the maximum length of the URL. Basic format of doGet() is as follows.

public void doGet(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

2. **doPost()**

doPost() is called by servlet via service() to handle an Http POST request. A POST request allows client to sent form data to server. With the post request, the form data is sent to the server separately instead of being appended to to the URL . This allows a large amount of data to be sent. Basic format of doPost() is as follows.

public void doPost(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

3. **doDelete ()**

doDelete() is called by servlet via service() to handle an Http DELETE request. A DELETE request allows client to remove a document or web page from the server. Basic format of doDelete() is as follows.

public void doDelete(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

4. **doOption()**

doOption() is called by servlet via service() to handle an Http OPTION request. An OPTION determines which Http method the server supports and sends the information back to the client by way of header.Basic format of doOption() is as follows.

public void doOption(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

5. **doPut()**

doPut() is called by servlet via service() to handle an Http PUT request. A PUT request allows client to sent place a file on the server and is conceptually similar to sending the file to the server via FTP. Basic format of doPut() is as follows.

public void doPut(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

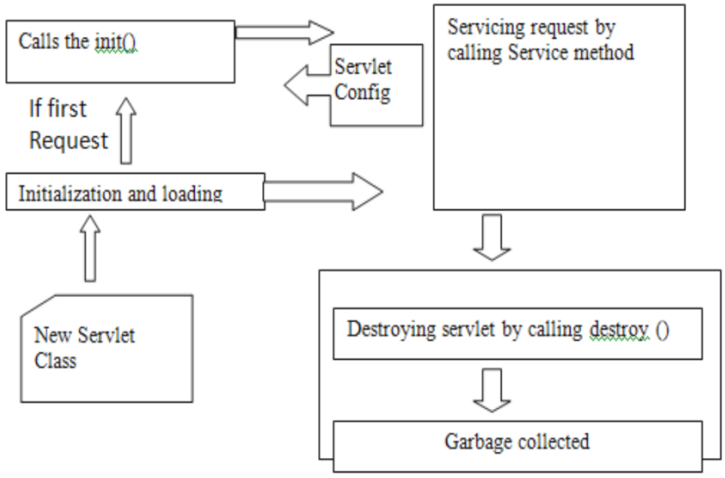
6. **doTrace()**

doTrace() is called by servlet via service() to handle an Http TRACE request. A TRACE request returns the header sent with the TRACE request back to the client. This can be useful for debugging purpose. This method is rarely overridden. Basic format of doTrace() is as follows.

public void doTrace(HttpServletRequest req, HttpServletResponse res)throws ServletException,IOException

Servlet Life Cycle

- Servlets are snippets of Java programs which run inside a Servlet Container.
- A Servlet Container is much like a Web Server which handles user requests and generates responses. Servlet Container is different from a Web Server because it can not only serve requests for static content like HTML page, GIF images, etc., it can also contain Java Servlets and JSP pages to generate dynamic response.
- Servlet Container is responsible for loading and maintaining the lifecycle of the a Java Servlet. Servlet Container can be used standalone or more often used in conjunction with a Web server.
- Example of a Servlet Container is Tomcat and that of Web Server is Apache.



init()

- When the servlet is first created, its init () is invoked, so init is where you put one time setup code. Most of the times your servlet deal only per-request data, and doGet or doPost are the only life cycle method you need. occasionally one want to perform complex setup tasks when the servlet id first loaded,but not repeat those tasks for each request.init() is designed for this case it is called when the servlet Is first created,and not called again for each user request,so it is used for one time initialization,just as with the init() of applet.
- This method is called once when the servlet is loaded into the servlet engine, before the servlet is asked to process its first request.
- The init method has a ServletConfig parameter. The servlet can read its initialization arguments through the ServletConfig object. How the initialization arguments are set is servlet engine dependent but they are usually defined in a configuration file.

service()

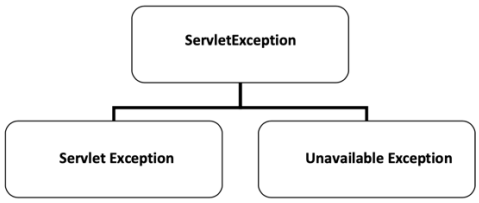
- Each time the server receives a request for a servlet, the server needs a new thread and calls service ().
- The service () checks the HTTP request type (GET, POST, PUT, DELETE etc) and calls doGet, doPut, doDelete etc. as appropriate.
- A GET request results from a normal request for a URL or from an HTML form that has no method specified. A POST request results from an HTML form that specifically lists POST as the methods. If you have a servlet that needs to handle both POST and GET requests identically, you may be tempted to override service directly rather than implementing both doGet and doPost.

destroy()

- The service may decide to remove a previously loaded servlet instance, perhaps because it is explicitly asked to do so by the server administrator or perhaps because the servlet is idle for a long time. Before it does, however, it calls the servlet's destroy method.
- This method gives your servlet a chance to close database connection, halt background threads, write cookie or hit counts to disk ,and perform other such clean-up activities. Be aware ,however, that is possible for the we server to crash.
- So don't count on destroy as the only mechanism for saving state to disk. If your servlet performs activities like counting lists of cookies values that indicate special access, you should also proactively write the data to disk periodically.

ServletException

The javax.servlet defines two exceptions, the first is ServletException which indicates that a servlet problem has been occured and the second is UnavailableException which extends ServletException and indicates that servlet is unavailable.



Servlet Exception

Servlet Exception indicates that a servlet problem has occurred. The general form of Servlet Exception class is:

public class Servlet Exception extends Exception

ServletException has following constructor.

- **ServletException()**: Basic exception constructor
- **ServletException (String msg)**: It creates a new servlet exception with a message.
- **ServletException (String msg, Throwable t)**: It creates a new servlet exception with a message and a wrapped exception.
- **ServletException (Throwable rootCause)**: It creates a new servlet exception with a message and a wrapped exception.
- **getRootCause()**:It returns the exception that caused this servlet exception.

Unavailable Exception

- Unavailable Exception extends ServletException. It defines exception that a servlet throws to indicate that it is permanently or temporarily unavailable. When a servlet is permanently unavailable, something is wrong with it, and it cannot handle request until some action is taken. For Example: a servlet might be configured incorrectly, or its state may be corrupted. The component should log both the error and corrective action that is needed.
- A servlet is temporarily unavailable if it cannot handle request momentarily due to some system wide problem. For example: 3 tier servers might not be accessible, or there may be insufficient memory or disk storage to handle request .A system administrator may need to take corrective action.
- Servlet container can safely treat both type of unavailable exception in the same way. However treating temporary unavailability effectively makes the servlet container more robust.
- General form of unavailable exception is:
  public class UnavailableException extends ServletException

UnavailableException has following constructor.

UnavailableException():Basic exception constructor
UnavailableException (String msg): It creates a new exception with a message that a servlet is permanently Unavailable.
UnavailableException (String msg, int sec): It creates a new exception with a message indicating that a servlet is temporarily Unavailable and giving an estimate of how long it will be Unavailable.

This provides following methods:
getUnavailableSeconds() : It returns the number of seconds the servlet expects to be temporarily available.
getServlet(): It returns the servlet that threw this exception or null if the servlet instance was not provided to the constructor.
IsPermanent(): It indicates that the servlet is permanently unavailable or not.

ServletRequest and ServletResponse

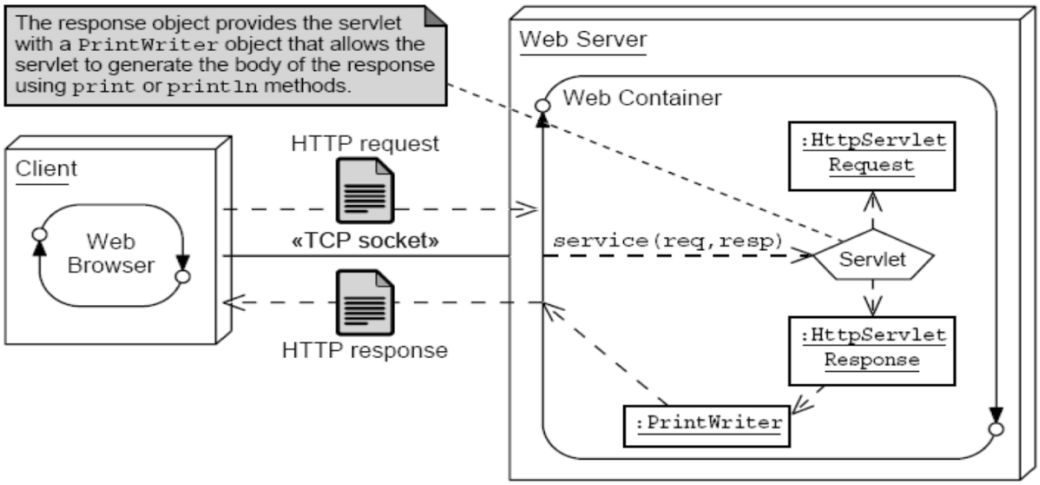


Diagram of How Servlet is execute with Request and Response

The ServletRequest interface is implemented by the server which enables the servlet to obtain the information about the client request.

ServletRequest and ServletResponse :

ServletRequest Class

Methods	**Description**
Void setAttribute(String name, Object obj)	It stores an attribute in the request. Mostly used with Request Dispatcher.
Object getAttribute(String name)	Returns the value of the attribute named name.
Enumeration getAttributNames()	It returns the array of the name of the attributes.
String getParameter(String prm)	Returns the value of the parameter named prm.
Enumeration getParameterNames()	Returns an enumeration of the parameter names for this request.
String[] getParameterValues(String nm)	Returns an array containing values associated with the parameter specified by nm.
String getRemoteAddr()	Returns the String equivalent of the client IP address.
String getRemoteHost()	Returns the String equivalent of the client host name.

ServletResponse Class

Methods	Description
PrintWriter getWriter() throws IOException	Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream () has already been invoked for this request.
void setContentType(String type)	Sets the content type for the response to type.

HttpServletRequest Class

Methods	Description
Enumeration getHeaders()	Returns all the values of the specified request headers as an Enumeration.
Cookies[] getCookies()	Used to obtain the array of cookie that are present in the request.
String getQueryString()	Returns the query string that is contained in the request URL.
HttpSession getSession()	Returns the current session associated with the request.

HttpServletRequest Class

Methods	Description
Void addCookie(Cookie cookie)	Adds cookie to specified HTTP response.
String encodeURL(String url)	It encodes the URL by including session id in it, if encoding is not needed returns URL unchanged.
void sendRedirect(String url) throws IOException	Redirects the client to url.
void setHeader(String name, String value)	Used to set response header with given name and value.
void setStatus(int sc)	Sets the status code for HTTP response SC_MOVED_TEMPORALILY,SC_OK

Session Tracking Approaches

- When a user makes a page requests to the server, the server creates a temporary session to identify that user. So when user goes to another page on the same site the server can recognize that user easily so a Session is temporary small unique connection between a server and the client enabling it to identify that user across multiple page requests or visit to that site.
- Http is a stateless protocol. Each request is independent of the previous one. However, in some application it is necessary to save the information. So that it can be collected from several interactions between a browser and a server. There are 2 important activities in session management.
- **Tracking the identity of a user**: User make multiple page requests to same web application so we need to associate each request with a client identifier, so we can identify a request from the same user.
- **Maintaining user state**: Since there is often data associated with each client request, we will need a way to associate the request data with the user that made the request, and a way to preserve that data across the requests.

It provides following mechanism to track the user state across requests.

1. **URL rewriting**

- To maintain the session state the URL rewriting is the best practice. URL rewriting is based on the idea of inserting a unique id (generated by server) in each URL of the response from the server.
- While generating the response to the first request, the server inserts this id in each URL. When client submit such a request to one such URL, the browser send this id back to the server. The server can identify the id with all requests.
- **Advantage** : User remains unknown and they are universally supported.
- **Disadvantage**: It is tedious to rewrite the entire URL and it only works with dynamically created documents.

2. **Hidden Form Fields**

- The component can include in the html form a field that is not displayed on the form. This field is called hidden field. A hidden field is similar to the other fields of the form. The main thing is the value of hidden field will not be entered by user but it is assigned by the Component.
- The date available in the hidden field will be treated same as the other regular fields. The value of the hidden filed can be abstracted from the form and can be set by the browser as a parameter Example <input type = "hidden" name = "grno" value = "1">
- In the above example, the grno is automated generated field. So the value will be assigned by the Component itself and not by the user. You can use the name grno at any place in the form just like a normal field.
- Remember that hidden fields are useful when the user will be unknown for the particular value of the field. It can be used for three purposes.
- For tracking session as a users move around within site.
- Hidden fields are used to provide predefined input to a server side program
- when a variety of static HTML pages act as a front ends to the same program on
- server.
- Hidden fields are used to store background information in pages that are dynamically generated.

- **Advantages**:
  - It is universally supported and allows anonymous users.
- **Disadvantages**:
  - It only works for a sequence of dynamically generated forms.
  - It breaks down with static documents, emailed documents, and bookmarked documents.
  - There are no browser shut downs.

3. Cookies:

- Cookies are small bits of textual information that a Web server sends to a browser and that the browser later returns unchanged when visiting the same Web site or domain. By letting the server read information it sent the client previously, the site can provide visitors with a number of conveniences such as presenting the site the way the visitor previously customized it or letting identifiable visitors in without their having to renter a password. Recognize and receive cookies from web server and send them back along with requests.
- **All modern browser can receive cookies from web servers** and then send them back along with requests. There is no limitation with cookies. All browser allow users to disable this functionality, which leads to browser not recognizing cookies. This is because cookies have a bad press they have sometimes been used to gather information about consumer without their knowledge.
- **Identifying a user during an e-commerce session.** Many on-line stores use a ”shopping cart” in which the user selects an item, adds it to shopping cart, and then continues shopping. Since the HTTP connection is closed after each page is sent, when the user selects a new item for his cart, how does the store know that he is the same user that put the previous item in his cart? Cookies are a good way of accomplishing this. In fact, this is so useful that servlets have an API specifically for this, and servlet authors don’t need to manipulate cookies directly to make use of it.
- **Avoiding username and password.** Many large sites require you to register in order to use their services, but it is inconvenient to remember the username and password. Cookies are a good alternative for low-security sites. When a user registers, a cookie is sent with a unique user ID. When the client reconnects at a later date, the user ID is returned, the server looks it up, determines it belongs to a registered user, and doesn’t require an explicit username and password.
- **Customizing a site.** Many portal sites let you customize the look of the main page. They use cookies to remember what you wanted, so that you get that result initially next time.
- Focusing advertising. Cookies let the site remember which topics
- Interest certain users and show advertisements relevant to those interests.

The Servlet Cookie API

To send cookies to the client, a servlet would create one or more cookies with the appropriate names and values via new Cookie (name, value), set any desired optional attributes via cookie.setXxx, and add the cookies to the response headers via response. addCookie (cookie). To read incoming cookies, call request.getCookies (), which returns an array of Cookie objects. In most cases, you loop down this array until you find the one whose name (getName) matches the name you have in mind, then call getValue on that Cookie to see the value associated with that name.

Creating Cookies

A Cookie is created by calling the Cookie constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain whitespace or any of:
[ ] ( ) = ," /? @ : ;

Reading and Specifying Cookie Attributes

Before adding the cookie to the outgoing headers, you can look up or set attributes of the cookie. Here's a summary:

- **getComment/setComment**

Gets/sets a comment associated with this cookie.

public void setComment(String purpose) public String getComment()

- **getDomain/setDomain**

Gets/sets the domain to which cookie applies. Normally, cookies are returned only to the exact host name that sent them. You can use this method to instruct the browser to return them to other hosts within the same domain.

public void setDomain(String pattern) public String getDomain()

- **getMaxAge/setMaxAge**

Gets/sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session (i.e. until the user quits the browser), and will not be stored on disk. See the LongLivedCookie class below, which defines a subclass of Cookie with a maximum age automatically set one year in the future.

public void setMaxAge(int expiry)
public int getmaxAge()

- **getName/setName**

Gets/sets the name of the cookie. The name and the value are the two pieces you virtually always care about. Since the getCookies method of HttpServletRequest returns an array of Cookie objects, it is common to loop down this array until you have a particular name, then check the value with getValue. See the getCookieValue method shown below.

public void setName(String name)
public String getName()

- **getPath/setPath**

Gets/sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories. This method can be used to specify something more general. For example, someCookie.setPath("/") specifies that all pages on the server should receive the cookie. Note that the path specified must include the current directory.

public void setPath(String uri)
public String getPath()

- **getSecure/setSecure**

Gets/sets the Boolean value indicating whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.The default value is false.

public void setSecure(Boolean flag)
public Boolean getSecure()

- **getValue/setValue**

Gets/sets the value associated with the cookie. Again, the name and the value are the two parts of a cookie that you almost always care about, although in a few cases a name is used as a Boolean flag, and its value is ignored (i.e the existence of the name means true).

public void setValue(String value) public String getValue()

- **getVersion/setVersion**

Gets/sets the cookie protocol version this cookie complies with. Version 0, complies with original cookie specification drafted by Netscape. Version 1, complies with RFC 2109. Cookies provided by a browser to use and identify the browser's cookie's version.

Public void setVersion(int v)
Public int getVersion()

Placing Cookies in the Response Headers
The cookie is added to the Set-Cookie response header by means of the addCookie method of HttpServletResponse. Here's an example:

Cookie userCookie = new Cookie("user", "someuser"); response.addCookie(userCookie);

Reading all Cookies stored in Request object

```
Cookie c[] = request.getCookies();
if(c.length==0)
        out.println("<p> No Cookies in Your PC");
for(int i=0; i<c.length; i++) {
        out.println("<br> Cookie-" + (i+1));
        out.println("<br> Name : " + c[i].getName());
        out.println("<br> Value : " + c[i].getValue());
}
```

Session API
When we write any servlet, it is necessary for the developer to maintain sessions. There are several methods and classes which are specifically designed to handle the session tracking on behalf of servlet provided by the Servlet API in other words we can say servlet has in built facility to manage the session.

HTTPSessioninterface:

- A public interface HttpSession is a member of javax.servlet.http package. It provides a way to identify a user across more than one page request or visit to a web site and to store information about that user. The servlet container uses this interface to create a session between an HTTP client and an HTTP Server. The session persists for a specified time period across more than one connection or page request from the user.

- An HttpSession class was introduced in the Servlet API. Instances of this class can hold information for one user session between requests. You start a new session by requesting an HttpSession object from the HttpServletRequest in your doGet or doPost method:
  HttpSession session = request.getSession (true);
- This method takes a Boolean argument. True means a new session shall be started if none exist, while false only returns an existing session.
- An HttpSession can store any type of object. A typical example is a database connection allowing multiple requests to be part of the same database transaction, or information about purchased products in a shopping cart application so the user can add items to the cart while browsing through the site. To save an object in an HttpSession you use the putValue method:
  Connection con = DriverManager.getConnection (databaseURL, user, password);
  session.putValue ("myappl.connection", con);

The HTTPSession interface is implemented by the server. It enables a servlet to read and write the state information that is associated with an HTTPSession which is mostly useful in tracking and managing user sessions.

Method	Description
Void setMaxInactiveInterval(int interval)	It sets the maximum interval between requests that this session will be kept by the host server.
Long getCreationTime()	Returns the time in milliseconds since midnight, Jan 1, 1970 for the session has been created.
String getId()	Returns the session ID.
Object getAttribute(String attr)	Returns the value associated with the name passed in attr. Returns null if attr is not found.
Enumeration getAttributeName()	Returns an enumeration of the attribute names associated with the session.
long getLastAccessedTime()	Returns the time in milliseconds since midnight, Jan 1, 1970 when the client made a request for the session.
Int getmaxInactiveInterval()	It returns maximum time interval,in seconds that servlet container will keep this session open between client accesses.A negative time indicates that session should never time out.
Object getValue(String name)	Return the object bound to the given name in the session's application layer data.Null if no such bindings.
String[] getValueNames()	Returns array of string object specifying tha name of all objects bound to this session.
void invalidate()	Invalidates this session and removes it from context.
Boolean isNew()	Returns true if the server has created the new session and has not been used by the client otherwise return false.
Void putValue(String name,Object value)	It binds specified object into the application's layer data with the given name.existing binding with same name is replaced.
void setAtttribute(String attr Object val)	Associates the value passed in val with the attribute name passed in attr.
Void Removevalue(String name)	It removes object bound with the specified name from this session.If session does not have an object bound with the specified name, this method does nothing.

HttpSessionContext interface:

HttpSessionContext provides access to all of the currently active sessions on the server.This can be useful for servlets that weed out inactive sessions,displaystatistics, otherwise share information.A servlet obtains a HttpSessionContext object grom getSessionContext() of HttpSession.

Method	Description
Enumeration getId()	Returns enumeration that contains the session id for all the currently valid session in the context.returns empty enumeration if there are no valid sessions.
HttpSession getSession(String sessionid)	Returns the session associated with the given session identifier.a list of valid session id can be obtained from getId()

Servlet Collaboration

Sometimes servlets have to co-operate that is useful by sharing some information. This communication can be called as Sort Servlet Collaboration. The servlets which are collaborating can pass the shared information directly from one servlet to another. They can do it through the method invocation but for this approach, each servlet is required to know about the other servlet with which it is collaborating which is an unnecessary burden but java has provides us better technique for it.

Request Dispatcher Interface:-

It defines an object they receives request from the client and send them to any resources such as servlet, html file call JSP file and the server. The servlet container creates the Request Dispatcher. This is used to as Wrapper around server resource located a particular path for given by a particular name. They define two methods.

1. **void forward (ServletRequest req,ServletResponse res) throws ServletException, IOException**

It will forward a request from the server to another resource. This method allows one servlet to do preliminary processing of a request and other resources to generate the response.
forward() should be called before the response has been committed to the client but if a response has been already commented then this method throws an IlligelStateException. The request & response particular must be the object of the same class.

2. **void include (ServletRequest req, ServletResponse res) throws ServletException, IOException**

It includes the contexts of resource in the response. The ServletResponse object has its path elements & parameter remains unchanged from the caller. The included servlet can not change the response status code or if any attempt to make change will be ignore. This method throws ServerException and IOException.

Web.xml

- A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.
- The deployment descriptor is a file named web.xml. It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.
- Here is a simple web.xml example that maps all URL paths (/*) to the servlet class mysite.server.ComingSoonServlet:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
<servlet>
        <servlet-name>comingsoon</servlet-name>
        <servlet-class>mysite.server.ComingSoonServlet</servlet-class>
</servlet>
        <servlet-mapping>
                <servlet-name>comingsoon</servlet-name>
                <url-pattern>/*</url-pattern>
        </servlet-mapping>
</web-app>
```

- web.xml defines mappings between URL paths and the servlets that handle requests with those paths. The web server uses this configuration to identify the servlet to handle a given request and call the class method that corresponds to the request method (e.g. the doGet() method for HTTP GET requests).
- To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.
- The <servlet> element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters. You can declare multiple servlets using the same class with different initialization parameters. The name for each servlet must be unique across the deployment descriptor.

```
<servlet>
        <servlet-name>redteam</servlet-name>
        <servlet-class>mysite.server.TeamServlet</servlet-class>
        <init-param>
                <param-name>teamColor</param-name>
                <param-value>red</param-value> </init-param>
        <init-param>
                <param-name>bgColor</param-name>
                <param-value>#CC0000</param-value> </init-param>
</servlet>

<servlet>
        <servlet-name>blueteam</servlet-name>
        <servlet-class>mysite.server.TeamServlet</servlet-class>
        <init-param>
                <param-name>teamColor</param-name>
                <param-value>blue</param-value>
        </init-param>
        <init-param>
                <param-name>bgColor</param-name>
                <param-value>#0000CC</param-value>
        </init-param>
</servlet>
```