**Assignment** **24-06-2021**

1. Write a program to demonstrate lifecycle of servlet.

Ans): import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class LifeCycle extends HttpServlet

{

private String output;

public void init() throws ServletException

{

output = "Java training";

}

public void doGet(HttpServletRequest req,

HttpServletResponse resp)

throws ServletException, IOException

{

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

out.println(output);

}

public void destroy()

{

System.out.println("complete");

}

}

1. Write code to retrieves the query string that is contained in the request URL after the path?

Ans): request. getQueryString()

1. Write code to return the port number on which this request was received?

Ans): request.getServerPort();

1. **Can a JSP be called using a Servlet? If yes then explain how?**

**Ans): Yes, you can call a JSP page from a servlet. A JSP can be called (navigated to) in couple of ways, by doing: Servlet response's send redirect. Request dispatcher's forward.**

1. **What is the workflow of a servlet?  Explain with a block diagram with real life example.**

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

* The servlet is initialized by calling the **init()** method.
* The servlet calls **service()** method to process a client's request.
* The servlet is terminated by calling the **destroy()** method.
* Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

public void init() throws ServletException {

// Initialization code...

}

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

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

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

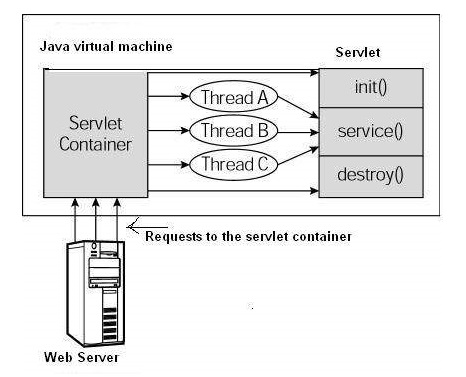
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

public void destroy() {

// Finalization code...

}

Architecture Diagram



The following figure depicts a typical servlet life-cycle scenario.

* First the HTTP requests coming to the server are delegated to the servlet container.
* The servlet container loads the servlet before invoking the service() method.
* Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

1. **What are the life cycle methods of a servlet? Explain with example.**

**Ans):** The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

public void init() throws ServletException {

// Initialization code...

}

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

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

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

public void destroy() {

// Finalization code...

}

1. What are the major components of the JDBC?

* Ans): **JDBC DriverManager:** The DriverManager class of the java.sql package manages different types of JDBC drivers. This class loads the driver classes. In addition to this, whenever a new connection establishes it chooses and loads the suitable driver from the previously loaded ones.

**Note:** From JDBC 4.0 the drivers in the CLASSPATH will be loaded automatically

* **JDBC API:** It is a Java abstraction which enables applications to communicate with relational databases. It provides two main packages namely, java.sql and javax.sql. It provides classes and methods to connect with a database, create statements (quires), execute statements and handle the results.
* **JDBC-ODBC Bridge driver:** This is a bridge driver which translates the JDBC method calls to ODBC function calls using this package you can communicate with the databases which uses ODBC drivers.

1. What is the correct sequence to create a database connection?

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

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection
6. **Write code for “Guess the number game” (team task)**

**Ans):**

**import java.util.Random;**

**import java.util.Scanner;**

**public class GuessNum {**

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

**Random Random\_number= new Random();**

**int right\_guess=Random\_number.nextInt(100);**

**int turns=0;**

**Scanner scan=new Scanner(System.in);**

**System.out.println("Guess a number between 1 to 100, You will have 10 turns!" );**

**System.out.println("best of luck!");**

**int guess;**

**int i=0;**

**boolean win=false;**

**while(win==false) {**

**guess=scan.nextInt();**

**turns++;**

**if(guess==right\_guess) {**

**win=true;**

**}**

**else if(i>8){**

**System.out.println("You loose! the right answer was: "+right\_guess);**

**return;**

**}**

**else if(guess<right\_guess){**

**i++;**

**System.out.println("Yor Guess is lower than the right guess! Turns left: "+(10-i));**

**}**

**else if(guess>right\_guess) {**

**i++;**

**System.out.println("Your Guess Is Higher Than THe Right Guess! Turns left: "+(10-i));**

**}**

**}**

**System.out.println("YOu win!");**

**System.out.println("Then number was "+right\_guess);**

**System.out.println("You used "+turns+" turns to guess the right number");**

**System.out.println("Your score is "+((11-turns)\*10)+" out of 100");**

**}**

**}**