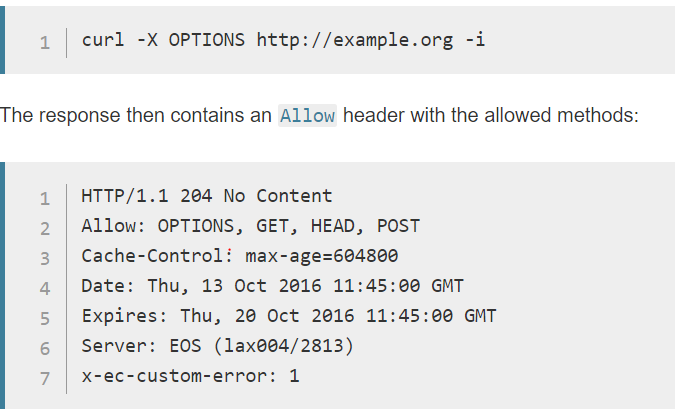
# HTTP Methods and Descriptions :

Method Name

Description

**OPTIONS** Request for communication options that are available on the request/response chain.



**GET** Request to retrieve information from server using a given URI.

**HEAD** Identical to GET except that it does not return a message-body, only the headers and status line.

**POST** Request for server to accept the entity enclosed in the body of HTTP method.

**DELETE** Request for the Server to delete the resource.

**PUT** This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content.

**CONNECT** Reserved for use with a proxy that can switch to being a tunnel.

**HTTP tunneling** is used to create a network link between two computers in conditions of restricted network connectivity including [firewalls](https://en.wikipedia.org/wiki/Firewall_(computing)), [NATs](https://en.wikipedia.org/wiki/Network_address_translation) and [ACLs](https://en.wikipedia.org/wiki/Access_control_list#Networking_ACLs)

The most common form of HTTP tunneling is the standardized [HTTP CONNECT](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) method.[[1]](https://en.wikipedia.org/wiki/HTTP_tunnel#cite_note-1)[[2]](https://en.wikipedia.org/wiki/HTTP_tunnel#cite_note-2) In this mechanism, the client asks an HTTP proxy server to forward the [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) connection to the desired destination. The server then proceeds to make the connection on behalf of the client. Once the connection has been established by the server, the proxy server continues to proxy the TCP stream to and from the client. Only the initial connection request is HTTP - after that, the server simply proxies the established TCP connection.

**HEADERS**:

HTTP Headers are an important part of the API request and response as they represent the meta-data associated with the API request and response. Headers carry information for:

* Request and Response Body
* Request Authorization
* Response Caching
* Response Cookies

# URL vs. URI vs. URN

# URI

A URI identifies a resource either by location, or a name, or both. More often than not, most of us use URIs that defines a location to a resource

A URI has two specializations known as URL and URN

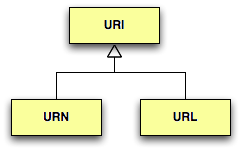
# URN

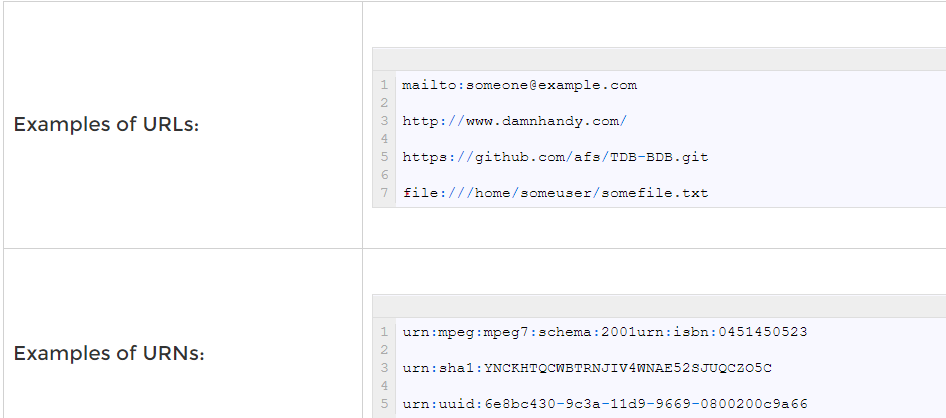
A URI identifies a resource by name in a given namespace but not define how the resource maybe obtained. This type of URI is called a URN

# URL

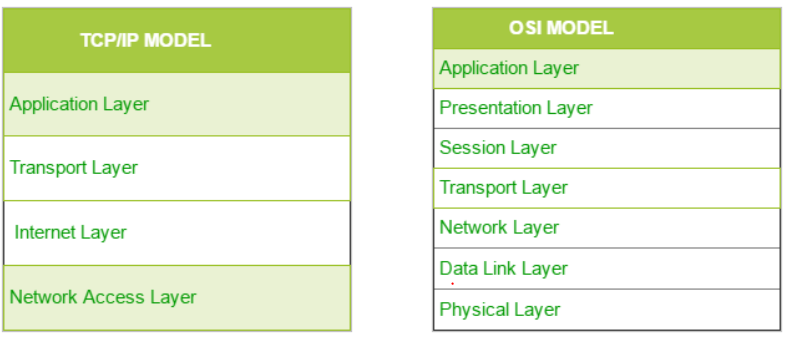
A URL is a specialization of URI that defines the network location of a specific resource. Unlike a URN, the URL defines how the resource can be obtained. We use URLs every day in the form of http://exemple.com, etc. But a URL doesn’t have to be an HTTP URL, it can be ftp://example.com, smb://example.com, etc.

Here is a diagram that shows the relationship between the three terms.





# TCP/IP Model



# web container

A **web container** is the component of a web server that interacts with Java servlets. A web container manages the life cycle of servlets; it maps a URL to a particular servlet while ensuring that the requester has relevant access-rights.

The web container implements the web component aspect of the Java engineering architecture; it specifies a run time environment for various components such as security, concurrency, transaction, and deployment.

Java servlets do not have a defined main() method, so a container is required to load them. The servlet gets deployed on the container.

Let’s have a look at what happens when a client sends a certain request that requires interaction with the servlet:

* The client sends a request to a web server.
* The web server, which contains a servlet, sends that request to the container.
* The container passes the request to the respective servlet.
* The servlet methods are loaded.
* The servlet hands over the relevant response to the container, which passes it to the server. Eventually, the response reaches the client.

# Servlets

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

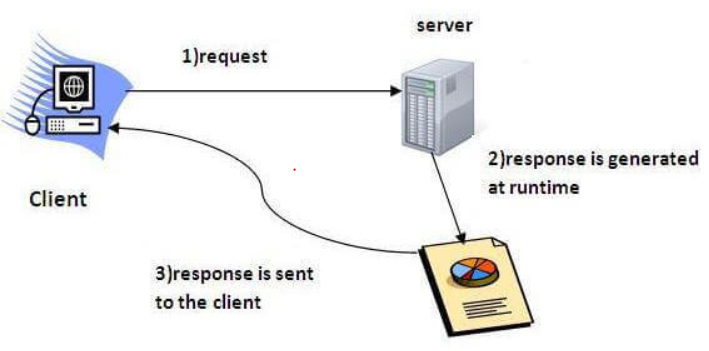
**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

# What is a Servlet?

Servlet can be described in many ways, depending on the context.

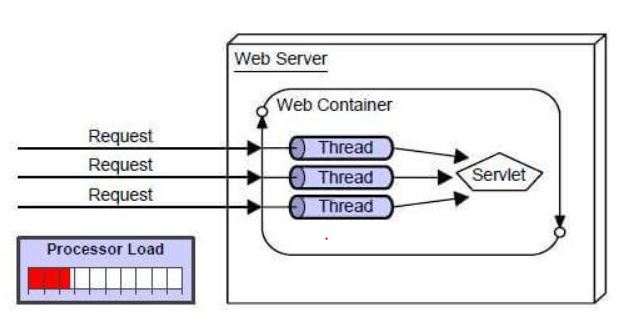
* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page



# What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

# Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) manages Servlets, so we don't need to worry about the memory leak, [garbage collection](https://www.javatpoint.com/Garbage-Collection), etc.
4. **Secure:** because it uses java language.

# Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

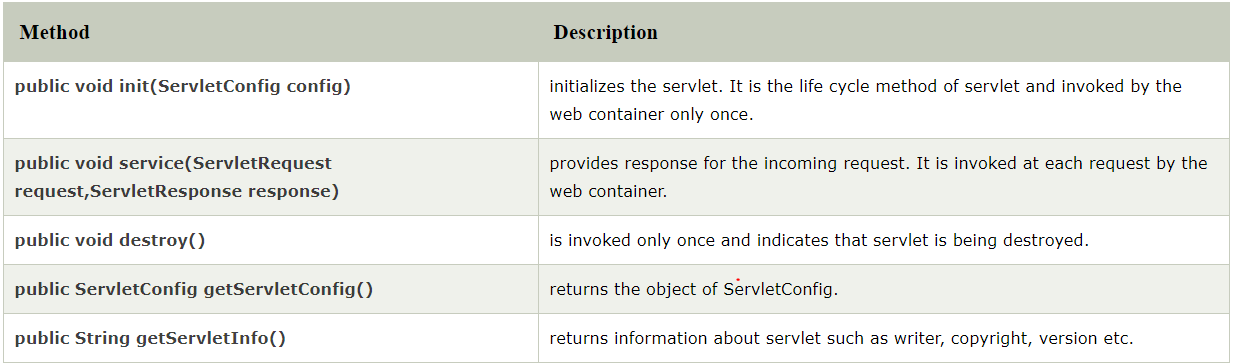
The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

# Servlet Interface

**Servlet interface provides** commonbehavior to all the servlets.Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.



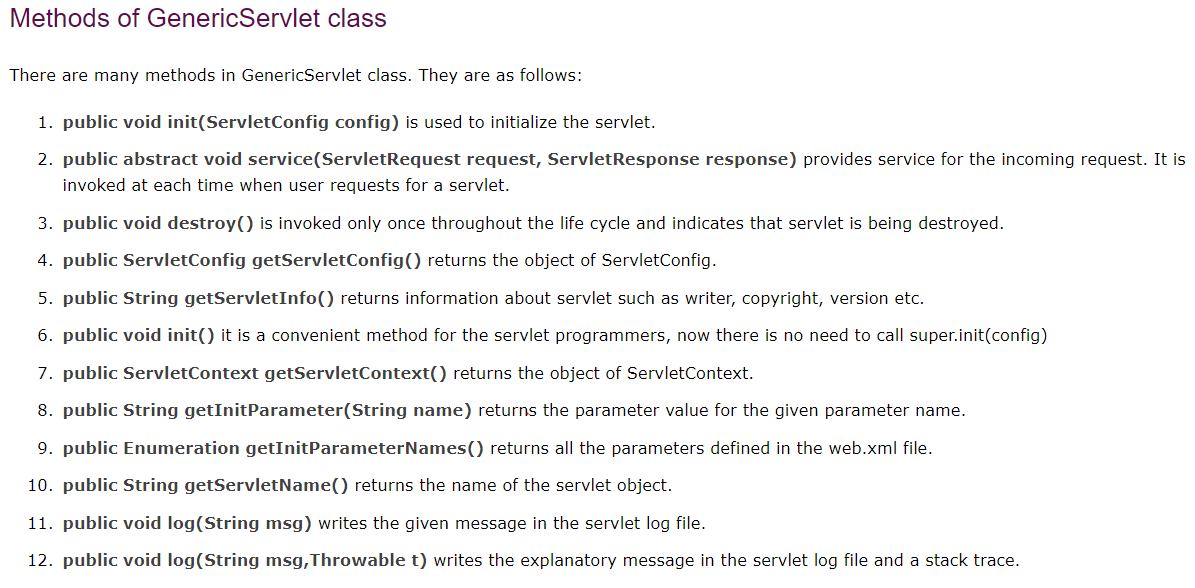


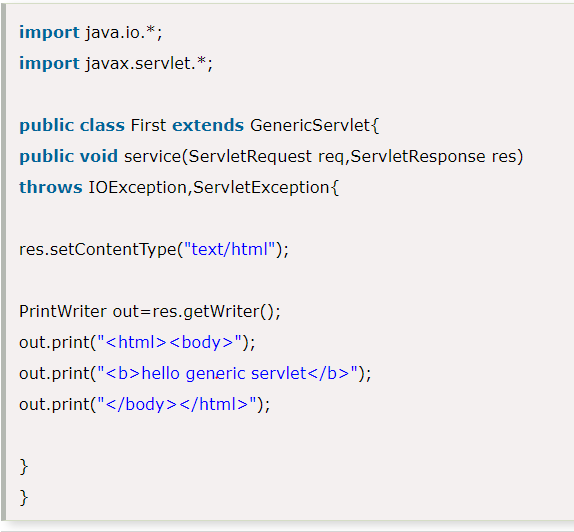
# GenericServlet class

**GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.





# HttpServlet class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The HttpServlet class extends the GenericServlet class and implements Serializable interface.  It provides http specific methods such as doGet, doPost, doHead, doTrace etc.    There is a possibility of developing ‘n’ types of servlets, like httpservlet, ftpservlet, smtpservlet etc. for all these protocol specific servlet classes GenericServlet is the common super class containing common properties and logics. So, GenericServlet is not a separate type of servlet.  to use different protocols extend the genericservlet and implement protocol based methods and conversions Life Cycle of a Servlet (Servlet Life Cycle) The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:   1. Servlet class is loaded. 2. Servlet instance is created. 3. init method is invoked. 4. service method is invoked. 5. destroy method is invoked.  1) Servlet class is loaded The classloader is responsible to load the servlet class. The servlet class is loaded when the first request  for the servlet is received by the web container 2) Servlet instance is created The web container creates the instance of a servlet after loading the servlet class. The servlet instance is  created only once in the servlet life cycle. 3) init method is invoked  |  |  |  |  | | --- | --- | --- | --- | | The web container calls the init method only once after creating the servlet instance. The init method  is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. 4) service method is invoked The web container calls the service method each time when request for the servlet is received. If servlet  is not initialized, it follows the first three steps as described above then calls the service method.  If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. 5) destroy method is invoked The web container calls the destroy method before removing the servlet instance from the service.  It gives the servlet an opportunity to clean up any resource for example memory, thread etc. Hot to deploy basic servlet: <https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server> **How Servlet works?** The server checks if the servlet is requested **for the first time**.  **If yes,** web container does the following tasks:   * loads the servlet class. * instantiates the servlet class. * calls the init method passing the ServletConfig object   **else**   * calls the service method passing request and response objects   The web container calls the destroy method when it needs to remove the servlet such as at time of  stopping server or undeploying the project. **How web container handles the servlet request?** The web container is responsible to handle the request. Let's see how it handles the request.   * maps the request with the servlet in the web.xml file. * creates request and response objects for this request * calls the service method on the thread * The public service method internally calls the protected service method * The protected service method calls the doGet method depending on the type of request. * The doGet method generates the response and it is passed to the client. * After sending the response, the web container deletes the request and response objects.   The thread is contained in the thread pool or deleted depends on the server implementation.       **welcome-file-list in web.xml** The **welcome-file-list** element of **web-app**, is used to define a list of welcome files. Its sub element is  **welcome-file** that is used to define the welcome file.  A **welcome file** is the file that is invoked automatically by the server, if you don't specify any file name.  By default server looks for the welcome file in following order:   1. welcome-file-list in web.xml 2. index.html 3. index.htm 4. index.jsp   If none of these files are found, server renders 404 error.   **load on startup in web.xml** The **load-on-startup** element of **web-app** loads the servlet at the time of deployment or server start if value is positive. It is also  known as **pre initialization of servlet**.  You can pass positive and negative value for the servlet. **Advantage of load-on-startup element** As you know well, servlet is loaded at first request. That means it consumes more time at first request  . If you specify the load-on-startup in web.xml, servlet will be loaded at project deployment time or  server start. So, it will take **less time** for responding to first request. **Passing positive value** If you pass the positive value, the lower integer value servlet will be loaded before the higher integer  value servlet. In other words, container loads the servlets in ascending integer value. The 0 value will be  loaded first then 1, 2, 3 and so on.   **Passing negative value** If you pass the negative value, servlet will be loaded at request time, at first request. **ServletRequest Interface** An object of ServletRequest is used to provide the client request information to a servlet such as content  type, content length, parameter names and values, header informations, attributes etc.   Servlet Response:Method of ServletResponse interface 1) String getCharacterEncoding(): It returns the name of the MIME charset used in body of the response sent to the client. 2) String getContentType(): It returns the response content type. e.g. text, html etc. 3) ServletOutputStream getOutputStream(): Returns a ServletOutputStream suitable for writing binary data in the response. 4) java.io.PrintWriter getWriter(): Returns the PrintWriter object. 5) void setCharacterEncoding(java.lang.String charset): Set the MIME charset (character encoding) of the response. 6) void setContentLength(int len): It sets the length of the response body. 7) void setContentType(java.lang.String type): Sets the type of the response data. 8) void setBufferSize(int size): Sets the buffer size. 9) int getBufferSize(): Returns the buffer size. 10) void flushBuffer(): Forces any content in the buffer to be written to the client. 11) boolean isCommitted(): Returns a boolean indicating if the response has been committed. 12) void reset(): Clears the data of the buffer along with the headers and status code.   RequestDispatcher in Servlet The RequestDispatcher interface provides the facility of dispatching the request to another resource it  may be html, servlet or jsp. This interface can also be used to include the content of another resource  also. It is one of the way of servlet collaboration.  There are two methods defined in the RequestDispatcher interface.   1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. 2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.      SendRedirect in servlet The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.  It accepts relative as well as absolute URL.  It works at client side because it uses the url bar of the browser to make another request. So, it can  work inside and outside the server.      The main difference between a redirection and a request dispatching is that, redirection makes the client(browser) create a new request to get to the resource, the user can see the new URL while request dispatch get the resource in same request and URL does not changes.  Also, another very important difference is that, sendRedirect() works on response object while request dispatch work on request object. ServletConfig Interface An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.  If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time. **Methods of ServletConfig interface**  1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name. 2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names. 3. **public String getServletName():**Returns the name of the servlet. 4. **public ServletContext getServletContext():**Returns an object of ServletContext.      ServletContext Interface An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.  If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element. **Advantage of ServletContext** **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem. **Usage of ServletContext Interface** There can be a lot of usage of ServletContext object. Some of them are as follows:   1. The object of ServletContext provides an interface between the container and servlet. 2. The ServletContext object can be used to get configuration information from the web.xml file. 3. The ServletContext object can be used to set, get or remove attribute from the web.xml file. 4. The ServletContext object can be used to provide inter-application communication  **Commonly used methods of ServletContext interface**  |  | | --- | | There is given some commonly used methods of ServletContext interface.   1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name. 2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters. 3. **public void setAttribute(String name,Object object):**sets the given object in the application scope. 4. **public Object getAttribute(String name):**Returns the attribute for the specified name. 5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects. 6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |        **Attribute in Servlet** An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:   1. request scope 2. session scope 3. application scope   The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again **Attribute specific methods of ServletRequest, HttpSession and ServletContext interface**  |  | | --- | | There are following 4 attribute specific methods. They are as follows:   1. **public void setAttribute(String name,Object object):**sets the given object in the application scope. 2. **public Object getAttribute(String name):**Returns the attribute for the specified name. 3. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects. 4. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.   The **getQueryString()** method is defined in the **HttpServletRequest** interface, which is used to retrieve the query string of the HTTP request. A query string is the string on the URL to the right of the path to the [**servlet**](https://javainterviewpoint.com/category/servlets/). Using this a programmer can know the data which is sent from the client(when a form is submitted) **What is a Query String?** A Query String is a String which is appeneded to the URL containing the form fields and data which is entered by the user. It will start with a **‘?’**and the fields are seperated by **‘&’**  http://localhost:8080/ServletsTutorial/QueryStringExample?firstName=Java&lastName=InterviewPoint  Here we can see that there are two Form fields firstName and lastName appended as query string as values and two fields are seperated by **&.** Session Tracking in Servlets **Session** simply means a particular interval of time.  **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.  Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular  user.  HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:   Why use Session Tracking? **To recognize the user** It is used to recognize the particular user. Session Tracking Techniques There are four techniques used in Session tracking:   1. **Cookies** 2. **Hidden Form Field** 3. **URL Rewriting** 4. **HttpSession**  **Cookies in Servlet** A **cookie** is a small piece of information that is persisted between the multiple client requests.  A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number **How Cookie works** By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.   Types of Cookie There are 2 types of cookies in servlets.   1. Non-persistent cookie 2. Persistent cookie  Non-persistent cookie It is **valid for single session** only. It is removed each time when user closes the browser. Persistent cookie It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout. **Advantage of Cookies**  1. Simplest technique of maintaining the state. 2. Cookies are maintained at client side.  Disadvantage of Cookies  1. It will not work if cookie is disabled from the browser. 2. Only textual information can be set in Cookie object.  **Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work**    2) Hidden Form Field In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.  In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.  Let's see the code to store value in hidden field. Advantage of Hidden Form Field  1. It will always work whether cookie is disabled or not.  Disadvantage of Hidden Form Field:  1. It is maintained at server side. 2. Extra form submission is required on each pages. 3. Only textual information can be used.    3)URL Rewriting In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:  url?name1=value1&name2=value2&??  A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.   **Advantage of URL Rewriting**  1. It will always work whether cookie is disabled or not (browser independent). 2. Extra form submission is not required on each pages.  **Disadvantage of URL Rewriting**  1. It will work only with links. 2. It can send Only textual information.    4) HttpSession interface In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:   1. bind objects 2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.      1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container. 2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from. 3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.         how to configure timeout to 15 minutes in web.xml?  <session-config>  <session-timeout>15</session-timeout>  </session-config>  or(Call method session.setMaxInactiveInterval() to set the session timeout.) | |  | | |