**Life Cycle of a Servlet**

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it. So, before creating a Servlet object, let’s first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

**Stages of the Servlet Life Cycle**: The Servlet life cycle mainly goes through four stages,

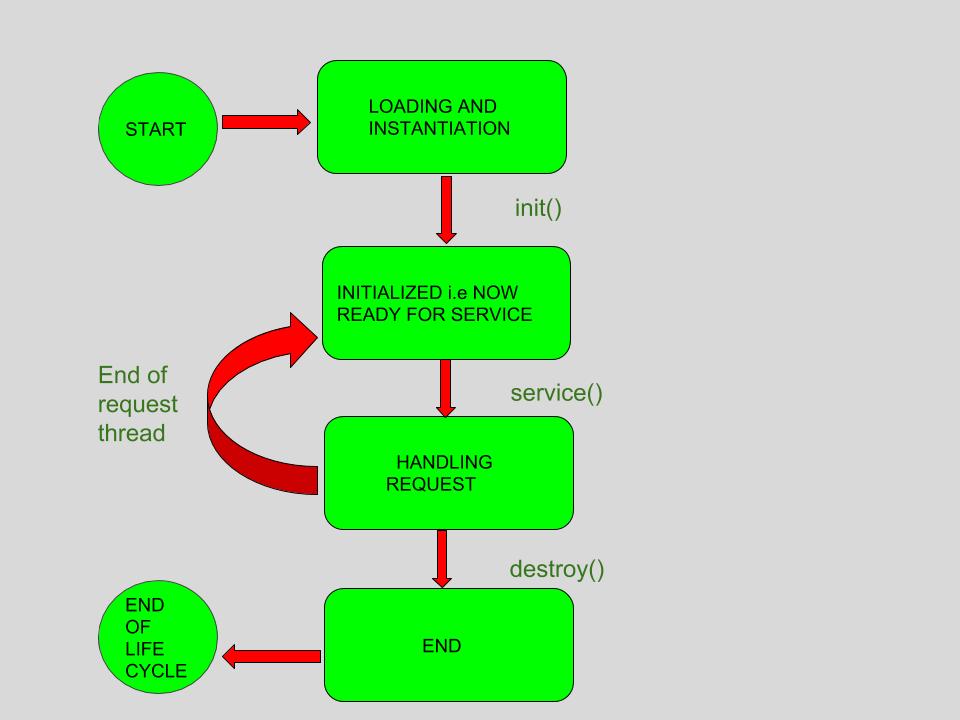
* Loading a Servlet.
* Initializing the Servlet.
* Request handling.
* Destroying the Servlet.

Let’s look at each of these stages in details:

1. **Loading a Servlet**: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :
   * Initializing the context, on configuring the Servlet with a zero or positive integer value.
   * If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

* + **Loading :** Loads the Servlet class.
  + **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



1. **Initializing a Servlet**: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

1. **Handling request**: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :
   * It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request, then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.
   * After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.

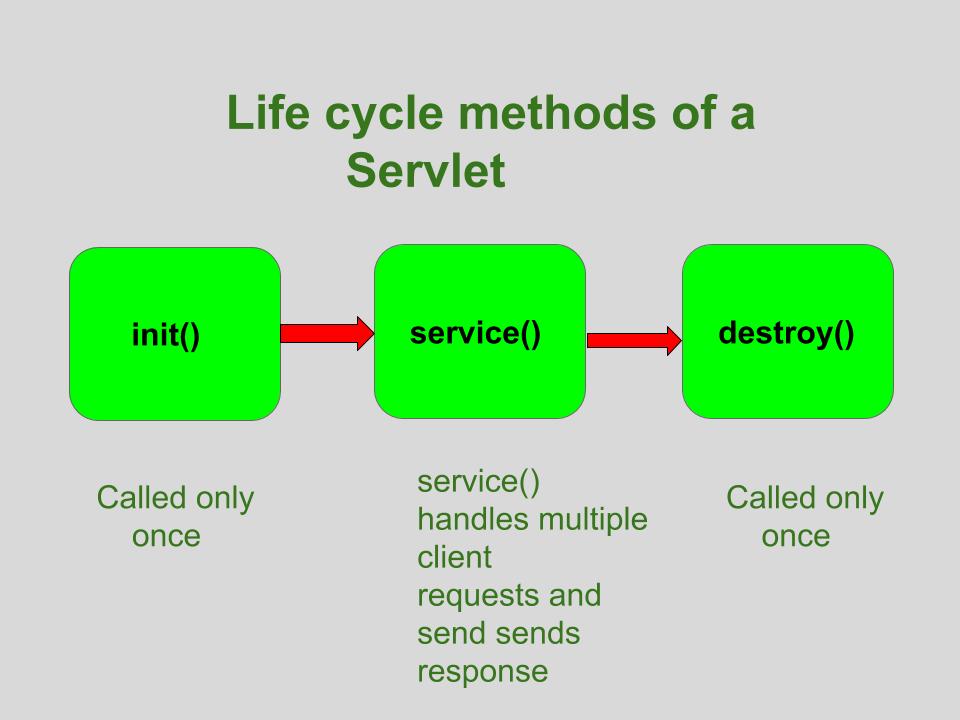
1. **Destroying a Servlet**: When a Servlet container decides to destroy the Servlet, it performs the following operations,
   * It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
   * After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

**Servlet Life Cycle Methods**

There are three life cycle methods of a Servlet :

* init()
* service()
* destroy()



Let’s look at each of these methods in details:

1. **init() method**: The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.
2. //init() method
3. public class MyServlet implements Servlet{
4. public void init(ServletConfig config) throws ServletException {
5. //initialization code
6. }
7. //rest of code
8. }
9. **service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.
   * This method uses **ServletRequest** object to collect the data requested by the client.
   * This method uses **ServletResponse** object to generate the output content.
10. // service() method
11. public class MyServlet implements Servlet{
12. public void service(ServletRequest res, ServletResponse res)
13. throws ServletException, IOException {
14. // request handling code
15. }
16. // rest of code
17. }
18. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.
19. //destroy() method
20. public void destroy()

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

**Servlet Life Cycle:**  
Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.  
The servlet life cycle consists these stages:

* Servlet is borned
* Servlet is initialized
* Servlet is ready to service
* Servlet is servicing
* Servlet is not ready to service
* Servlet is destroyed

**Life cycle methods:**  
Life cycle methods are those methods which are used to control the life cycle of the servlet. These methods are called in specific order during the servlets’s entire life cycle.  
The class **Servlet** provides the methods to control and supervise the life cycle of servlet. There are three life cycle methods in the Servlet interface. There are as follows:

* **init() method :**
  1. A servlet’s life begins here .
  2. This method is called only once to load the servlet.Since it is called only once in it’s lifetime,therefore “connected architecture” code is written inside it because we only want once to get connected with the database.  
     **Now Question Arises is that:-**  
     Q.Why can’t we write connected architecture code inside the constructor, since constructor also run only once in it’s entire life?  
     Ans. Suppose if the connection doesn’t get established, then we can throw an exception from init() and the rest of the steps stop executing. But in the constructor we can’t use, throw in it’s prototype otherwise it is an error.
  3. This method receives only one parameter, i.e **ServletConfig** object.
  4. This method has the possibility to throw the ServletException.
  5. Once the servlet is initialized, it is ready to handle the client request.
  6. The prototype for the init() method:

public void init(ServletConfig con)throws ServletException{ }

where **con** is ServletConfig object

* **NOTE:-** In programs of servlet,we use non parameterized version of init().
* **Now,Question Arises is that:-**  
  Q. Why it is recommended to use the non parameterized version of init() instead of parameterized version as seen above?  
  Ans. To answer this, we have to go into detail. Think like developers,i.e there must be some valid reason for this and the answer will blow your mind. Coming to answer:
* **APPROACH 1**  
  Whenever the lifecycle method of a servlet starts executing,i.e when public void init(ServletConfig con) throws ServletException gets call then our class public void init(ServletConfig con) throws ServletException gets called but we have to run the code which initializes servlet config object which is written inside “HttpServlet” method public void init(ServletConfig con) throws ServletException,i.e:  
  Coding of HttpServlet class be like:
* public void init(ServletConfig con) throws ServletException
* {
* //code to initialise ServletConfig object
* init(); //This HttpServlet has 2 init() one which is parameterized and the other one is non
* //parameterized.But this non parameterized version of init() has a blank body.
* //So this call is useless.
* }
* Now see the coding of our class
* public void init(ServletConfig con) throws ServletException
* {
* super.init(con); //Since,our class init() will run first,but to run HttpServlet init() we
* // have used super keyword.And Database connectivity code will be their
* }
* **NOTE:-** As we can see, total 3 init() calls we have to make.First init() gets called of our class then of HttpServlet class then non parameterized version of HttpServlet class.
* But now, we will achieve the same thing with less number of calls:
* **APPROACH 2**  
  Coding of HttpServlet parametrized and non parameterized versions of init() will remain the same. **But in our class instead of overriding parameterized version of init(), we will override non parameterized version of init()**.
* Let’s see the coding of our class non parameterized version of init():
* public void init() throws ServletException
* {
* //database connectivity code
* }
* **NOTE:** Since this method public void init() throws ServletException ,we have override from HttpServlet class whose coding is like:
* public void init() throws ServletException
* {
* //empty body
* }
* Since it’s body is blank, therefore it is known as **“Helper method”** as it is used for overriding purpose.
* Now, as the servlet starts executing its methods, it will call the parameterized version of init(). Since we have not to override the parameterized version, therefore it will give a call to the HttpServlet parameterized version of init(). Since coding of a parameterized version of init() of HttpServlet is as same as above, therefore, from there on it will call init() (i.e non parameterized version of init). It will give a call to our class non parameterized version of init() and the code continues.  
  **Now, as you can see, total number of init() calls are 2 which is less than the first approach. Therefore, execution time is less in 2nd approach and less headache for CPU for maintaining stack and it’s speed increases as compared to 1st approach.**  
  Therefore, it is highly recommended to override non parameterized version of init().Although both will run but due to efficiency first approach is rarely used and also in first approach we have to use super keyword too.Therefore in below mentioned program,we have override non parameterized version of init().
* **service() method :**
  1. The service() method is the most important method to perform that provides the connection between client and server.
  2. The web server calls the service() method to handle requests coming from the client( web browsers) and to send response back to the client.
  3. This method determines the type of Http request (GET, POST, PUT, DELETE, etc.) .
  4. This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete(), etc. as required.
  5. This method accepts two parameters.
  6. The prototype for this method:
  7. public void service(ServletRequest req, ServletResponse resp)

throws ServletException, IOException { }

where

* + - **req** is the ServletRequest object which encapsulates the connection from client to server
    - **resp** is the ServletResponse object which encapsulates the connection from server back to the client
* **destroy() method :**
  1. The destroy() method is called only once.
  2. It is called at the end of the life cycle of the servlet.
  3. This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.
  4. When this method is called, the garbage collector comes into action.
  5. The prototype for this method is:

public void destroy() { // Finalization code...}

Below is a sample program to illustrate Servlet in Java:

|  |
| --- |
| // Java program to show servlet example  // Importing required Java libraries  **import** java.io.\*;  **import** javax.servlet.\*;  **import** javax.servlet.http.\*;    // Extend HttpServlet class  **public** **class** AdvanceJavaConcepts **extends** HttpServlet  {  **private** String output;       // Initializing servlet  **public** **void** init() **throws** ServletException     {        output = "Advance Java Concepts";     }       // Requesting and printing the output  **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("Over");        }  } |