the doGet() and doPost() methods (as well as other HttpServlet methods) are called by the service() method.

Concluding, if you must respond to GET or POST requests made by a HTTP protocol client (usually a browser) don't hesitate to extend HttpServlet and use its convenience methods.

If you must respond to requests made by a client that is not using the HTTP protocol, you must use service().

doPost has no limitations on paramater numbers while doGet has.

doGet is faster than doPost.

doPost is secured than doGet

doPost has no limitations on paramater numbers while doGet has.

doGet is faster than doPost.

doPost is secured than doGet

doService(): use it when you know what you're doing, the default implementation calls doGet()or doPost() so if you overwrite it, you wont get the other method called.

doGet(): by convention, a method GET shouldn't change the internal state of your application (you shouldn't do updates or the like)

doPost(): by convention, POST is used to modify the internal state of your application (do inserts, updates, deletes)

Methods of GenericServlet class:

Here is the list of all the methods of GenericServlet class.

1. public void init(): it is a convenient method. This method can be overridden so that there’s no need to call super.init(config).

2. public void init(ServletConfig config): Called by the servlet container to indicate that the servlet is being placed into service, this method is used for initializing the servlet.

3. public String getInitParameter(String name): Returns a String containing the value of the given initialization parameter, or null if the parameter does not exist.

4. public Enumeration getInitParameterNames(): Returns the names of all the parameters defined in the web.xml file or null if web.xml does’t have any parameter.

5. public abstract void service(ServletRequest request, ServletResponse response): Called by the Servlet container to allow servlet to respond to the requests made by client.

6. public void destroy(): It is called by servlet container once at the end of servlet life cycle to indicate that servlet is being destroyed.

7. public ServletConfig getServletConfig(): Return the ServletConfig object that initialized this servlet

8. public String getServletInfo(): Returns information about servlet.

9. public ServletContext getServletContext(): Return ServletContext object, passed to this servlet by the init method

10. public String getServletName(): Returns the name of the servlet instance.

11. public void log(String msg): Writes the given message in the servlet log file.

12. public void log(String msg,Throwable t): Writes the explanatory message in the servlet log file including a String that describes the error or exception.

Methods of HttpServlet class

1. protected void doGet(HttpServletRequest req, HttpServletResponse resp): This method is called by servlet service method to handle the HTTP GET request from client. When overriding this method, read the request data, write the response headers, get the response’s writer or output stream object, and finally, write the response data.

2. protected long getLastModified(HttpServletRequest req): Returns a long integer specifying the time the HttpServletRequest object was last modified, in milliseconds since midnight, January 1, 1970 GMT, or -1 if the time is not known

3. protected void doHead(HttpServletRequest req, HttpServletResponse resp): This method is called by servlet service method to handle the HTTP HEAD request from client. The client sends a HEAD request when it wants to see only the headers of a response, such as Content-Type or Content-Length

4. protected void doPost(HttpServletRequest req, HttpServletResponse resp): This method is called by servlet service method to handle the POST request from client. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information to the server. Unlike, doGet where we get information from the sever this method is used when we are transferring information from client to the server.

5. protected void doPut(HttpServletRequest req, HttpServletResponse resp): This method is called by servlet service method to handle the PUT request from client. This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server.

6. protected void doDelete(HttpServletRequest req, HttpServletResponse resp): Called by servlet service() method to handle the DELETE request from client that allows a client to delete a document, webpage or information from the server.

7. protected void doOptions(HttpServletRequest req, HttpServletResponse resp): Called by the service method to allow a servlet to handle a OPTIONS request. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header.

8. protected void doTrace(HttpServletRequest req, HttpServletResponse resp): This method is called by service() method for handling TRACE request. Used for debugging purposes.

9. protected void service(HttpServletRequest req, HttpServletResponse resp): There is no need to override this method, this method receives the HTTP request from client and forwards them to the corresponding doXXX methods such as doGet(), doPost(), doHEAD() etc.

10. public void service(ServletRequest req, ServletResponse res): Forwards client request to the protected service method. There’s no need to override this method as well.

**How Servlet is better than CGI**

CGI programs are handled by a new process every time a new request has been made. Unlike CGI, the servlet programs are handled by separate threads that can run concurrently more efficiently.

CGI program can be written in any programming language that makes it mostly platform dependent as not all programming languages are platform independent. Servlet only uses Java as programming language that makes it platform independent and portable. Another benefit of using java is that the servlet can take advantage of the object oriented programming features of java.

## Life Cycle of Servlet

Servlet life cycle contains five steps: 1) Loading of Servlet 2) Creating instance of Servlet 3) Invoke init() once 4) Invoke service() repeatedly for each client request 5) Invoke destroy()

For those who are wondering what is instance and invoke means: Instance and objects are same thing. Invoking a method means calling a method, it is just a fancy word that we use in programming world in place of calling :)

Let’s back to the main topic. Here are the five steps of servlet life cycle.

**Step 1: Loading of Servlet**  
When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.

**Step 2: Creating instance of Servlet**  
Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

**Step 3: Invoke init() method**  
Once all the servlet classes are instantiated, the init() method is invoked for each instantiated servlet. This method initializes the servlet. There are certain init parameters that you can specify in the deployment descriptor (web.xml) file. For example, if a servlet has value >=0 then its init() method is immediately invoked during web container startup.

**Note: The init() method is called only once during the life cycle of servlet.**

**Step 4: Invoke service() method**  
Each time the web server receives a request for servlet, it spawns a new thread that calls service() method. If the servlet is GenericServlet then the request is served by the service() method itself, if the servlet is HttpServlet then service() method receives the request and dispatches it to the correct handler method based on the type of request.

**Note**: Unlike init() and destroy() that are called only once, the service() method can be called any number of times during servlet life cycle. As long as servlet is not destroyed, for each client request the service() method is invoked.

**Out of all the 5 steps in life cycle, this is the only step that executes multiple times.**

**Step 5: Invoke destroy() method**  
When servlet container shuts down(this usually happens when we stop the web server), it unloads all the servlets and calls destroy() method for each initialized servlets.

The HttpSession object is used for session management. A session contains information specific to a particular user across the whole application. When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session. This unique ID can be stored into a cookie or in a request parameter.

The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file( web.xml). The default timeout value is 30 minutes, this is used if you don’t specify the value in tag. This means that when the user doesn’t visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

This is how you create a HttpSession object.

protected void doPost(HttpServletRequest req,

HttpServletResponse res)

throws ServletException, IOException {

HttpSession session = req.getSession();

}

You can store the user information into the session object by using setAttribute() method and later when needed this information can be fetched from the session.

session.setAttribute("uName", "ChaitanyaSingh");

session.setAttribute("uemailId", "myemailid@gmail.com");

session.setAttribute("uAge", "30");

TO get the value from session we use the getAttribute() method of HttpSession interface. Here we are fetching the attribute values using attribute names.

String userName = (String) session.getAttribute("uName");

String userEmailId = (String) session.getAttribute("uemailId");

String userAge = (String) session.getAttribute("uAge");

A unique Id is assigned to the session. The **Servlet container also sets a Cookie in the header of the HTTP response with cookie name and the unique session ID as its value.**

**The Servlet container checks the request header for cookies and get the session information from the cookie and use the associated session from the server memory.**

**Cookie remains active as long as the user’s browser is running**, as soon as the browser is closed, the cookie and associated session info is destroyed. So when the user opens the browser again and sends request to web server, the new session is being created.

## Types of Cookies

We can classify the cookie based on their expiry time:

1. Session
2. Persistent

**1) SessionCookies:**  
Session cookies do not have expiration time. It lives in the browser memory. As soon as the web browser is closed this cookie gets destroyed.

**2) Persistent Cookies:**  
Unlike Session cookies they have expiration time, they are stored in the user hard drive and gets destroyed based on the expiry time.

## How to send Cookies to the Client

Here are steps for sending cookie to the client:

1. Create a Cookie object.
2. Set the maximum Age.
3. Place the Cookie in HTTP response header.

#### 1) Create a Cookie object:

Cookie c = new Cookie("userName","Chaitanya");

#### 2) Set the maximum Age:

By using **setMaxAge ()** method we can set the maximum age for the particular cookie in seconds.

c.setMaxAge(1800);

#### 3) Place the Cookie in HTTP response header:

We can send the cookie to the client browser through response.addCookie() method.

response.addCookie(c);

## How to read cookies

Cookie c[]=request.getCookies();

//c.length gives the cookie count

for(int i=0;i<c.length;i++){

out.print("Name: "+c[i].getName()+" & Value: "+c[i].getValue());

}